INTRODUCTION ABOUT THE PROJECT

In today's digitally connected educational landscape, the need for effective communication and knowledge sharing within academic institutions has become paramount. While traditional methods of campus interaction—such as notice boards and classroom discussions—remain valuable, they often lack the immediacy and breadth required by modern students. The rise of social platforms has transformed how information is consumed, yet many generic social media sites fail to offer a focused, academic-centric environment free from distractions. This is where **Chetan College Forum** comes in.

Chetan College Forum bridges the gap between students, peers, and knowledge by offering a comprehensive, feature-rich discussion platform. Designed with a focus on community building, scalability, and user engagement, the platform provides a streamlined approach to collaborative learning. Students can explore various discussion "Spaces" across categories such as Technology, Business, Science, and Music, with features like personalized topic feeds, advanced tagging, and the ability to vote on content. Users can create engaging topics, conduct polls to gather opinions, and participate in meaningful debates through a robust commenting system.

The goal of Chetan College Forum is to create a one-stop solution that fosters a vibrant campus community while assisting individuals in solving academic queries efficiently. By leveraging modern web technologies and best practices, the platform not only provides a seamless user experience but also ensures that high-quality, relevant content rises to the top through community moderation (upvotes and downvotes). This democratized approach ensures that the most helpful answers and discussions are easily accessible to all.

As educational trends shift towards hybrid and digital-first learning, Chetan College Forum is built to meet these evolving needs. Its responsive design, dynamic search capabilities, and interactive

features—such as real-time polls and rich-text editing—make it a powerful tool for users seeking to expand their knowledge base beyond the classroom.

From a technical perspective, the platform's development is rooted in modern full-stack web development principles, utilizing the **MERN Stack (MongoDB, Express.js, React.js, and Node.js)** to create a scalable, single-page application (SPA). The project follows an Agile development approach, ensuring features like authentication, poll creation, and space management are implemented iteratively. The back-end architecture is built using a modular Controller-Service pattern with RESTful APIs, ensuring security, maintainability, and ease of future enhancements.

By combining ease of use, performance, and a personalized community experience, Chetan College Forum seeks to stand out as a premier digital hub for student interaction. Whether an individual is looking for technical help, career advice, or simply wishes to discuss shared interests, Chetan College Forum offers the tools and resources needed to succeed in a collaborative academic environment.

OBJECTIVE AND SCOPE OF PROJECT

The **Chetan College Forum** has the following specific objectives:

- **To foster a collaborative academic environment** where students can discuss various subjects, share knowledge, and solve queries collectively across dedicated "Spaces" (e.g., Technology, Science, Business).
- **To democratize knowledge sharing** by implementing a community-driven moderation system (upvotes and downvotes), ensuring that high-quality and relevant answers rise to the top.
- **To enhance user engagement** through interactive features such as real-time polls, rich-text topic creation, and a robust commenting system.
- **To provide a personalized user experience** by allowing students to follow peers, track their own contributions (topics/answers), and customize their profiles with avatars and bios.
- **To facilitate easy information retrieval** through advanced search functionality and filtering options, allowing users to find specific topics or browse by category (Space) and tags.
- **To ensure accessibility and ease of use** via a responsive, mobile-friendly interface that works seamlessly across desktops, tablets, and smartphones.

The scope for developing the **Chetan College Forum** includes:

- **User Management:** Implementing secure authentication (Login/Register) and comprehensive profile management, including tracking user statistics like "Top Contributors" and "Top Helpers."
- **Content Management System:** Developing a robust system for users to create, edit, and delete topics, as well as participate in polls and discussions.

- **Interactive Community Features:** integrating a voting system (upvote/downvote) for topics and comments to highlight valuable contributions.
- **Advanced Search & Filtering:** Creating algorithms to sort content by "Latest," "Popular," "Most Replied," or "Most Upvoted," and filtering content by specific Spaces.
- **Responsive User Interface:** Utilizing the **React-Bootstrap** framework to ensure the platform adapts dynamically to different screen sizes.
- **Data Security:** Implementing secure backend architecture using Node.js and MongoDB to protect user data and ensure safe interactions.

INTRODUCTION OF HTML AND CSS

HTML (Hypertext Markup Language) and **CSS (Cascading Style Sheets)** are the foundational technologies of web development. In the context of **Chetan College Forum**, these technologies are utilized within the React.js framework (via JSX) to build a dynamic and visually engaging Single Page Application (SPA).

HTML (Hypertext Markup Language): HTML serves as the structural backbone of the application. In this project, HTML is implemented using **JSX (JavaScript XML)**, a syntax extension for React. This allows us to write HTML structures directly within JavaScript code, defining the layout of components such as the Header, Sidebar, Topic Cards, and Polls. It categorizes elements like input forms, buttons, text blocks, and navigation links, providing the semantic structure necessary for accessibility and SEO.

CSS (Cascading Style Sheets) & Bootstrap: CSS is responsible for the visual presentation of the platform. It controls the layout, colors, fonts, and spacing, ensuring a pleasing aesthetic.

- **Custom CSS:** The project utilizes custom stylesheets (`App.css`, `poll.css`, `Responsive.css`) to define specific visual identities, such as the card layouts for topics and the styling of voting buttons.
- **Bootstrap Framework:** To accelerate development and ensure responsiveness, the project integrates **Bootstrap** (via `react-bootstrap`). This framework provides pre-styled components and a grid system that automatically adjusts the forum's layout based on the user's device, ensuring a consistent experience whether on a laptop or a mobile phone.

Together, HTML (via JSX) and CSS (enhanced by Bootstrap) enable the rapid development of a user-friendly interface that separates content structure from visual design, adhering to modern web standards.

INTRODUCTION OF JAVASCRIPT (FULL STACK)

JavaScript is a versatile and powerful programming language that serves as the core technology for the entire **Chetan College Forum** stack. Unlike traditional web development where JavaScript was limited to the browser, this project utilizes **Full Stack JavaScript**, employing it on both the client-side (React.js) and the server-side (Node.js).

Key Features of JavaScript in this Project:

Unified Language (Full Stack Development): One of the most significant advantages of this project is the use of JavaScript across the entire technology stack (MERN). This allows for seamless data flow (JSON) between the frontend and backend, reducing context switching and simplifying the development process.

Client-Side Scripting (React.js): On the frontend, JavaScript is used to build interactive user interfaces. It handles:

- **DOM Manipulation:** Updating the view dynamically without reloading the page (e.g., when a user votes on a poll or posts a comment).
- **State Management:** Using tools like **Redux Toolkit** to manage application state (user authentication, fetched topics, UI loading states).
- **Event Handling:** Responding to user actions such as button clicks, form submissions, and keyboard shortcuts (e.g., Ctrl+Enter to submit).

Server-Side Scripting (Node.js): On the backend, JavaScript runs on the server via **Node.js**. It handles:

- **API Logic:** Processing requests from the frontend (e.g., fetching topics, registering users).

- **Database Interaction:** Communicating with MongoDB via **Mongoose** to store and retrieve data.
- **Authentication:** Verifying user identities and managing secure sessions (JWT).

Asynchronous Programming: The project heavily relies on JavaScript's asynchronous features (`async/await`, `Promises`). This ensures non-blocking operations, such as waiting for database queries or API responses without freezing the user interface, resulting in a highly performant application.

Object-Oriented & Functional Paradigms: JavaScript supports both object-oriented and functional programming styles. This project utilizes modern **ES6+ features** (Arrow Functions, Destructuring, Modules), allowing for clean, modular, and maintainable code structure across both the client and server environments.

INTRODUCTION OF NODE.JS

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. Initially created by Ryan Dahl in 2009, Node.js was developed to push JavaScript beyond the client-side, allowing developers to use it for server-side scripting. Unlike PHP, which is a blocking, synchronous language, Node.js is built on the V8 JavaScript engine—the same high-performance engine that powers Google Chrome—converting JavaScript directly into native machine code.

Node.js operates on an event-driven, non-blocking I/O model, which makes it lightweight and efficient. Instead of creating a new thread for every user request (as traditional web servers do), Node.js uses a single-threaded event loop to handle thousands of concurrent connections. This architecture makes it uniquely suited for data-intensive, real-time applications like **Chetan College Forum**, where features such as live polls and instant updates are crucial.

While Node.js is primarily associated with backend web development, it is a versatile environment capable of powering command-line tools, desktop applications, and Internet of Things (IoT) devices. In the context of the MERN stack, Node.js serves as the backbone, hosting the Express.js framework and managing interactions between the React frontend and the MongoDB database.

The Node.js ecosystem is supported by the Node Package Manager (NPM), which is the largest software registry in the world. This vast library of reusable packages allows developers to integrate complex functionalities—such as authentication, image processing, and database connectivity—without reinventing the wheel.

As of 2024, Node.js is the technology of choice for high-traffic platforms like Netflix, Uber, and LinkedIn, proving its reliability and scalability. By allowing the use of a single language (JavaScript)

on both the client and server sides, Node.js unifies web application development, streamlining the coding process and reducing the learning curve for developers.

Features of Node.js:

Asynchronous and Event-Driven: All APIs of the Node.js library are asynchronous, meaning they are non-blocking. A Node.js-based server never waits for an API to return data. The server moves to the next API after calling it, and a notification mechanism helps the server receive a response from the previous API call. This ensures high throughput and speed.

High Performance: Built on Google Chrome's V8 JavaScript Engine, Node.js compiles JavaScript code directly into native machine code. This results in incredibly fast execution speeds, making it superior for tasks requiring high computation or frequent database operations.

Single-Threaded but Scalable: Node.js uses a single-threaded model with event looping. This mechanism helps the server respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. This allows **Chetan College Forum** to handle a large number of students simultaneously without crashing.

Unified Tech Stack (Full Stack JavaScript): One of the distinct advantages of Node.js is that it allows developers to write JavaScript for both the client-side (Frontend) and server-side (Backend). This eliminates the need to context-switch between languages (like PHP and JS), leading to cleaner, more consistent code and faster development cycles.

Cross-Platform: Like PHP, Node.js is completely cross-platform. It can run seamlessly on Windows, Linux, macOS, and Unix systems. This ensures that the application can be deployed on virtually any server environment without modification.

No Buffering: Node.js applications output the data in chunks rather than buffering the entire data set. This is particularly useful for features like uploading large images or streaming video content, providing a smoother user experience.

Active Community and NPM: Node.js boasts a massive, active community of developers. Through NPM (Node Package Manager), developers have access to millions of open-source libraries and modules. This community support ensures that security patches, updates, and new features are continuously available, keeping the platform secure and modern.

JSON Support: Since Node.js is JavaScript-based, it handles JSON (JavaScript Object Notation) natively. This is a significant advantage over other backend languages, as JSON is the standard format for API communication (REST APIs) and is also the native storage format of MongoDB, ensuring seamless data flow throughout the application..

INTRODUCTION OF MONGODB

What is a Database? A database is a specialized application designed to store, manage, and organize data in a structured manner. It provides distinct APIs that allow users to create, access, manage, search, and replicate the data it contains. While traditional systems used tabular data structures, modern web applications like **Chetan College Forum** often require more flexible data storage solutions to handle complex, unstructured data efficiently.

NoSQL Database Management System Unlike Relational Database Management Systems (RDBMS) which organize data into rigid tables, MongoDB is a **NoSQL (Not Only SQL)** database. It is classified as a **Document-Oriented Database**. Instead of rows and columns, it stores data in flexible, JSON-like documents. This structure allows for:

- **Dynamic Schemas:** Data structures can change over time without modifying the entire database.
- **High Scalability:** Designed to handle massive amounts of data across many servers (horizontal scaling).
- **Hierarchical Data Storage:** Complex data structures (like arrays and nested objects) can be stored within a single document, reducing the need for complex joins.

MongoDB Terminology Before exploring the specifics of MongoDB, it is important to understand the key terminology used in Document-Oriented databases and how they differ from traditional RDBMS:

- **Database:** A physical container for collections. Each database gets its own set of files on the file system.
- **Collection:** A collection is a grouping of MongoDB documents. It is the equivalent of an RDBMS **Table**. A collection exists within a single database.

- **Document:** A document is a set of key-value pairs. It is the equivalent of an RDBMS **Row** (or record). Documents have dynamic schemas, meaning documents in the same collection do not need to have the same set of fields.
- **Field:** A field is a key-value pair in a document. It is the equivalent of an RDBMS **Column**.
- **_id:** The primary key for every document. MongoDB automatically assigns a unique `_id` field to every document to ensure it can be uniquely identified.
- **BSON:** Binary JSON. This is the format MongoDB uses to store documents. It extends the JSON model to provide additional data types and to be efficient for encoding and decoding.
- **Embedding:** The practice of nesting data (such as comments inside a topic) directly within a document. This improves read performance by keeping related data together.
- **Sharding:** The process of storing data records across multiple machines to support big data deployments.

What is BSON and MQL? While SQL is used for relational databases, MongoDB uses **MQL (MongoDB Query Language)** and stores data in **BSON (Binary JSON)**.

BSON (Binary JSON): BSON is a binary-encoded serialization of JSON-like documents. It supports embedding of documents and arrays within other documents and records. BSON also contains extensions that allow representation of data types that are not part of the JSON spec, such as `Date` and `BinData`.

CRUD Operations (The Foundation of MQL): Similar to the DML (Data Manipulation Language) in SQL, MongoDB provides powerful operators to interact with data:

1. **Create:** `insertOne()`, `insertMany()` – Used to add new documents to a collection.

2. **Read:** `find()`, `findOne()` – Used to query data. MongoDB supports rich queries, including searching by field, range queries, and regular expression searches.
3. **Update:** `updateOne()`, `updateMany()` – Used to modify existing documents. It supports atomic operators like `$set`, `$inc` (increment), and `$push` (add to array).
4. **Delete:** `deleteOne()`, `deleteMany()` – Used to remove documents from a collection.

MongoDB MongoDB is a widely used, open-source, non-relational database management system developed by MongoDB Inc. It is designed for modern application developers and the cloud era. As a document database, MongoDB makes it easy for developers to store structured or unstructured data.

MongoDB is a core component of the **MERN Stack** (MongoDB, Express, React, Node.js) used in this project. It connects seamlessly with Node.js and React because data is passed as JSON objects throughout the entire application lifecycle.

Advantages of MongoDB:

- **Schema-less:** MongoDB is a schema-less database, which implies that one collection can hold different documents. This flexibility allows for rapid iteration during the development of features like Polls and Spaces in the Chetan College Forum.
- **Performance:** MongoDB stores data in internal memory (RAM) for faster access. It is optimized for high read/write throughput, making it ideal for real-time applications.
- **Scalability:** MongoDB supports horizontal scaling through **Sharding**, allowing the database to handle growing amounts of data by distributing it across multiple servers.
- **Deep Querying:** MongoDB supports dynamic queries on documents using a document-based query language that is nearly as powerful as SQL.

- **High Availability:** MongoDB's replication feature (Replica Sets) provides automatic failover and data redundancy, ensuring the application remains online even if a server fails.
- **JSON Compatibility:** Since the entire project is built on JavaScript (React frontend, Node backend), MongoDB's native JSON-like storage format eliminates the need for complex data translation layers, simplifying the codebase.

DEFINITION OF PROBLEM FOR CHETAN COLLEGE FORUM

The current landscape of student interaction and knowledge sharing faces several challenges related to accessibility, organization, and engagement. While students utilize various tools for communication, they often lack a centralized, structured platform dedicated to academic and co-curricular discussions. These problems impact the efficiency of knowledge transfer and the overall campus community experience. Below are the primary problems Chetan College Forum addresses:

1. Fragmented Communication Channels:

- **Issue:** Important academic discussions currently happen in ephemeral channels like WhatsApp groups or verbal conversations.
- **Impact:** Valuable information and solutions are lost over time, forcing students to ask the same questions repeatedly.

2. Lack of Structured Knowledge Organization:

- **Issue:** Without specific categories or "Spaces," distinct topics (e.g., Coding vs. Music vs. Sports) get mixed together.
- **Impact:** Students struggle to find relevant information quickly, leading to cluttered feeds and reduced engagement.

3. Content Quality and Validation:

- **Issue:** On general social media, it is difficult to distinguish between correct answers and misinformation.
- **Impact:** Students may rely on incorrect advice. The lack of a voting system (upvotes/downvotes) makes it hard to identify the best solutions.

4. User Engagement and Interactivity:

- **Issue:** Static notice boards or read-only websites fail to engage the modern student.
- **Impact:** Low participation rates. The platform needs interactive features like **Polls** and **Real-time Commenting** to foster active community participation.

5. Database Scalability and Performance:

- **Issue:** As the number of topics and comments grows, retrieving data efficiently becomes challenging.
- **Impact:** Slow loading times can deter users. The system needs optimized queries (e.g., pagination) to handle high volumes of text-based content.

6. Mobile Accessibility:

- **Issue:** Students primarily access the internet via smartphones.
- **Impact:** A desktop-only interface would alienate the majority of users. The platform requires a fully responsive design to ensure seamless access on mobile devices.

7. Data Security and User Privacy:

- **Issue:** Handling student data (emails, passwords) requires strict security measures.
- **Impact:** Potential vulnerability to unauthorized access. The system must implement robust authentication (JWT) and password encryption (Bcrypt) to safeguard user identities.

SYSTEM ANALYSIS

System analysis for **Chetan College Forum** involves understanding the requirements of students and the academic community. The system must provide a seamless user experience while ensuring data security and efficient management of discussion topics. The analysis phase includes identifying key functionalities that will make the forum a vibrant hub for learning.

1. Problem Identification:

- Current methods of campus communication are disorganized and lack permanence.
- Students struggle to find archived answers to common queries.
- There is no unified platform that combines social interaction with structured academic categorization (Spaces).

2. User Requirements:

- **Students (Seekers):** Need a platform to ask questions, search for existing answers, filter topics by interest (Spaces), and receive notifications for replies.
- **Contributors (Experts):** Require features to post detailed answers, share rich content (code snippets, images), create polls, and build a reputation via upvotes and follower counts.

3. System Requirements:

- **Functional:**
    - User Authentication (Login/Register/Email Verify).
    - Topic Management (Create, Read, Update, Delete) with Rich Text.
    - Poll Creation and Voting mechanism.
    - Space and Tag management for categorization.

- Interactive features: Upvoting, Downvoting, and Commenting.

- **Non-Functional:**
  - **Scalability:** The ability to handle growing numbers of users and topics via MongoDB's document model.
  - **Performance:** Fast page loads using React's Virtual DOM and Single Page Application (SPA) architecture.
  - **Security:** Protection against common web vulnerabilities (XSS, NoSQL Injection) and secure session management.

4. System Architecture:

- The system is based on the **MERN architecture**, separating the frontend (Client) from the backend (Server).
- **Frontend (View):** Developed using **React.js** and **Redux Toolkit**, providing a dynamic, responsive user interface that communicates via APIs.
- **Backend (Controller/Model):** Developed using **Node.js** and **Express.js**, managing business logic and API routes.
- **Database:** Uses **MongoDB**, a NoSQL database, to store unstructured data like discussion threads and user profiles efficiently.

5. Data Flow:

- The system manages user input (e.g., creating a poll) through React forms.
- Data is sent via **JSON** payloads over HTTP requests to the Node.js server.
- The server validates the token (JWT), processes the logic, and stores the data in MongoDB collections.

- The response is sent back to the client to update the UI instantly without a page reload (via Redux state updates).

6. User Interface and Experience:

- The system emphasizes specific "Spaces" (e.g., Technology, Music) accessible via a sidebar.
- **React-Bootstrap** is used to ensure the layout is consistent and responsive across all devices.

SYSTEM REQUIREMENTS

Minimum Hardware Requirements for Development

- **CPU:** Intel Core i5 or equivalent (Node.js and React dev servers require moderate processing power).
- **Memory (RAM):** 8 GB (4 GB is absolute minimum, but 8 GB is recommended for running MongoDB, Backend, and Frontend servers simultaneously).
- **Hard Disk:** 256 GB SSD (Solid State Drive is preferred for faster compilation and database reads).
- **System Type:** 64-bit Operating System.

Minimum Software Requirements for Development

- **Operating System:** Windows 10/11, macOS, or Linux (Ubuntu 20.04+).
- **Runtime Environment:** Node.js (v14.0.0 or later).
- **Database:** MongoDB (Local installation or MongoDB Atlas Cloud).
- **Package Manager:** NPM (Node Package Manager) or Yarn.
- **Code Editor:** Visual Studio Code (VS Code) with extensions for ES7+ React/Redux and Prettier.
- **API Testing:** Postman or Thunder Client.
- **Version Control:** Git and GitHub.
- **Browser:** Google Chrome (with React Developer Tools installed).

Minimum Hardware and Software Requirements for Running the Website (Client-Side)

On PCs (Desktops and Laptops)

- **CPU:** Intel Core i3 or equivalent.
- **Memory (RAM):** 4 GB or higher.
- **Internet Connection:** Broadband connection for loading dynamic content and real-time updates.
- **Web Browser:** Modern browsers supporting ES6 JavaScript features (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).

On Mobile Devices (Phones and Tablets)

- **CPU:** Quad-core processor (Snapdragon 600 series or equivalent) for smooth UI rendering.
- **Memory (RAM):** 3 GB or higher.
- **Operating System:** Android 10+ or iOS 14+.
- **Browser:** Chrome for Android, Safari, or Firefox Mobile.
- **Screen Resolution:** 720p HD or higher (The responsive design adjusts automatically to fit these screens).
- ---

SYSTEM DESIGN AND CODING

INTRODUCTION: Software design is a critical phase in the development of **Chetan College Forum**, laying the architectural foundation for the entire application. It transcends basic coding by establishing a structured plan for how the MERN stack components (MongoDB, Express, React, Node.js) interact. Design is initiated after the system requirements (such as the need for Spaces, Polls, and User Authentication) have been specified and analyzed. This step forms the core of the development phase and precedes the actual implementation of features. The primary goal of this phase is to create a scalable model of the forum that ensures seamless data flow between the client and server.

The central focus of the design for Chetan College Forum is **User Experience (UX)** and **Data Integrity**. Through the design process, the quality of the final platform is ensured by translating student requirements—like the need for categorized discussions and real-time voting—into a structured framework of React Components and API Endpoints. A robust design minimizes the risk of bugs, ensures secure data handling (e.g., password encryption), and creates a system that is easy to maintain and expand in the future.

In this phase, detailed representations of data structures (Mongoose Schemas), program structures (Redux State Slices), and procedural steps (Authentication Middleware) are defined and implemented. From both a technical and project management perspective, this system design plays a pivotal role in guiding the project toward its successful deployment.

INPUT DESIGN: Input design is the process of converting user-oriented input into a computer-readable format. In **Chetan College Forum**, input design focuses on the various forms and interactive elements users engage with to create content. As a key component of the overall system design, this

was executed with meticulous attention to detail to ensure that students can easily ask questions, vote on polls, and customize their profiles without technical friction.

The primary objectives of input design in this project were:

1. **Efficiency:** To create input processes (like One-Click Voting or Auto-complete Tags) that are fast and minimize user effort.
2. **Accuracy:** To ensure high data integrity by implementing client-side validation (React) and server-side validation (Node.js/Mongoose) to prevent empty posts or invalid data.
3. **User-Friendliness:** To ensure that complex actions, such as creating a poll with dynamic options, are intuitive and straightforward.

Input Data Considerations in Chetan College Forum: When designing the input interfaces, the goal was to create a process that is logical and error-free.

- **Data Recording:** Capturing raw data from users via **React Bootstrap Forms**. This includes text inputs for Topic Titles, rich text for Content, and file inputs for Avatar uploads.
- **Data Conversion:** Converting user actions into JSON format. For example, when a user selects "Technology" from a dropdown, it is converted into a string value sent to the backend API.
- **Data Verification:** Checking inputs before submission. For instance, the `NewTopic` component prevents submission if the title is empty or if a poll has fewer than two options.
- **Interactive Input:** The system features real-time interactive inputs, such as the "Search" bar which filters topics instantly, and the "Vote" buttons which capture user feedback immediately.

Specific Input Interfaces Designed:

- **Authentication Forms:** Secure inputs for Email and Password with validation checks.

- **Topic Creation Wizard:** A comprehensive form allowing users to input Title, Content, select a "Space" (Category), and add dynamic Tags.
- **Poll Creator:** A dynamic input module where users can add or remove poll options and set expiration durations.
- **Profile Editing:** Inputs for updating personal details, bio, and uploading profile pictures, which are then processed by Cloudinary.

OUTPUT DESIGN: Output design refers to the process of determining how the results of processing will be communicated to the users. In **Chetan College Forum**, output design plays a crucial role in displaying academic discussions, poll results, and community interactions in a way that is readable and engaging.

The outputs of the system are categorized as follows:

1. **Interactive Outputs:** The most common output type in this Single Page Application (SPA). This includes the **Topic Feed**, where data fetched from the database is dynamically rendered as a list of cards. When a user upvotes a topic, the output (vote count) updates instantly without a page reload.
2. **Visual Outputs:** The **Poll Results** are a key visual output. Instead of just showing numbers, the system calculates percentages and renders them as animated progress bars, making the data easy to interpret at a glance.
3. **Operational Outputs:** System feedback such as "Topic Created Successfully" or "Invalid Password" (Toast Notifications) helps users understand the status of their actions.
4. **Turnaround Outputs:** Data that allows for further action, such as the "Edit Profile" form which pre-fills with existing user data (Output) so the user can modify it (Input).

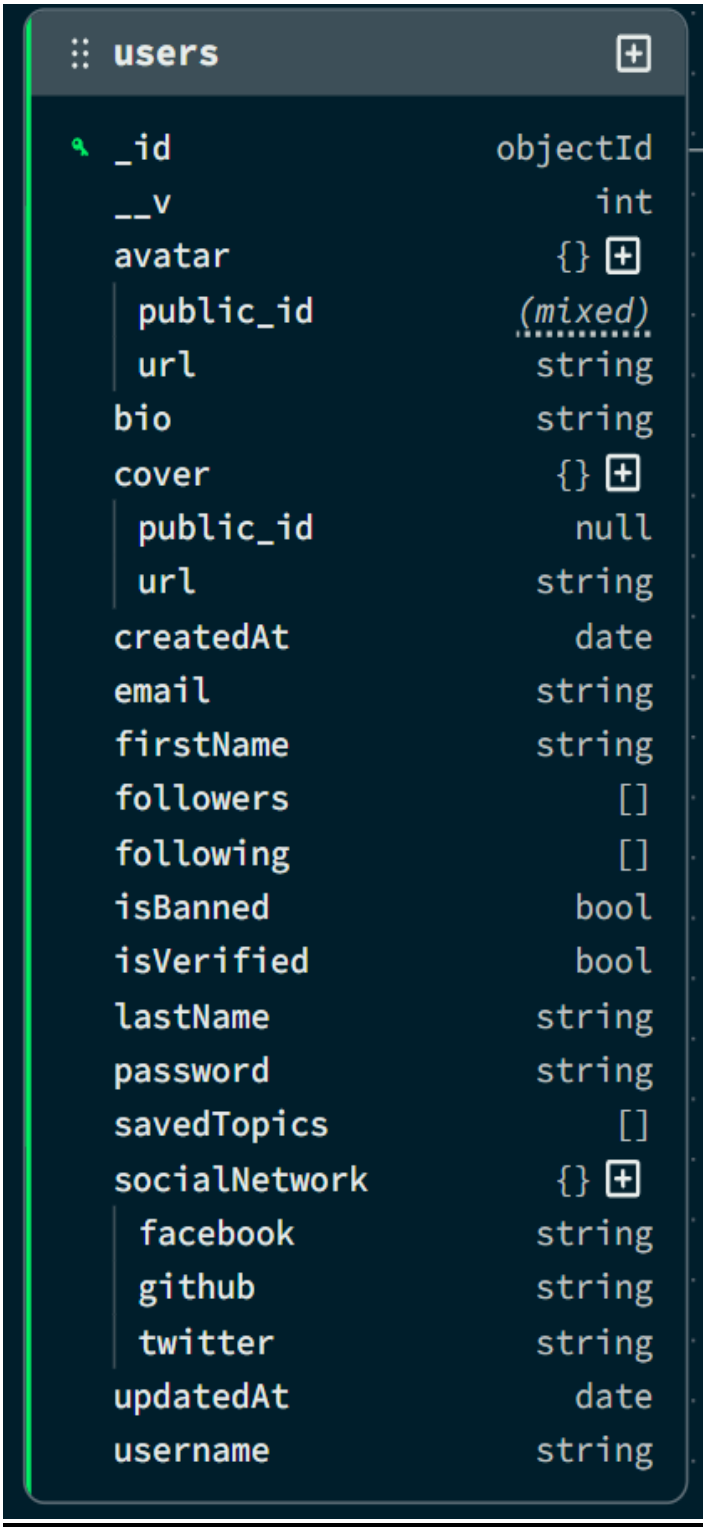
Key Output Design Principles Applied:

- **Responsiveness:** Using the Bootstrap Grid system, outputs are designed to adapt to any screen size. A topic list that displays as a grid on desktop transforms into a vertical stack on mobile devices.
- **Clarity:** Information is structured hierarchically. The Topic Title is prominent, followed by metadata (Author, Date, Space), and finally the content, ensuring users can quickly scan for relevant information.
- **Navigation:** The layout includes a dynamic Sidebar (Output) that lists active Spaces and Top Contributors, allowing users to navigate through the system effectively based on the data presented.

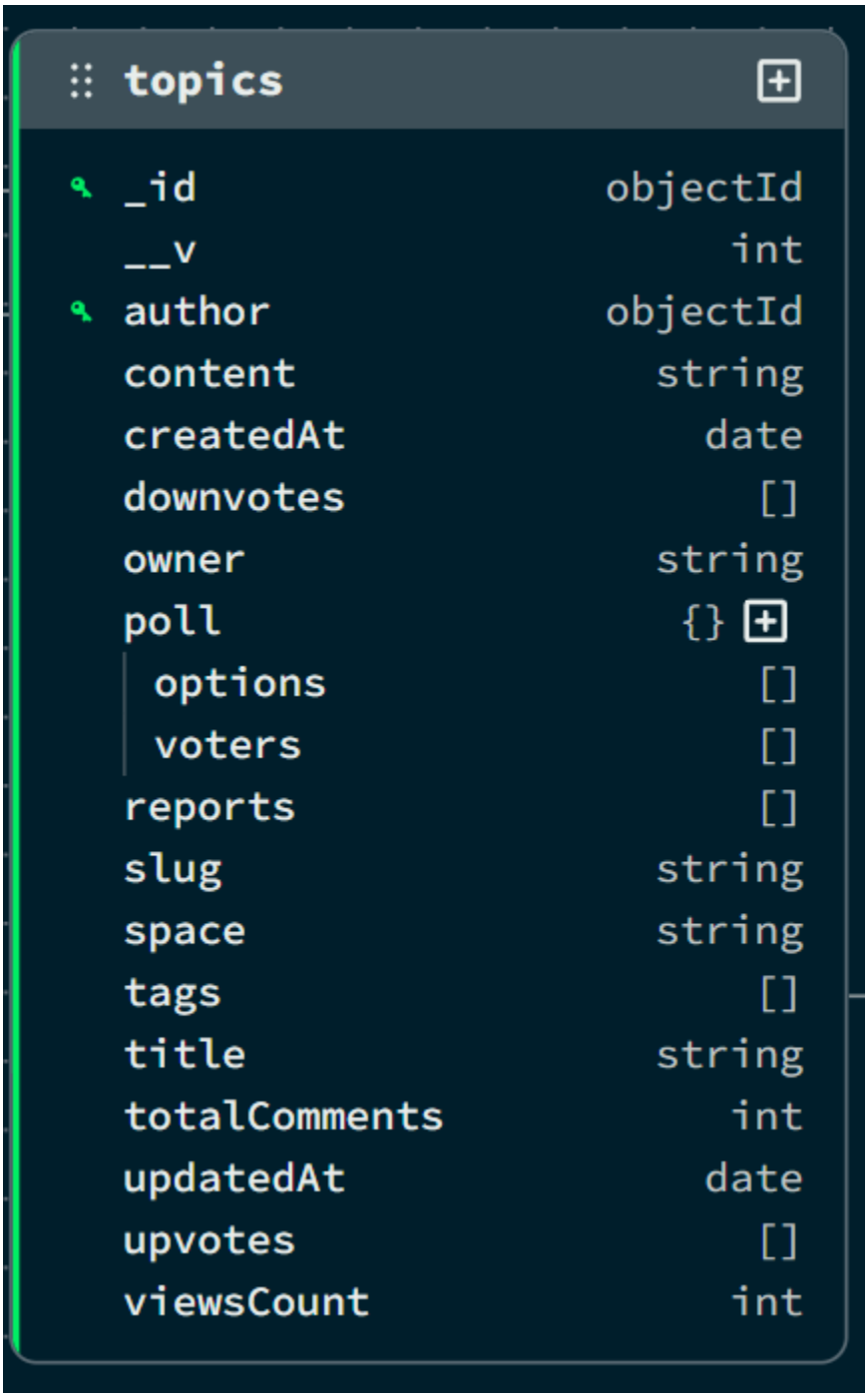
CONCLUSION: In the software engineering process of **Chetan College Forum**, system design was foundational for building a quality, reliable, and user-friendly academic platform. The rigorous **Input Design** ensures that student queries and poll data are captured accurately and validated before storage. Meanwhile, the thoughtful **Output Design** guarantees that knowledge is presented in a well-structured, visually appealing manner that encourages engagement. By focusing on these design principles using the MERN stack, the system successfully meets the needs of the modern student community, operating efficiently and remaining scalable for future enhancements.

DATABASE DESIGN

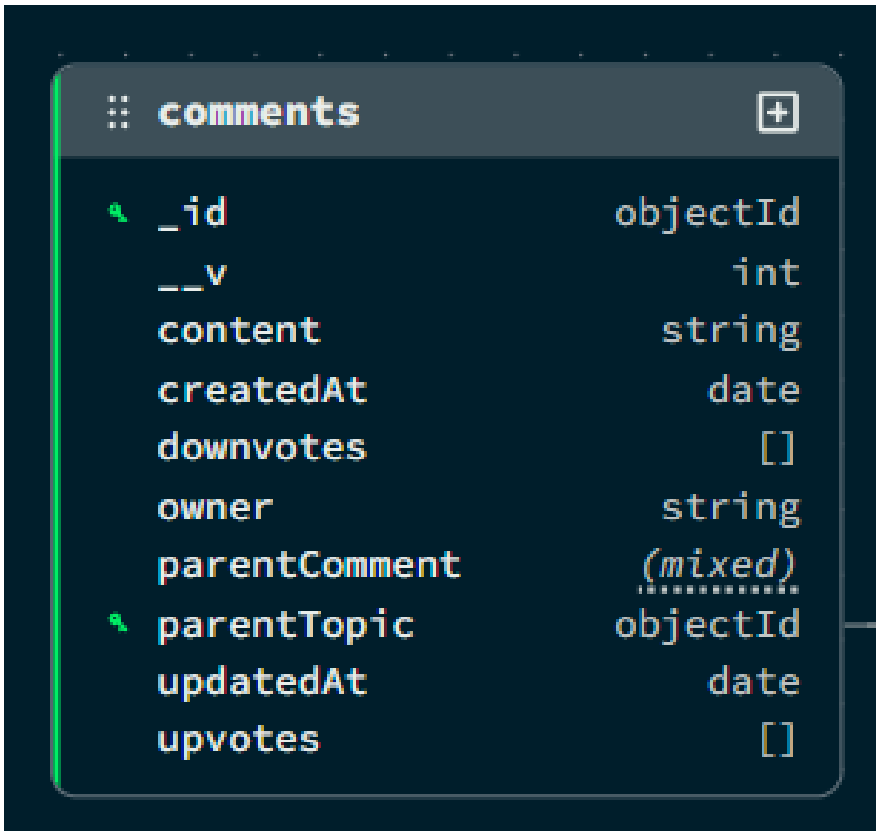
1. Users:



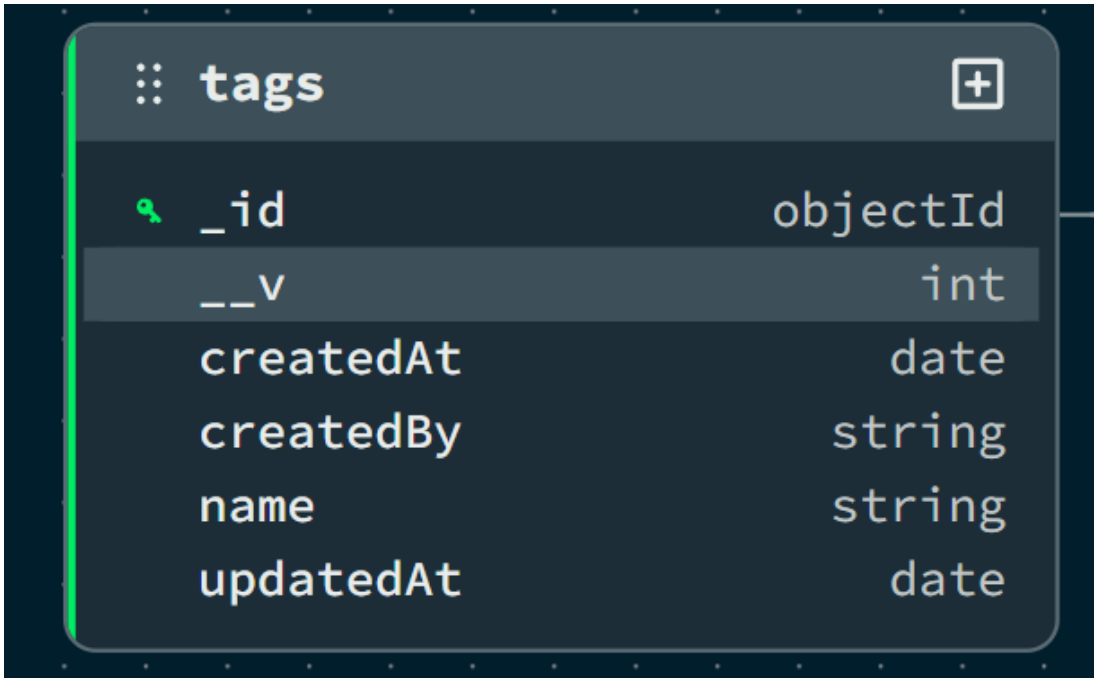
2. Topics


topics
_id objectId
__v int
author objectId
content string
createdAt date
downvotes []
owner string
poll {}
options []
voters []
reports []
slug string
space string
tags []
title string
totalComments int
updatedAt date
upvotes []
viewsCount int

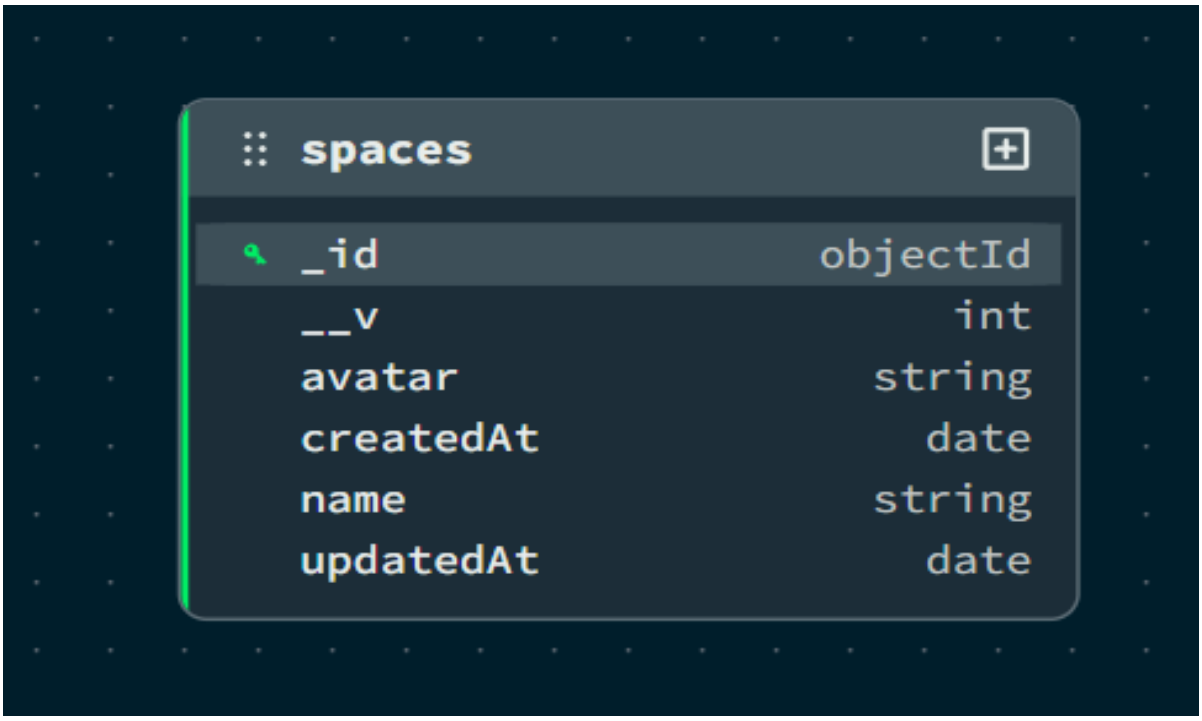
3. Comments



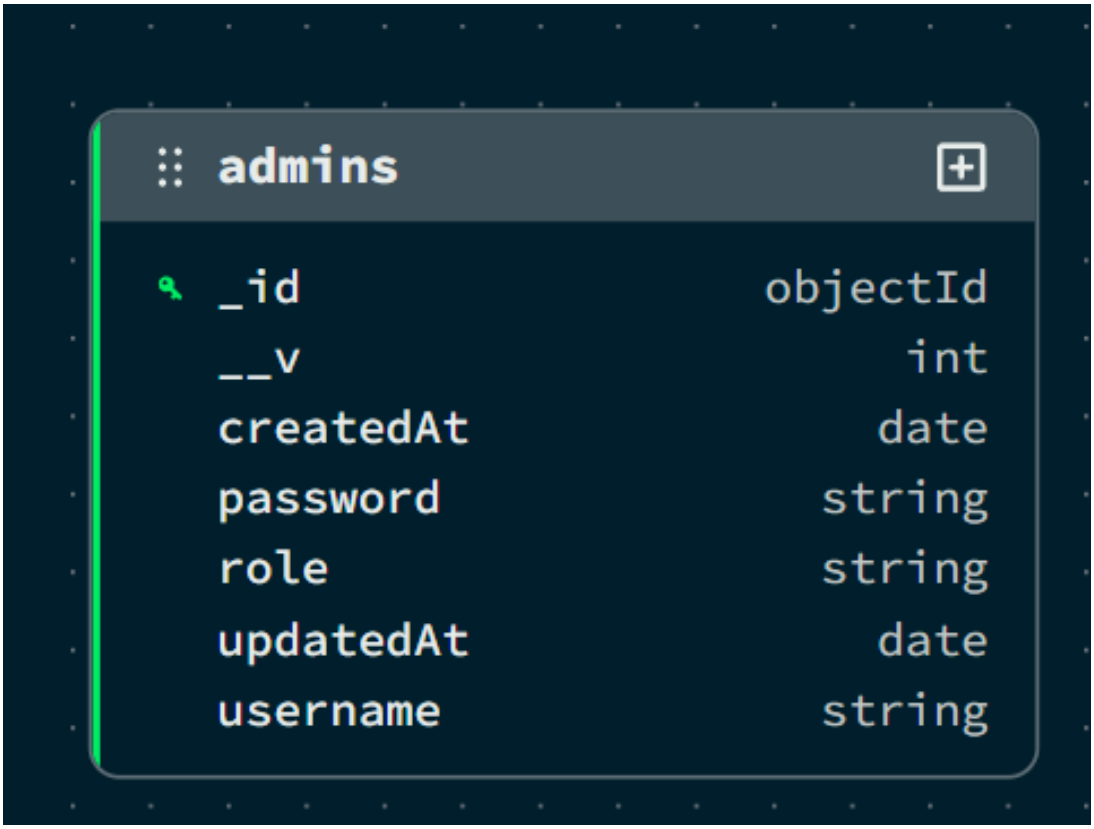
4. Tag



5. **Spaces**



Collection 6: Admins



DATA FLOW DIAGRAM (DFD)

A Data Flow Diagram (DFD) is a graphical tool used to describe and analyze the flow of data through a system. It focuses on the data flowing into the system, the processes that transform the data, and the data flowing out to storage or external entities.

DFDs are of two types:

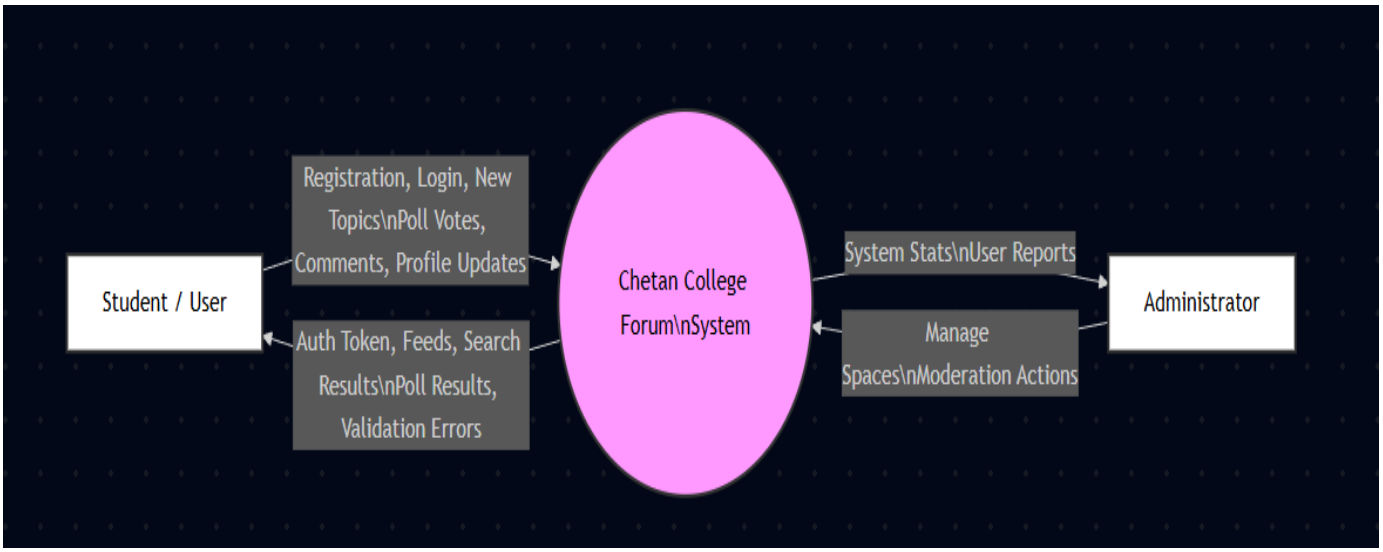
1. **Physical DFD:** The physical DFD is a model of the current system and is used to ensure that the current system has been clearly understood.
2. **Logical DFD:** The logical DFD is a model of the proposed system. It clearly shows the requirements on which the new system should be built, focusing on *what* happens, not *how* it happens.

Notation Used In DFD: There are four simple notations used to complete DFDs:

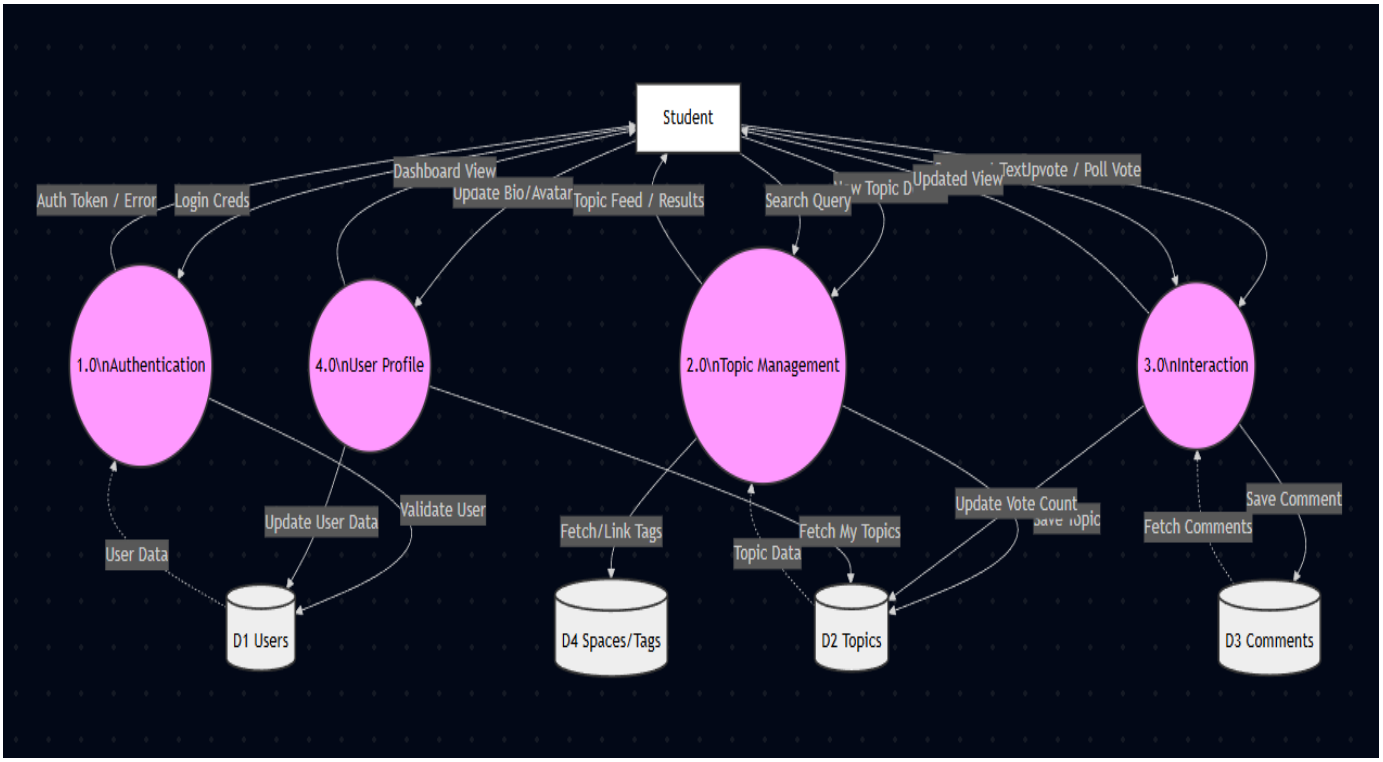
- **Data Flow (Arrow):** Represents the movement of data packets from one part of the system to another.
- **External Entity (Rectangle):** Represents sources (where data comes from) or destinations (where data goes) outside the system boundaries (e.g., Users, Admin).
- **Process (Circle/Rounded Rectangle):** Represents a function or logic that transforms incoming data into outgoing data (e.g., "Verify Login").
- **Data Store (Open Rectangle/Parallel Lines):** Represents repositories where data is stored for later use (e.g., Database Collections like Users, Topics).
-

DFD FOR THE SYSTEM

1. Level 0:



2. Level 1:



ENTITY RELATIONSHIP DIAGRAM

An Entity Relationship Diagram (ERD) expresses the overall logical structure of the database graphically. It illustrates the interrelationships between the entities in the Chetan College Forum system, providing a clear blueprint of how data is organized and connected within the MongoDB database.

The components of the E-R Diagram are:

1. Entity: An entity is a real-world object or concept that is distinguishable from all other objects. In this project, examples include User, Topic, and Space.
2. Relationship: It is an association among several entities. For instance, a "User" creates a "Topic," representing a relationship between the two.
3. Weak Entity: A weak entity is an entity that cannot be uniquely identified by its own attributes alone and depends on a strong entity. In this system, a Comment can be viewed conceptually as a weak entity because it is dependent on the existence of a specific Topic.
4. Attributes: A property or characteristic of an entity. For example, a User entity has attributes like Username, Email, and Password.

ENTITIES IN CHETAN COLLEGE FORUM

Based on the Mongoose models designed for the system, the key entities and their attributes are:

1. User: Represents the students and faculty registered on the forum.

    - Attributes: _id, firstName, lastName, username, email, password, bio, avatar, isVerified, followers, following.

2. Topic: Represents the main discussion threads created by users.

    - Attributes: _id, title, slug, content, space, tags, viewsCount, upvotes, downvotes, poll_data, createdAt.

3. Comment: Represents answers or replies posted within a topic.

    - Attributes: _id, content, parentTopic, parentComment, upvotes, downvotes, createdAt.

4. Space: Represents the categories or communities (e.g., Technology, Music) under which topics are organized.

    - Attributes: _id, name, description, avatar.

5. Tag: Represents keywords associated with topics for easier searching.

    - Attributes: _id, name, createdBy.

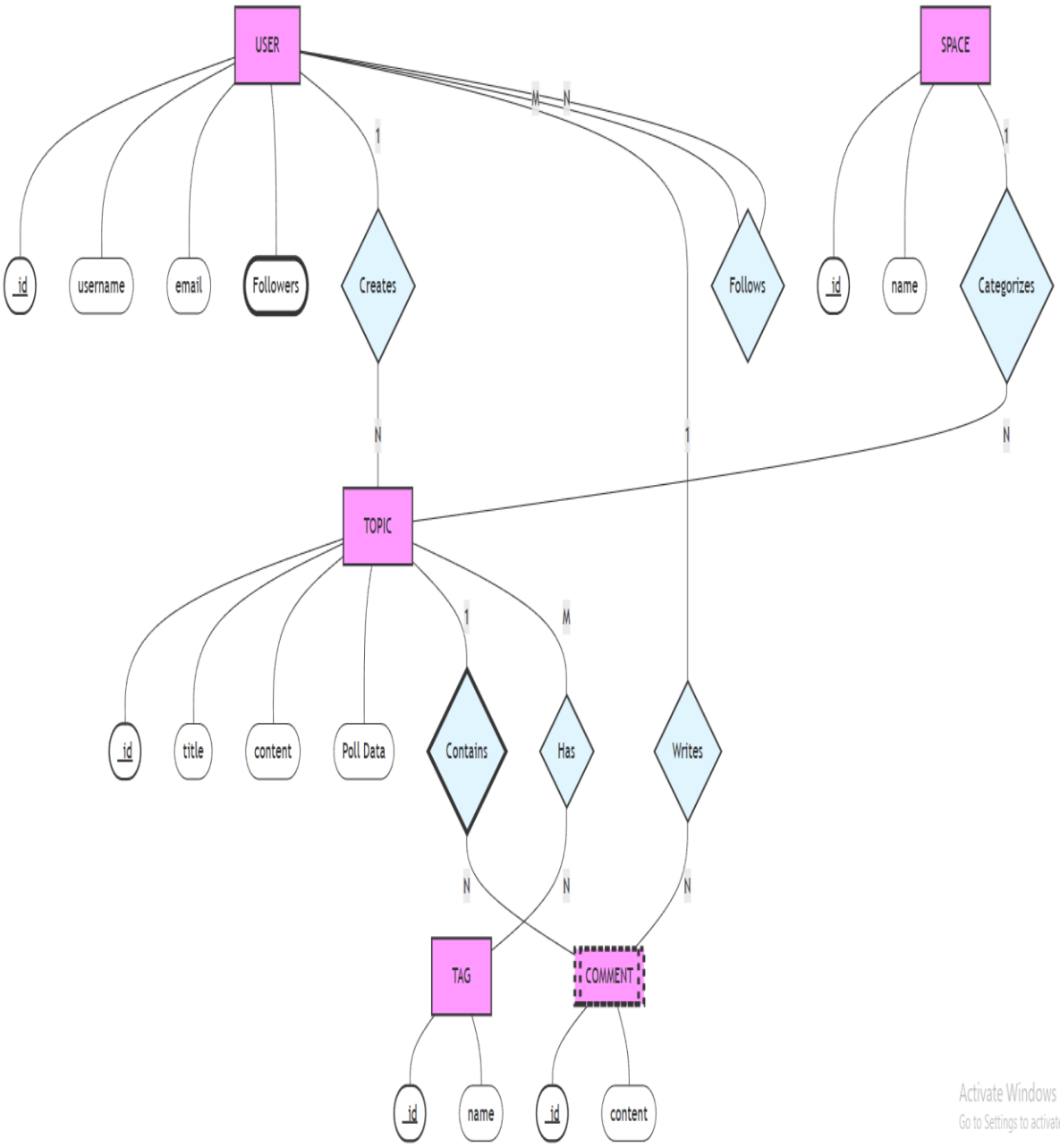
RELATIONSHIPS

The system enforces the following relationships to ensure data integrity and connectivity:

1. User — Topic (1:N): A single User can create multiple Topics (One-to-Many).
2. User — Comment (1:N): A single User can post multiple Comments across different threads.
3. Topic — Comment (1:N): A single Topic can have many Comments associated with it.
4. Space — Topic (1:N): A Space (Category) acts as a container for multiple Topics.

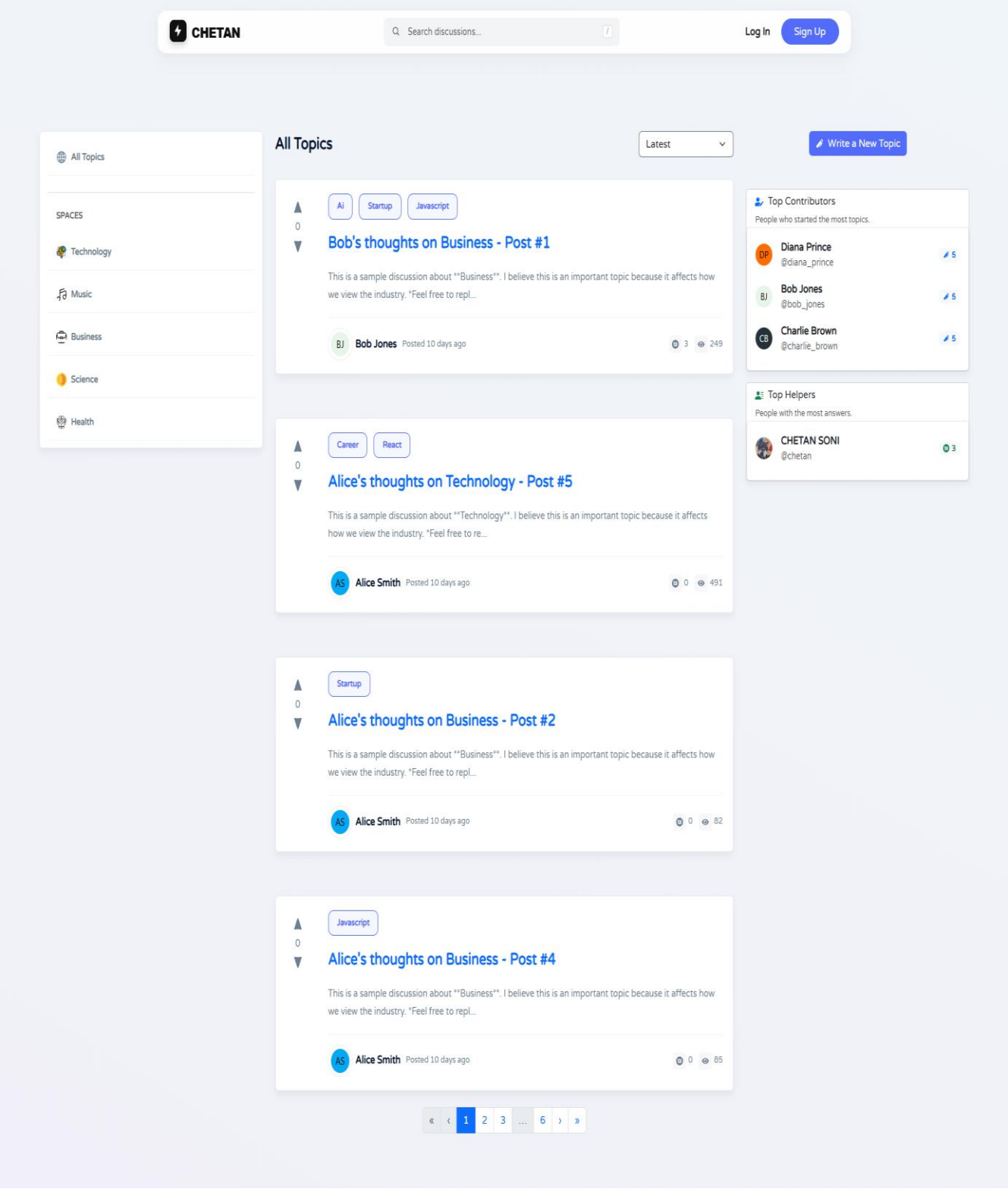
5. Topic — Tag (M:N): A Topic can have multiple Tags, and a Tag can be associated with multiple Topics (Many-to-Many).
6. User — User (M:N): A User can follow many other Users, and can be followed by many Users (Recursive Many-to-Many).

ENTITY RELATIONSHIP DIAGRAM



INPUT FORMS, PAGES AND CODING

1. Home Page:

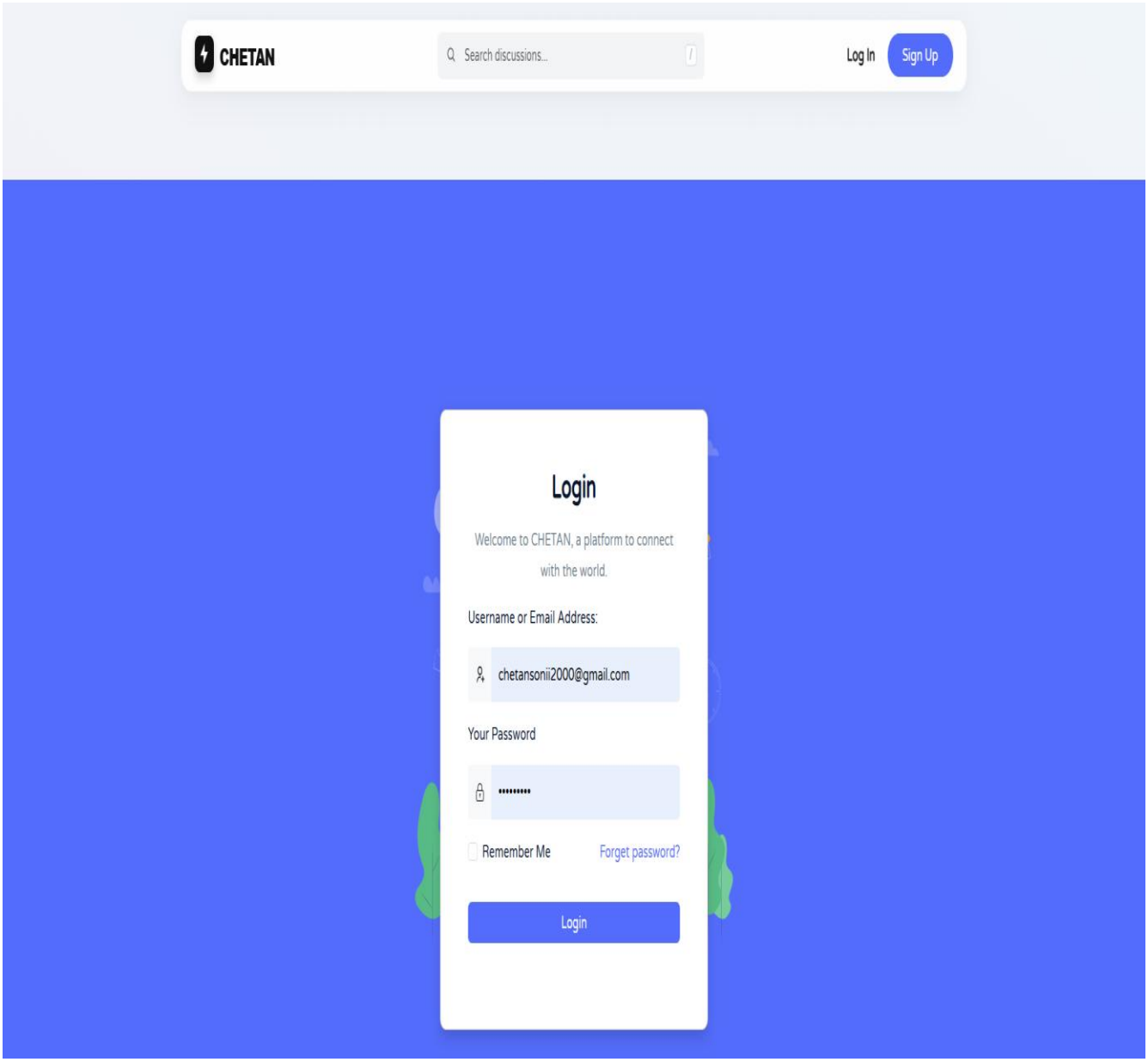


Code:

```
import { useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useParams } from "react-router-dom";
import { Container, Row, Col, Form } from "react-
bootstrap";
import { getAllTopics, setSortOption } from
"../redux/slices/topicSlice";
import TopicItem from
"../components/Topic/TopicItem";
import LeftSidebar from
"../components/LeftSidebar/LeftSidebar";
import RightSidebar from
"../components/RightSidebar/RightSidebar";
import Pagination from
"../components/Pagination/Pagination";
import SkeletonTopicItem from
"../components/Skeletons/SkeletonTopicItem";
const Home = () => {
  const dispatch = useDispatch();
  const { space } = useParams(); // Get active Space
from URL
  // Redux State: Get topics and loading status
  const { topics, getAllTopicsIsLoading, totalPages } =
useSelector(
    (state) => state.topic
  );
  const { sortOption, searchQuery } =
useSelector((state) => state.topic);
  const [page, setPage] = useState(1);
  // Effect: Reset page when filters change
  useEffect(() => {
    setPage(1);
  }, [sortOption, searchQuery, space]);
  // Effect: Fetch Topics based on filters & pagination
  useEffect(() => {
    dispatch(getAllTopics({ sortOption, searchQuery,
space, page }));
  }, [dispatch, sortOption, searchQuery, space, page]);
  // Handler: Change Page
  const handlePageChange = (newPage) => {
    if (newPage >= 1 && newPage <= totalPages) {
      setPage(newPage);
      window.scrollTo(0, 0);
    }
  };
  return (
    <main>
```

```
      <Container>
        <Row>
          {/* 1. Left Sidebar (Spaces & Navigation) */}
          <Col lg={3} className="d-none d-lg-block">
            <LeftSidebar />
          </Col>
          {/* 2. Main Feed Area */}
          <Col lg={6} className="main-content">
            {/* Filter Section */}
            <div className="filter d-flex justify-content-
between">
              <Form.Select
                onChange={(e) =>
dispatch(setSortOption(e.target.value))}
                value={sortOption}
              >
                <option value="latest">Latest</option>
                <option value="popular">Popular</option>
                <option value="most_upvoted">Most
Upvoted</option>
              </Form.Select>
            </div>
            {/* Topics List or Loading Skeletons */}
            {getAllTopicsIsLoading ? (
              [...Array(5)].map((_, i) =>
<SkeletonTopicItem key={i} />)
            ) : (
              topics?.map((topic) => (
                <TopicItem key={topic._id}
topic={topic} />
              ))
            )}
            {/* Pagination Component */}
            {!getAllTopicsIsLoading && topics?.length >
0 && (
              <Pagination
                currentPage={page}
                totalPages={totalPages}
                onPageChange={handlePageChange} />
            )} </Col>
          {/* 3. Right Sidebar (Top Contributors) */}
          <Col lg={3} className="d-none d-lg-block">
            <RightSidebar />
          </Col>        </Row>       </Container>
</main>);
};

export default Home;
```

2. Login Page:



Code:

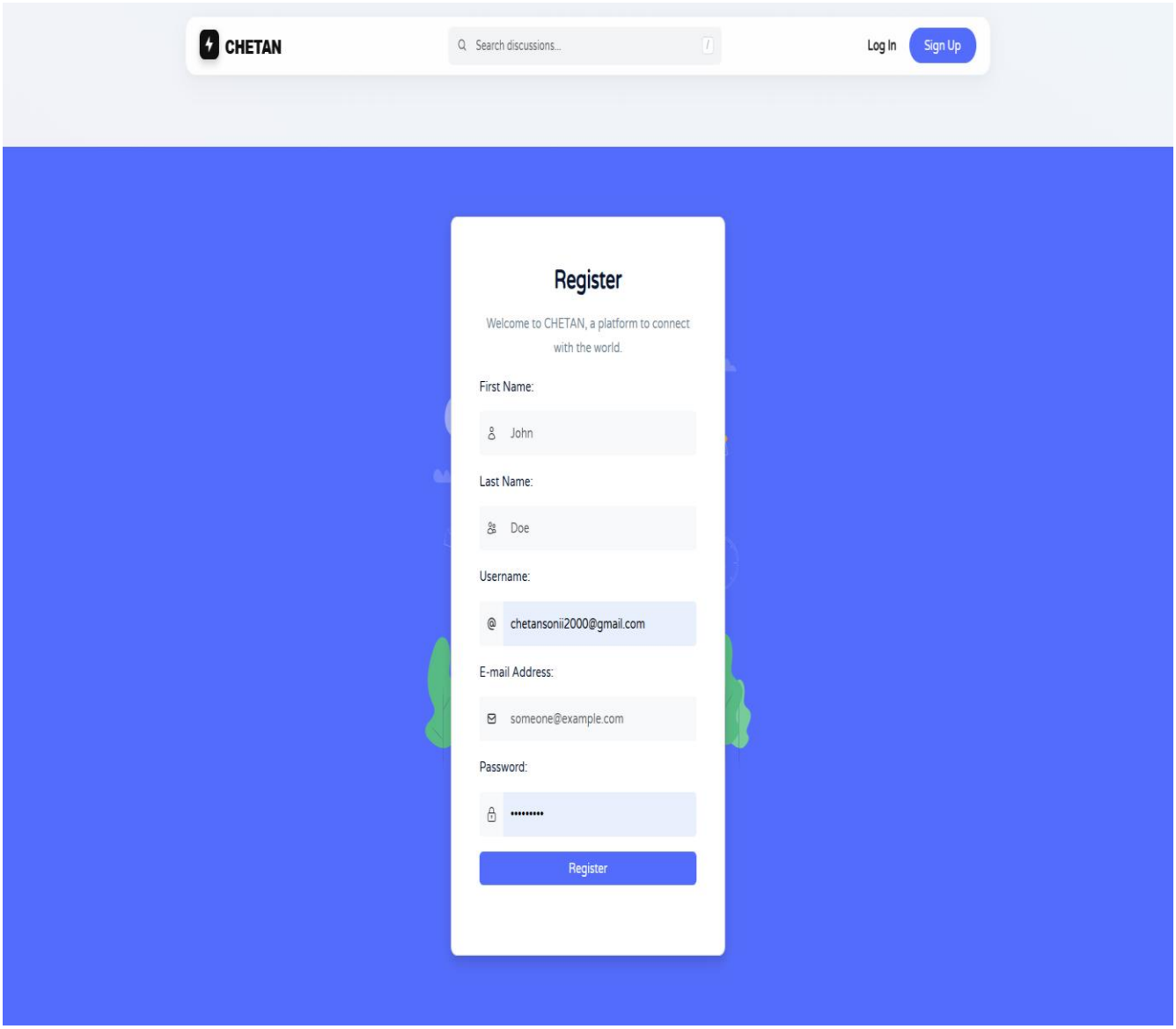
```
import { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useNavigate, Link } from "react-router-dom";
import { Form, Button, Row, Col, Card } from "react-
bootstrap";
import { login, resetLogin } from
"../redux/slices/authSlice";

const Login = () => {
  const [email, setEmail] = useState("");
```

```
  const [password, setPassword] = useState("");

  const navigate = useNavigate();
  const dispatch = useDispatch();

  // Redux State: Check auth status and login feedback
  const { isLoggedIn } = useSelector((state) =>
state.auth);
  const { isLoading, message, isError } =
useSelector((state) => state.auth.login);

  // Effect: Redirect to Home if already logged in
  useEffect(() => {
    if (isLoggedIn) navigate("/");
  }, [isLoggedIn, navigate]);

  // Effect: Reset login state (errors/messages) on page
load
  useEffect(() => {
    dispatch(resetLogin());
  }, [dispatch]);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (email && password) {
      dispatch(login({ email, password }));
    }
  };

  return (
    <Row className="justify-content-center align-
items-center" style={{ minHeight: "80vh" }}>
      <Col lg={5} md={8}>
        <Card className="shadow-sm">
          <Card.Body className="p-4">
            <h3 className="text-center mb-4">Login to
Chetan Forum</h3>

            {/* Feedback Message Block */}
            {message && (
              <div className={`alert ${isError ? "alert-
danger" : "alert-success"}`}>
                {message}
              </div>
            )}

            <Form onSubmit={handleSubmit}>
              {/* Email Input */}
```

```
            <Form.Group className="mb-3">
              <Form.Label>Email Address</Form.Label>
              <Form.Control
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                placeholder="Enter your email"
                required
              />
            </Form.Group>

            {/* Password Input */}
            <Form.Group className="mb-3">
              <Form.Label>Password</Form.Label>
              <Form.Control
                type="password"
                value={password}
                onChange={(e) =>
setPassword(e.target.value)}
                placeholder="Enter your password"
                required
              />
            </Form.Group>

            <div className="d-flex justify-content-
between mb-3">
              <Form.Check label="Remember Me" />
              <Link to="/forgot-password">Forgot
Password?</Link>
            </div>

            <Button type="submit" className="w-100"
variant="primary" disabled={isLoading}>
              {isLoading ? "Logging In..." : "Login"}
            </Button>
          </Form>

          <div className="text-center mt-3">
            Don't have an account? <Link
to="/register">Register</Link>
          </div>
        </Card.Body>
      </Card>
    </Col>
  </Row>
 );
};
```

```
export default Login;
```

3. Register Page:



Code:

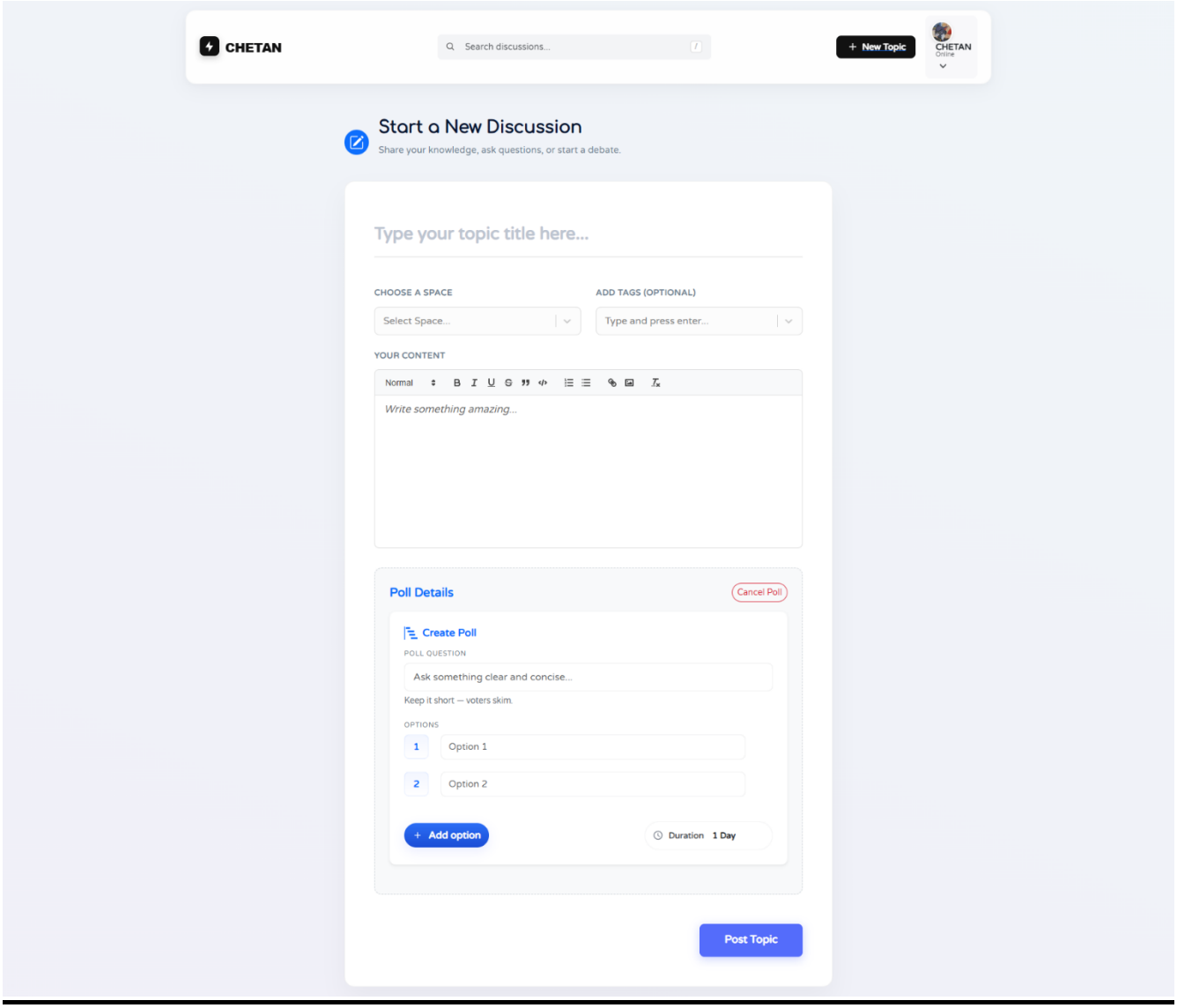
```
import { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useNavigate } from "react-router-dom";
import { Form, Button, Row, Col, Card, InputGroup }
from "react-bootstrap";
import { register, resetRegister } from
"../redux/slices/authSlice";
const Register = () => {
  // Local State for Form Inputs
  const [formData, setFormData] = useState({
```

```
    firstName: "", lastName: "", username: "", email: "",
password: ""
  });
  const navigate = useNavigate();
  const dispatch = useDispatch();
  // Redux State: Auth Status & Registration Feedback
  const { isLoggedIn } = useSelector((state) =>
state.auth);
  const { isLoading, message, isError } =
useSelector((state) => state.auth.register);
  // Effect: Redirect if already logged in
  useEffect(() => {
    if (isLoggedIn) navigate("/");
  }, [isLoggedIn, navigate]);

  // Effect: Clear previous registration messages
  useEffect(() => {
    dispatch(resetRegister());
  }, [dispatch]);
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]:
e.target.value });
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    if (Object.values(formData).every(field => field !==
"")) {
      dispatch(register(formData));
    }
  };
  return (
    <Row className="justify-content-center align-
items-center" style={{ minHeight: "80vh" }}>
      <Col lg={6} md={8}>
        <Card className="shadow-sm">
          <Card.Body className="p-4">
            <h3 className="text-center mb-4">Create
Account</h3>

            {/* Feedback Message */}
            {message && (
              <div className={`alert ${isError ? "alert-
danger" : "alert-success"}`}>
                {message}
              </div>
            )}

            <Form onSubmit={handleSubmit}>
```

```
            <Row>
              <Col md={6}>
                <Form.Group className="mb-3">
                  <Form.Label>First Name</Form.Label>
                  <Form.Control name="firstName"
onChange={handleChange} required />
                </Form.Group>
              </Col>
              <Col md={6}>
                <Form.Group className="mb-3">
                  <Form.Label>Last Name</Form.Label>
                  <Form.Control name="lastName"
onChange={handleChange} required />
                </Form.Group>
              </Col>
            </Row>

            <Form.Group className="mb-3">
              <Form.Label>Username</Form.Label>
              <Form.Control name="username"
onChange={handleChange} required />
            </Form.Group>

            <Form.Group className="mb-3">
              <Form.Label>Email Address</Form.Label>
              <Form.Control type="email" name="email"
onChange={handleChange} required />
            </Form.Group>

            <Form.Group className="mb-4">
              <Form.Label>Password</Form.Label>
              <Form.Control type="password"
name="password" onChange={handleChange}
required />
            </Form.Group>

            <Button type="submit" className="w-100"
disabled={isLoading}>
              {isLoading ? "Registering..." : "Register"}
            </Button>
          </Form>
        </Card.Body>
      </Card>
    </Col>
  </Row>
 );
};
```

```
export default Register;
```

4. New Topic



Code:

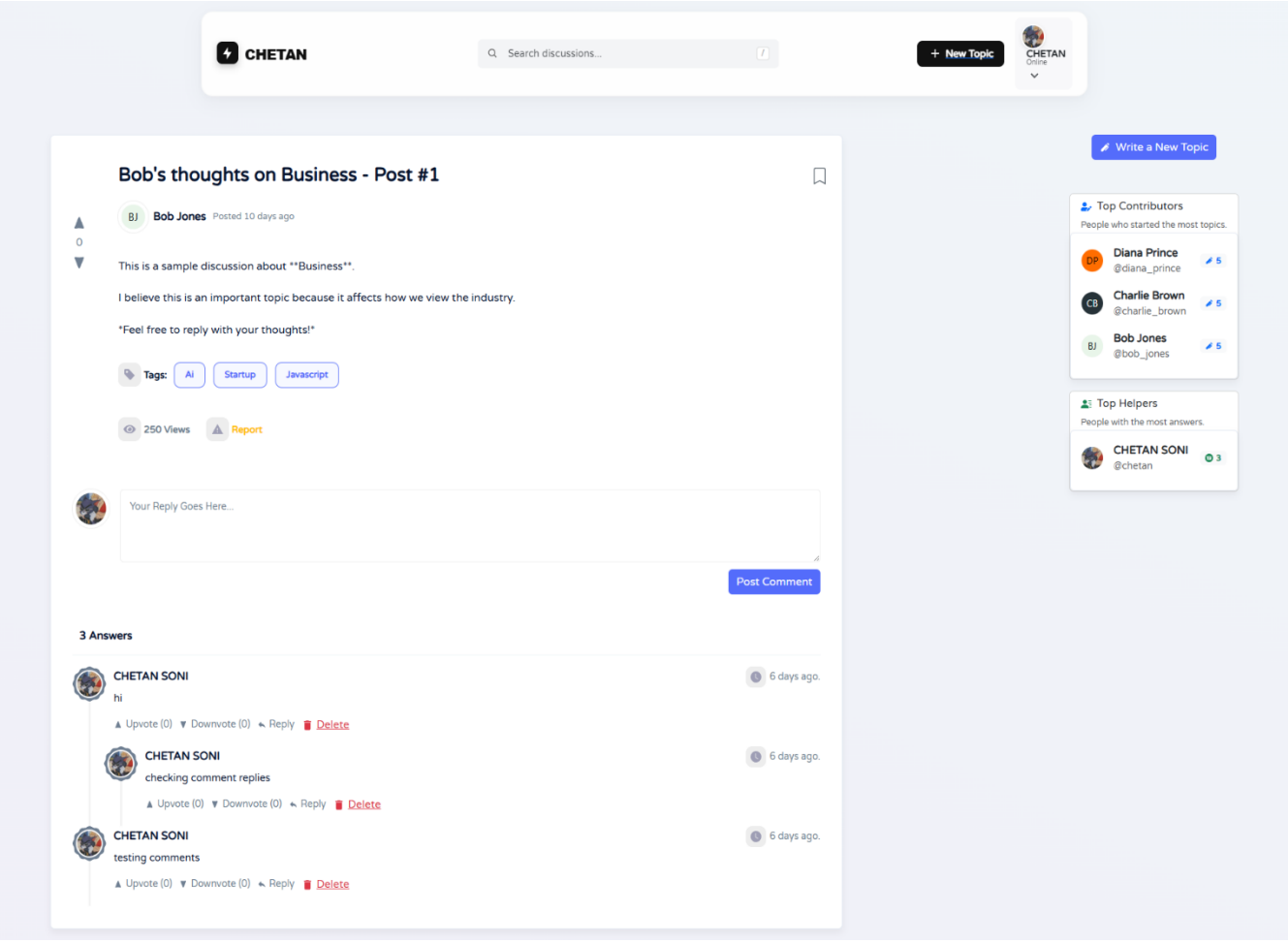
```
import { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Container, Row, Col, Form, Button } from
"react-bootstrap";
import Select from "react-select";
import CreatableSelect from "react-select/creatable";
import { addTopic, getSpaces, resetNewTopic } from
"../redux/slices/topicSlice";
```

```
import CreatePoll from
"../components/Topic/Poll/CreatePoll";

const NewTopic = () => {
  const dispatch = useDispatch();

  // Local State
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");
  const [selectedSpace, setSelectedSpace] =
useState(null);
  const [selectedTags, setSelectedTags] = useState([]);
  const [showPoll, setShowPoll] = useState(false);
  const [pollData, setPollData] = useState(null);

  // Redux State
  const { spaces } = useSelector((state) => state.topic);
  const { isLoading, message, isSuccess } =
useSelector((state) => state.topic.addTopic);

  // Fetch Spaces on Mount
  useEffect(() => {
    dispatch(getSpaces());
    dispatch(resetNewTopic());
  }, [dispatch]);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (title && content && selectedSpace) {
      dispatch(addTopic({
        title,
        content,
        selectedSpace: selectedSpace.value,
        selectedTags,
        poll: showPoll ? pollData : null // Attach Poll Data
if active
      }));
    }
  };

  return (
    <Container className="py-5">
      <Row className="justify-content-center">
        <Col lg={8}>
          <h3 className="mb-4">Create New Topic</h3>

          {message && <div className={`alert
${isSuccess ? "alert-success" : "alert-
```

```
danger"}`}>{message}</div>}
        <Form onSubmit={handleSubmit}>
          <Form.Group className="mb-3">
            <Form.Label>Topic Title</Form.Label>
            <Form.Control value={title} onChange={(e)
=> setTitle(e.target.value)} required />
          </Form.Group>
          <Form.Group className="mb-3">
            <Form.Label>Space</Form.Label>
            <Select
              options={spaces?.map(s => ({ value: s.name,
label: s.name }))}
              onChange={setSelectedSpace}
            />
          </Form.Group>
          <Form.Group className="mb-3">
            <Form.Label>Content</Form.Label>
            <Form.Control as="textarea" rows={5}
value={content} onChange={(e) =>
setContent(e.target.value)} required />
          </Form.Group>

          <Form.Group className="mb-3">
            <Form.Label>Tags</Form.Label>
            <CreatableSelect isMulti
onChange={setSelectedTags} />
          </Form.Group>

          {/* Poll Toggle Section */}
          <div className="mb-4">
            <Button variant="outline-primary" size="sm"
onClick={() => setShowPoll(!showPoll)}>
              {showPoll ? "Remove Poll" : "Add Poll"}
            </Button>
            {showPoll && <CreatePoll
onPollChange={setPollData} />}
          </div>
          <Button type="submit" variant="primary"
size="lg" disabled={isLoading}>
            {isLoading ? "Posting..." : "Create Topic"}
          </Button>
        </Form>
      </Col>
    </Row>
  </Container>
 );
};
```

```
export default NewTopic;
```

5. Topic Page



Code:

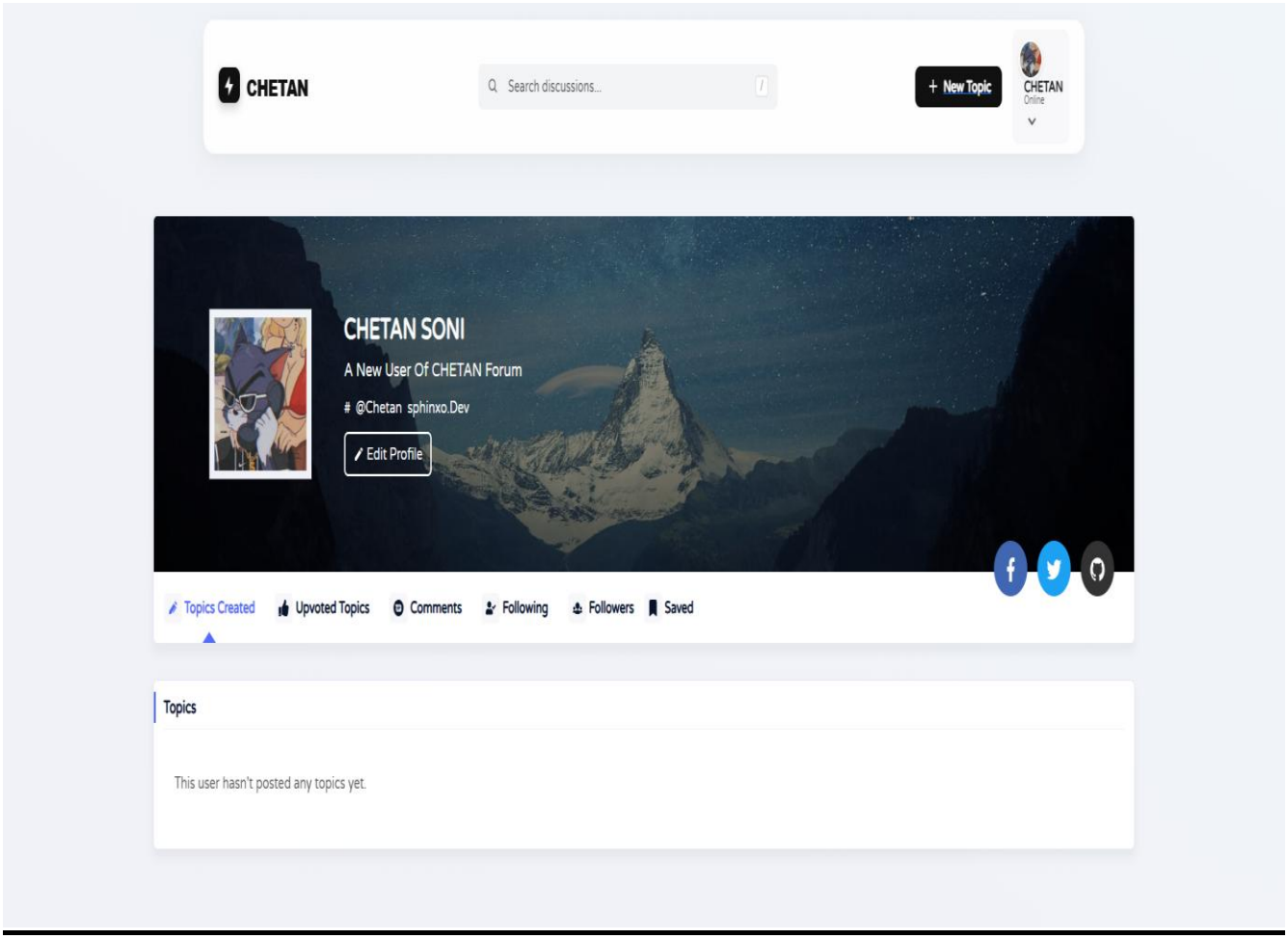
```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { Container, Col, Row, Image } from "react-
bootstrap";
import { getTopic, resetTopics } from
"../redux/slices/topicSlice";
import TopicContent from
"../components/Topic/TopicContent";
import CommentsContainer from
"../components/Comment/CommentsContainer";
import CommentForm from
```

```
"../components/Comment/CommentForm";
import RightSidebar from
"../components/RightSidebar/RightSidebar";
import SkeletonTopicPage from
"../components/Skeletons/SkeletonTopicPage";

const Topic = () => {
  const { id, slug } = useParams();
  const dispatch = useDispatch();

  // Redux State
  const { topic, getTopicIsLoading, message } =
useSelector((state) => state.topic);
  const { user } = useSelector((state) => state.auth);

  // Fetch Topic Data on Mount or ID change
  useEffect(() => {
    dispatch(resetTopics());
    dispatch(getTopic({ id, slug }));
  }, [dispatch, id, slug]);

  if (getTopicIsLoading) return <SkeletonTopicPage />;

  return (
    <Container className="py-4">
      <Row>
        {/* Main Content Column */}
        <Col lg={8}>
          {message && <div className="alert alert-
info">{message}</div>}

          {topic && (
            <article className="bg-white p-4 rounded
shadow-sm mb-4">
              {/* 1. Topic Body (Title, Content, Polls) */}
              <TopicContent topic={topic} />

              {/* 2. Add Comment Section */}
              <div className="d-flex mt-5 gap-3">
                <Image
                  src={user?.avatar?.url || "/default-
avatar.png"}
                  roundedCircle
                  width={40} height={40}
                />
                <div className="flex-grow-1">
                  <CommentForm topic={topic} />
                </div>
```

```
            </div>

            {/* 3. Comments List */}
            <CommentsContainer topic={topic} />
          </article>
        )}
      </Col>

      {/* Right Sidebar Column */}
      <Col lg={4} className="d-none d-lg-block">
        <RightSidebar />
      </Col>
    </Row>
  </Container>
 );
};

export default Topic;
```

---

6. Profile Page



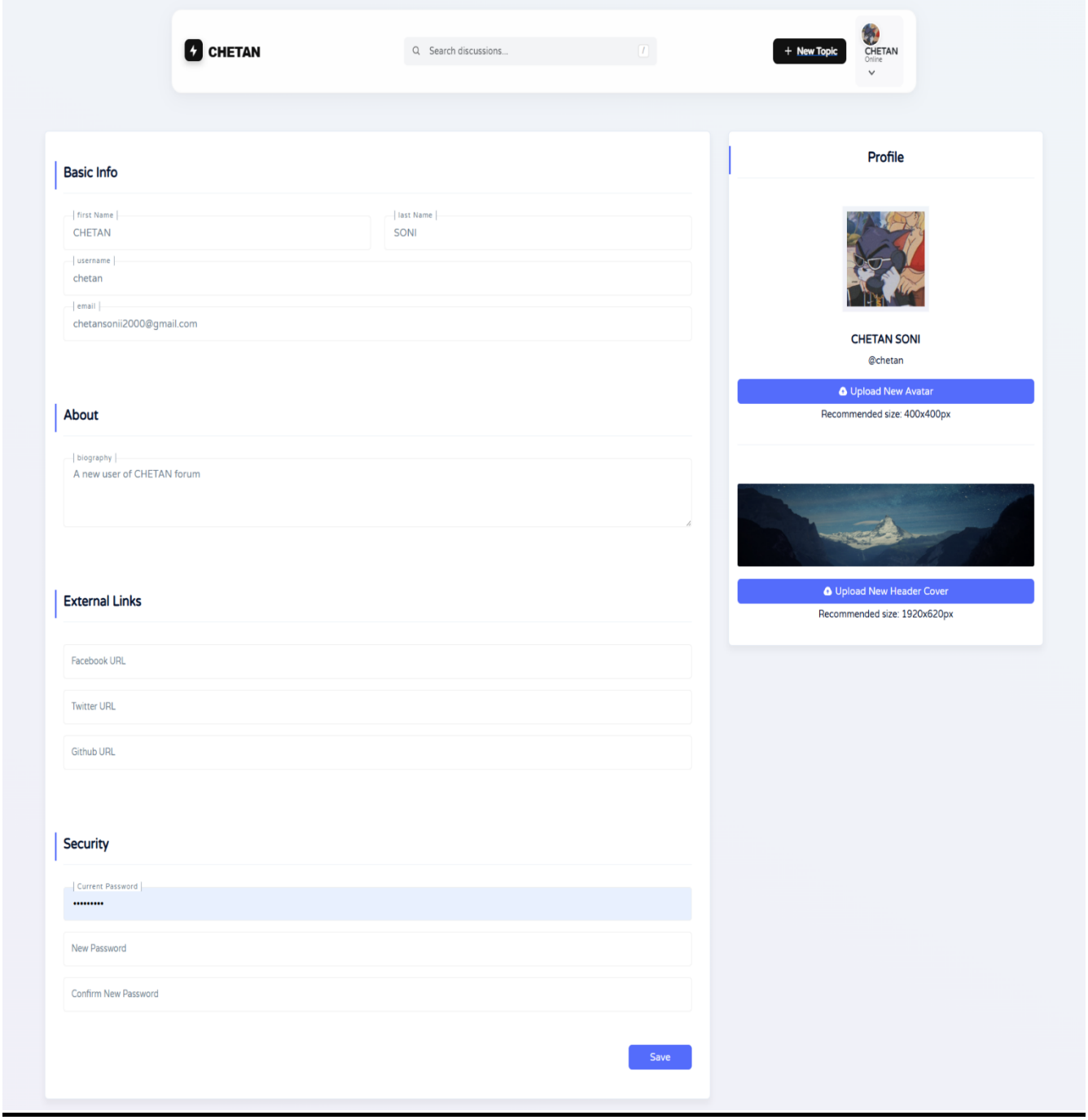
Code:

```
import { Container, Row, Col } from "react-bootstrap";
import { Outlet } from "react-router-dom";
import ProfileHeader from
"../components/Profile/ProfileHeader";

const Profile = () => {
  return (
    <main>
      <Container>
        <Row>
          <Col lg={10} className="mx-auto">
            {/* 1. Profile Header (Avatar, Bio, Stats) */}
            <ProfileHeader />
```

```
            {/* 2. Nested Routes Render Here (Topics,
Comments, Upvoted Tabs) */}
            <Outlet />
          </Col>
        </Row>
      </Container>
    </main>
  );
};

export default Profile;
```

---

7. Edit Profile Page



Code:

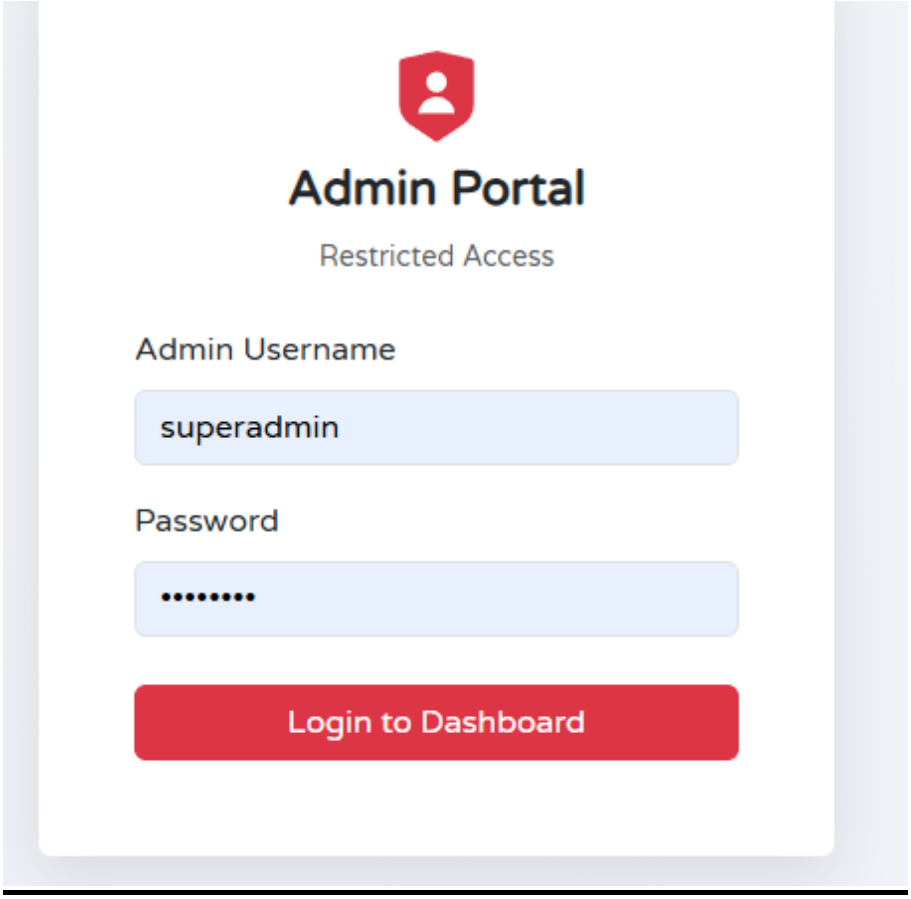
```
import { useState, useEffect, useRef } from "react";
```

```
import { useParams, Navigate } from "react-router-
dom";
import { useDispatch, useSelector } from "react-redux";
import { Form, Button, Row, Col, Container, Image }
from "react-bootstrap";
import { getUserProfile } from
"../redux/slices/profileSlice";
import { updateUserProfile, resetUpdateProfile } from
"../redux/slices/authSlice";
import SkeletonEditProfile from
"../components/Skeletons/SkeletonEditProfile";

const EditProfile = () => {
  const { username } = useParams();
  const dispatch = useDispatch();

  // Redux State
  const { user } = useSelector((state) => state.auth);
  const { isLoading, message, isSuccess } =
useSelector((state) =>
state.auth.updateUserProfileState);
  const { profileIsLoading } = useSelector((state) =>
state.profile);

  // Form State
  const [formData, setFormData] = useState({
    firstName: "", lastName: "", email: "", bio: "",
    password: "", newPassword: ""
  });
  const [avatar, setAvatar] = useState(null);
  const avatarRef = useRef();

  // Fetch Data on Load
  useEffect(() => {
    dispatch(getUserProfile(username));
    return () => dispatch(resetUpdateProfile());
  }, [dispatch, username]);

  // Populate Form
  useEffect(() => {
    if (user) {
      setFormData({
        firstName: user.firstName || "",
        lastName: user.lastName || "",
        email: user.email || "",
        bio: user.bio || "",
        // ... other fields
      });
```

```
    }
  }, [user]);

  const handleSubmit = (e) => {
    e.preventDefault();
    // Create FormData for File Uploads
    const submitData = new FormData();
    Object.keys(formData).forEach(key =>
submitData.append(key, formData[key]));
    if (avatar) submitData.append("avatar", avatar);

    dispatch(updateUserProfile(submitData));
  };

  // Redirect if trying to edit another user's profile
  const loggedUser =
JSON.parse(localStorage.getItem("user"))?.username;
  if (loggedUser && loggedUser !== username) return
<Navigate to={`/user/${loggedUser}/edit`} />;

  if (profileIsLoading) return <SkeletonEditProfile />;

  return (
    <Container className="py-5">
      <Form onSubmit={handleSubmit}
encType="multipart/form-data">
        <Row>
          {/* Main Edit Form */}
          <Col lg={8}>
            {message && <div className={`alert
${isSuccess ? "alert-success" : "alert-
danger"}`}>{message}</div>}

            <section className="mb-4">
              <h5>Basic Info</h5>
              <Row>
                <Col><Form.Control placeholder="First
Name" value={formData.firstName} onChange={(e)
=> setFormData({...formData, firstName:
e.target.value})} /></Col>
                <Col><Form.Control placeholder="Last
Name" value={formData.lastName} onChange={(e) =>
setFormData({...formData, lastName: e.target.value})}
/></Col>
              </Row>
            </section>

            <section className="mb-4">
```

```
            <h5>About</h5>
            <Form.Control as="textarea"
placeholder="Bio" value={formData.bio}
onChange={(e) => setFormData({...formData, bio:
e.target.value})} />
          </section>

          <div className="d-flex justify-content-end">
            <Button type="submit"
disabled={isLoading}>{isLoading ? "Saving..." : "Save
Changes"}</Button>
          </div>
        </Col>

        {/* Sidebar: Avatar Upload */}
        <Col lg={4}>
          <div className="text-center p-3 border
rounded">
            <Image src={user?.avatar?.url} roundedCircle
width={150} height={150} className="mb-3" />
            <input type="file" ref={avatarRef} hidden
onChange={(e) => setAvatar(e.target.files[0])} />
            <Button variant="outline-primary" onClick={()
=> avatarRef.current.click()}>Upload New
Avatar</Button>
          </div>
        </Col>
      </Row>
    </Form>
  </Container>
 );
};

export default EditProfile;
```

8. Admin Login:



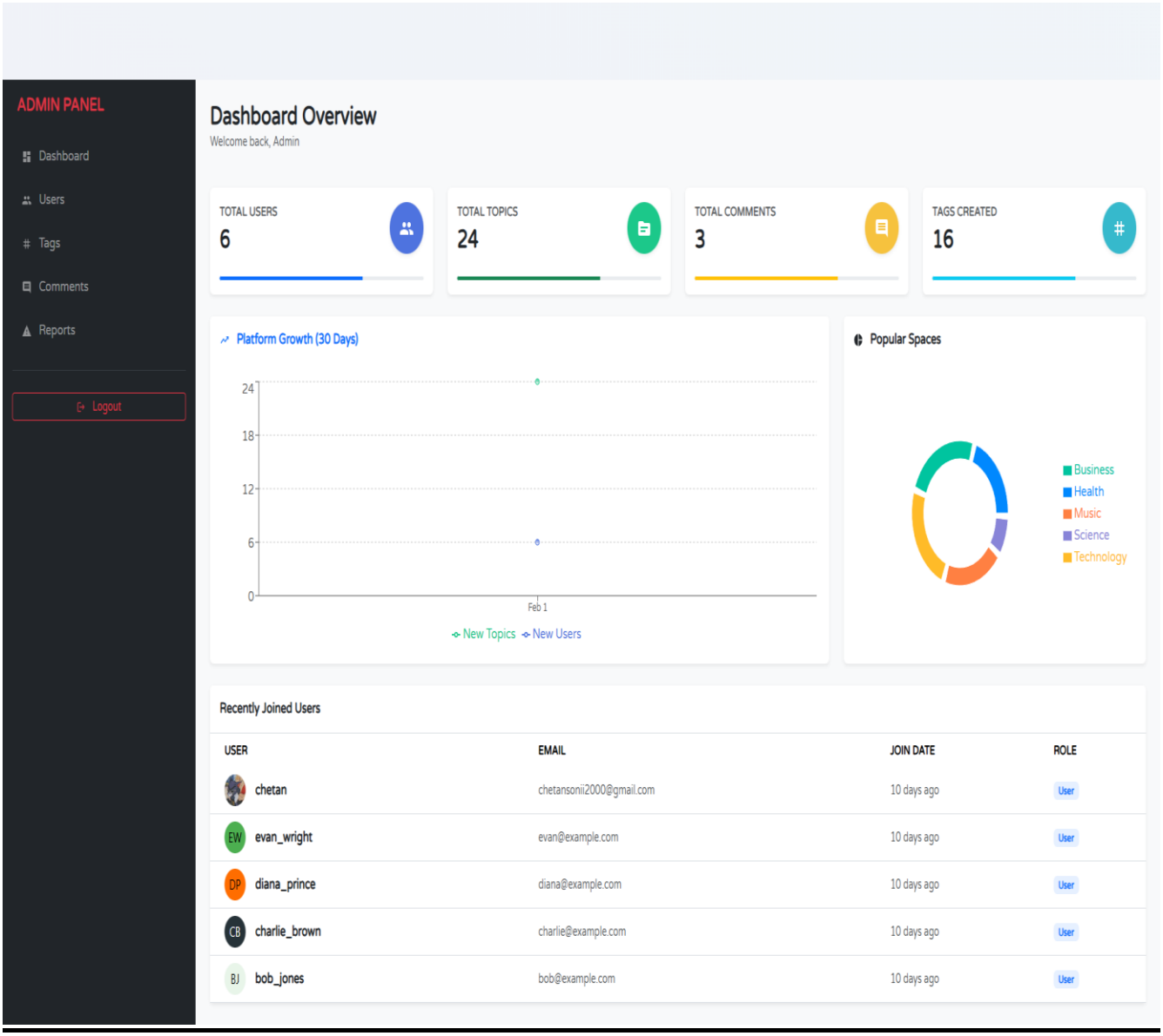
Code:

```
import { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useNavigate } from "react-router-dom";
import { Form, Button, Row, Col, Card, Container }
from "react-bootstrap";
import { adminLogin, resetAuth } from
"../../redux/slices/authSlice";

const AdminLogin = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const dispatch = useDispatch();
  const navigate = useNavigate();
```

```
  const { user, isError, message } = useSelector((state)
=> state.auth);

  useEffect(() => {
    if (user?.role === "admin")
navigate("/admin/dashboard");
    dispatch(resetAuth());
  }, [user, navigate, dispatch]);

  const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(adminLogin({ email, password }));
  };

  return (
    <Container className="d-flex justify-content-center
align-items-center vh-100">
      <Col md={5}>
        <Card className="shadow-lg border-0">
          <Card.Header className="bg-dark text-white
text-center py-3">
            <h4>Admin Portal</h4>
          </Card.Header>
          <Card.Body className="p-4">
            {isError && <div className="alert alert-
danger">{message}</div>}

            <Form onSubmit={handleSubmit}>
              <Form.Group className="mb-3">
                <Form.Label>Admin Email</Form.Label>
                <Form.Control
                  type="email"
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  required
                />
              </Form.Group>

              <Form.Group className="mb-4">
                <Form.Label>Password</Form.Label>
                <Form.Control
                  type="password"
                  value={password}
                  onChange={(e) =>
setPassword(e.target.value)}
                  required
                />
              </Form.Group>
```

```
            <Button type="submit" variant="dark"
classNameName="w-100">
              Access Dashboard
            </Button>
          </Form>
        </Card.Body>
      </Card>
    </Col>
  </Container>
 );
};

export default AdminLogin;
```

9. Admin Dashboard



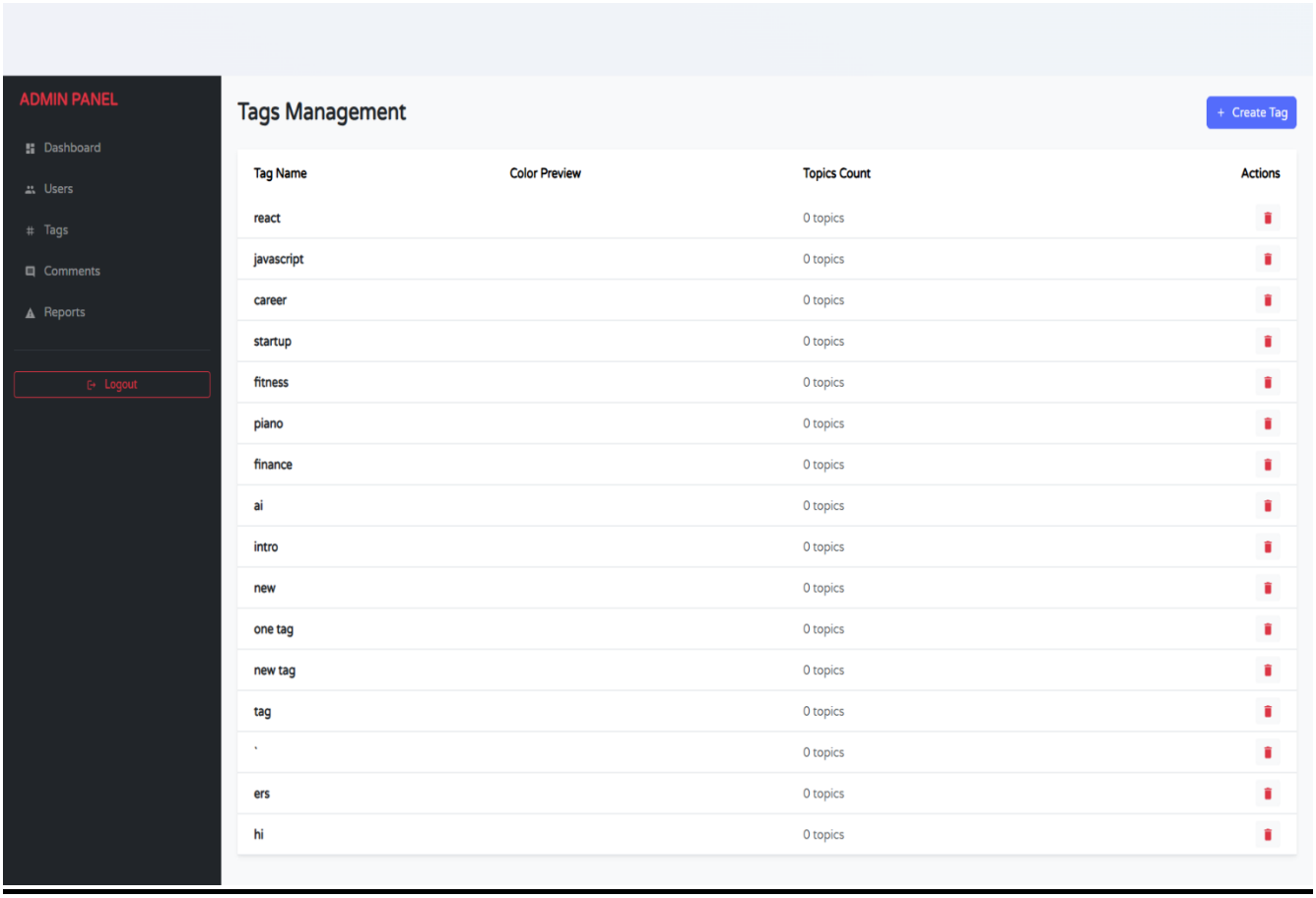
Code:

```
import { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Container, Row, Col, Card } from "react-
bootstrap";
import { getSystemStats } from
"../../redux/slices/adminSlice";
```

```
import { FaUsers, FaComments, FaLayerGroup } from
"react-icons/fa";
import LeftSidebar from
"../../components/LeftSidebar/LeftSidebar"; // Reusing
Sidebar

const StatCard = ({ title, count, icon, color }) => (
  <Card className={`border-0 shadow-sm text-white
bg-${color} mb-4`}>
    <Card.Body className="d-flex justify-content-
between align-items-center">
      <div>
        <h6 className="opacity-75">{title}</h6>
        <h2 className="fw-bold">{count}</h2>
      </div>
      <div style={{ fontSize: "2.5rem", opacity: 0.3
}}>{icon}</div>
    </Card.Body>
  </Card>
);

const AdminDashboard = () => {
  const dispatch = useDispatch();
  const { stats, isLoading } = useSelector((state) =>
state.admin);

  useEffect(() => {
    dispatch(getSystemStats());
  }, [dispatch]);

  return (
    <Container fluid className="py-4">
      <Row>
        <Col md={2}>
          <LeftSidebar adminMode={true} />
        </Col>

        <Col md={10}>
          <h3 className="mb-4">System Overview</h3>
          <Row>
            <Col md={4}>
              <StatCard
                title="Total Users"
                count={isLoading ? "..." : stats?.userCount}
                icon={<FaUsers />}
                color="primary"
              />
            </Col>
```

```
            <Col md={4}>
              <StatCard
                title="Total Topics"
                count={isLoading ? "..." : stats?.topicCount}
                icon={<FaLayerGroup />}
                color="success"
              />
            </Col>
            <Col md={4}>
              <StatCard
                title="Total Comments"
                count={isLoading ? "..." :
stats?.commentCount}
                icon={<FaComments />}
                color="warning"
              />
            </Col>
          </Row>

          {/* Recent Activity Section could go here */}
        </Col>
      </Row>
    </Container>
  );
};

export default AdminDashboard;
```

10. Admin Tags



Code:

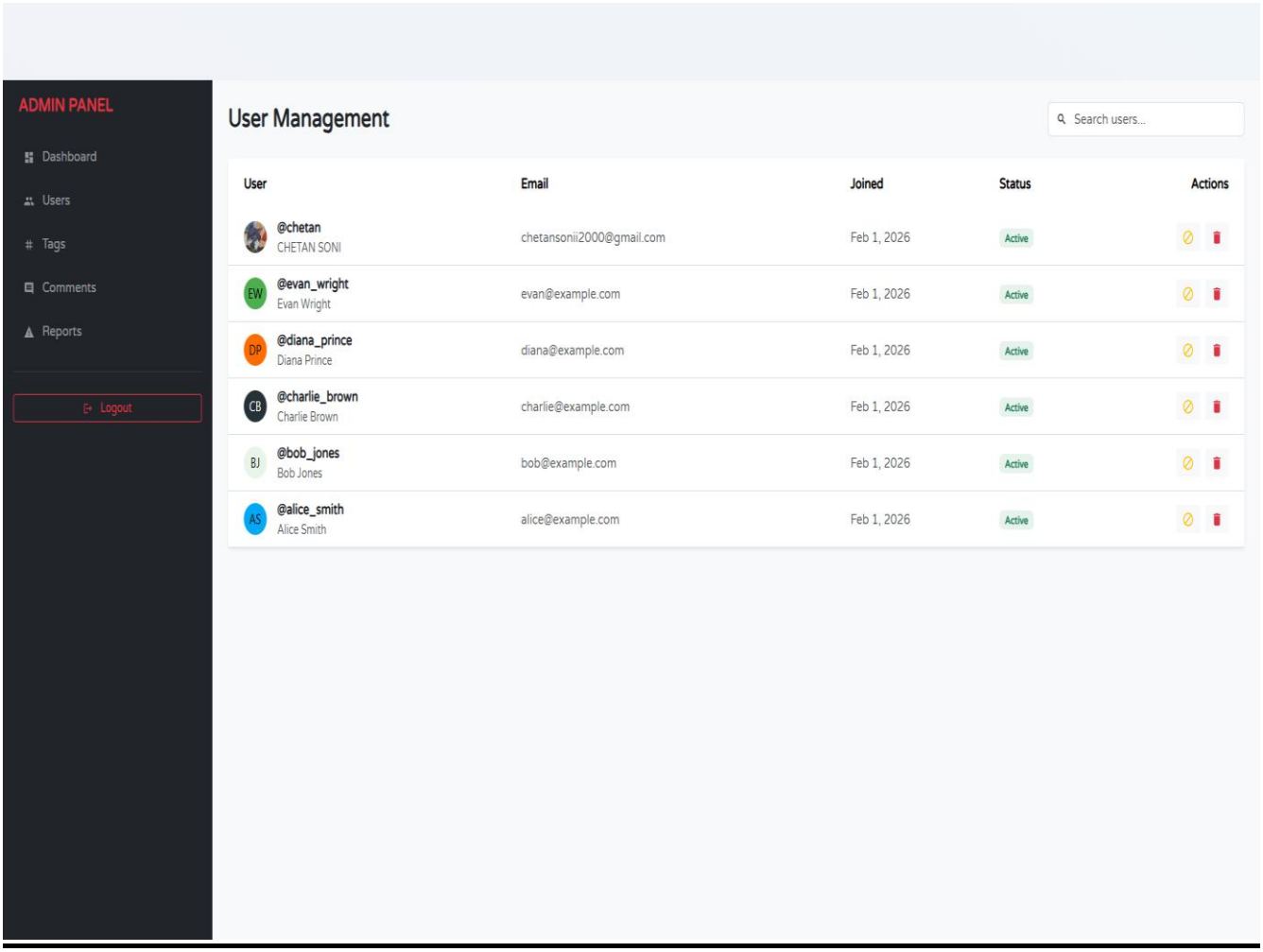
```
import { useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Container, Table, Button, Form, Row, Col }
from "react-bootstrap";
import { getAllTags, deleteTag, createTag } from
"../../redux/slices/adminSlice";
import { MdDelete } from "react-icons/md";

const AdminTags = () => {
  const dispatch = useDispatch();
  const { tags, isLoading } = useSelector((state) =>
state.admin);
  const [newTag, setNewTag] = useState("");
```

```
  useEffect(() => {
    dispatch(getAllTags());
  }, [dispatch]);

  const handleAddTag = (e) => {
    e.preventDefault();
    if (newTag.trim()) {
      dispatch(createTag({ name: newTag }));
      setNewTag("");
    }
  };

  return (
    <Container className="py-4">
      <div className="d-flex justify-content-between
align-items-center mb-4">
        <h3>Manage Tags</h3>
        <Form onSubmit={handleAddTag}
className="d-flex gap-2">
          <Form.Control
            placeholder="New Tag Name"
            value={newTag}
            onChange={(e) => setNewTag(e.target.value)}
          />
          <Button type="submit"
variant="primary">Add</Button>
        </Form>
      </div>

      <Table striped hover responsive
className="shadow-sm">
        <thead className="bg-light">
          <tr>
            <th>#</th>
            <th>Tag Name</th>
            <th>Usage Count</th>
            <th>Created By</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {tags?.map((tag, index) => (
            <tr key={tag._id}>
              <td>{index + 1}</td>
              <td><span className="badge bg-
secondary">{tag.name}</span></td>
              <td>{tag.topicsCount || 0}</td>
              <td>{tag.creator || "Admin"}</td>
```

```
            <td>
              <Button
                variant="outline-danger"
                size="sm"
                onClick={() =>
dispatch(deleteTag(tag._id))}
              >
                <MdDelete /> Delete
              </Button>
            </td>
          </tr>
        ))}
      </tbody>
    </Table>
  </Container>
 );
};

export default AdminTags;
```

---

11. Admin Users



Code:

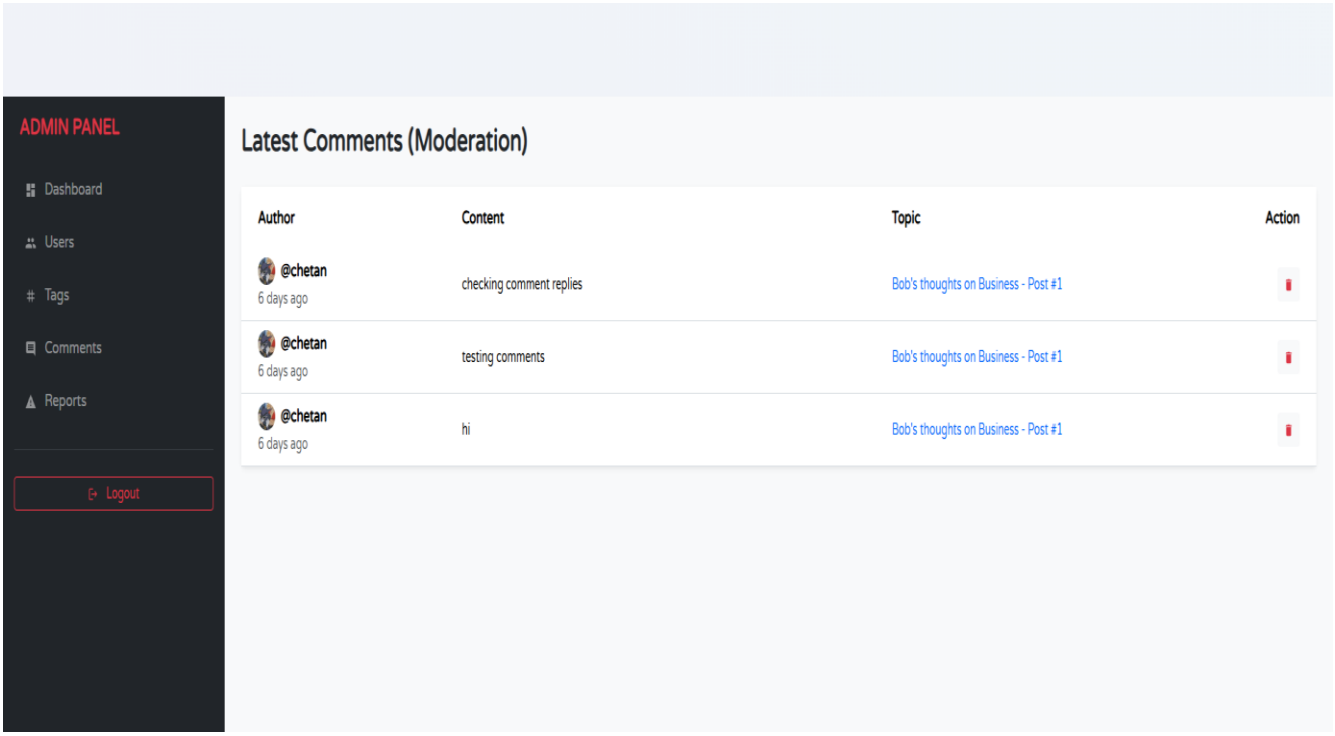
```
import { useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Container, Row, Col, Table, Button, Form,
Image, Badge } from "react-bootstrap";
import { getAllUsers, deleteUser } from
"../../redux/slices/adminSlice"; // Hypothetical Slice
import LeftSidebar from
"../../components/LeftSidebar/LeftSidebar";
import { FaTrash, FaSearch } from "react-icons/fa";

const AdminUsers = () => {
```

```
  const dispatch = useDispatch();
  const { users, isLoading } = useSelector((state) =>
state.admin);
  const [search, setSearch] = useState("");

  useEffect(() => {
    dispatch(getAllUsers());
  }, [dispatch]);

  // Filter users based on search
  const filteredUsers = users?.filter(user =>

user.username.toLowerCase().includes(search.toLower
Case()) ||

user.email.toLowerCase().includes(search.toLowerCase
())
  );

  return (
    <Container fluid className="py-4">
      <Row>
        <Col md={2} className="d-none d-md-block">
          <LeftSidebar adminMode={true} />
        </Col>

        <Col md={10}>
          <div className="d-flex justify-content-between
align-items-center mb-4">
            <h3>User Management</h3>
            <div className="d-flex align-items-center bg-
white p-2 rounded shadow-sm border">
              <FaSearch className="text-muted me-2" />
              <Form.Control
                type="text"
                placeholder="Search users..."
                className="border-0 p-0 shadow-none"
                value={search}
                onChange={(e) =>
setSearch(e.target.value)}
              />
            </div>
          </div>

          <div className="bg-white rounded shadow-sm
overflow-hidden">
            <Table responsive hover className="m-0
align-middle">
```

```
            <thead className="bg-light">
              <tr>
                <th className="py-3 ps-4">User</th>
                <th>Role</th>
                <th>Status</th>
                <th>Joined</th>
                <th className="text-end pe-
4">Actions</th>
              </tr>
            </thead>
            <tbody>
              {isLoading ? <tr><td colSpan="5"
className="text-center p-4">Loading...</td></tr> :
               filteredUsers?.map(user => (
                <tr key={user._id}>
                  <td className="ps-4">
                    <div className="d-flex align-items-
center">
                      <Image src={user.avatar?.url}
roundedCircle width={40} height={40}
className="me-3 border" />
                      <div>
                        <div className="fw-
bold">{user.username}</div>
                        <div className="small text-
muted">{user.email}</div>
                      </div>
                    </div>
                  </td>
                  <td>
                    <Badge bg={user.isAdmin ? "danger" :
"primary"}>
                      {user.isAdmin ? "Admin" : "Student"}
                    </Badge>
                  </td>
                  <td>
                    <Badge bg={user.isVerified ? "success" :
"warning"} text={!user.isVerified && "dark"}>
                      {user.isVerified ? "Verified" :
"Pending"}
                    </Badge>
                  </td>
                  <td className="text-muted small">{new
Date(user.createdAt).toLocaleDateString()}</td>
                  <td className="text-end pe-4">
                    {!user.isAdmin && (
                      <Button
                        variant="outline-danger"
```

```
                        size="sm"
                        onClick={() => {
                          if(window.confirm("Are you sure
you want to delete this user?"))
                            dispatch(deleteUser(user._id));
                        }}
                       >
                        <FaTrash /> Delete
                       </Button>
                    )}
                   </td>
                  </tr>
                ))}
               </tbody>
              </Table>
             </div>
            </Col>
           </Row>
          </Container>
        );
};

export default AdminUsers;
```

12. Admin Comments



Code:

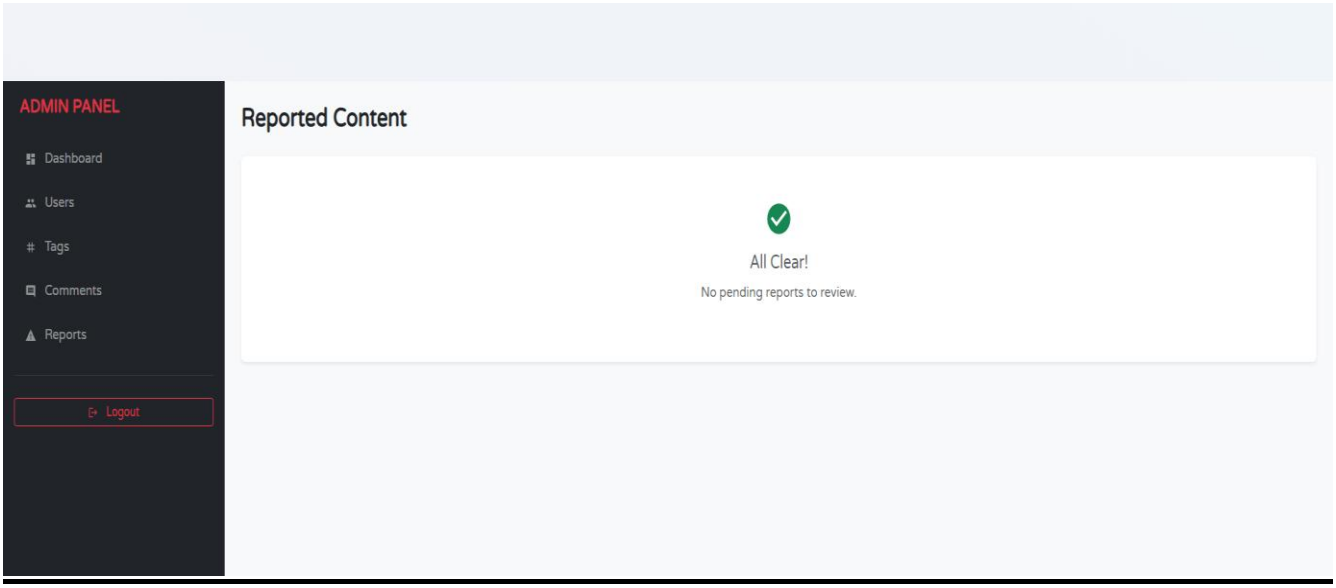
```
import { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Container, Row, Col, Table, Button } from
"react-bootstrap";
import { getAllComments, deleteComment } from
"../../redux/slices/adminSlice";
import LeftSidebar from
"../../components/LeftSidebar/LeftSidebar";
import { MdDeleteForever } from "react-icons/md";
import { Link } from "react-router-dom";

const AdminComments = () => {
  const dispatch = useDispatch();
  const { comments, isLoading } = useSelector((state)
=> state.admin);

  useEffect(() => {
    dispatch(getAllComments());
  }, [dispatch]);
```

```
 return (
  <Container fluid className="py-4">
   <Row>
    <Col md={2} className="d-none d-md-block">
     <LeftSidebar adminMode={true} />
    </Col>

    <Col md={10}>
     <h3 className="mb-4">Content Moderation
(Comments)</h3>

     <Table striped hover responsive className="bg-
white shadow-sm rounded">
      <thead>
       <tr>
        <th style={{ width: "40%" }}>Comment
Content</th>
        <th>Author</th>
        <th>Topic</th>
        <th>Date</th>
        <th>Action</th>
       </tr>
      </thead>
      <tbody>
       {comments?.map(comment => (
        <tr key={comment._id}>
         <td>
          <div className="text-truncate" style={{
maxWidth: "300px" }}>
           {comment.content}
          </div>
         </td>
         <td>@{comment.author?.username ||
"Deleted User"}</td>
         <td>
          <Link
to={`/topics/${comment.topic?._id}/slug`}
target="_blank" className="text-decoration-none">
           View Topic
          </Link>
         </td>
         <td className="small text-muted">{new
Date(comment.createdAt).toLocaleDateString()}</td>
         <td>
          <Button
           variant="danger"
           size="sm"
```

```
                    onClick={() =>
dispatch(deleteComment(comment._id))}
                  >
                    <MdDeleteForever /> Remove
                  </Button>
                </td>
              </tr>
            ))}
          </tbody>
        </Table>
      </Col>
    </Row>
  </Container>
 );
};

export default AdminComments;
```

---

13. Admin Reports



Code:

```
import { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { Container, Row, Col, Card, Button, Badge }
from "react-bootstrap";
import { getReports, resolveReport,
deleteReportedContent } from
"../../redux/slices/adminSlice";
import LeftSidebar from
"../../components/LeftSidebar/LeftSidebar";

const AdminReports = () => {
  const dispatch = useDispatch();
  const { reports, isLoading } = useSelector((state) =>
state.admin);

  useEffect(() => {
    dispatch(getReports());
  }, [dispatch]);

  return (
    <Container fluid className="py-4">
      <Row>
        <Col md={2} className="d-none d-md-block">
```

```
        <LeftSidebar adminMode={true} />
      </Col>

      <Col md={10}>
        <h3 className="mb-4 text-danger">Flagged
Content Reports</h3>

        <div className="d-flex flex-column gap-3">
          {reports?.length === 0 ? <p className="text-
muted">No pending reports.</p> :
           reports?.map(report => (
            <Card key={report._id} className="border-
danger shadow-sm">
              <Card.Header className="bg-danger text-
white d-flex justify-content-between">
                <span>Reason:
<strong>{report.reason}</strong></span>
                <Badge bg="light"
text="dark">{report.status}</Badge>
              </Card.Header>
              <Card.Body>
                <Row>
                  <Col md={8}>
                    <h6 className="text-
muted">Reported Content:</h6>
                    <p className="p-2 bg-light border
rounded">
                      {report.contentType === 'Topic' ?
report.reportedItem?.title :
report.reportedItem?.content}
                    </p>
                    <small className="text-muted">
                      Reported by:
@{report.reporter?.username} • Target:
@{report.reportedUser?.username}
                    </small>
                  </Col>
                  <Col md={4} className="d-flex flex-
column justify-content-center gap-2">
                    <Button
                      variant="outline-success"
                      onClick={() =>
dispatch(resolveReport(report._id))}
                    >
                      Ignore & Resolve
                    </Button>
                    <Button
                      variant="dark"
```

```
                        onClick={() =>
dispatch(deleteReportedContent({
                          reportId: report._id,
                          itemId: report.reportedItem?._id,
                          type: report.contentType
                        }))}
                      >
                        Delete Content & Resolve
                      </Button>
                    </Col>
                  </Row>
                </Card.Body>
              </Card>
            ))}
          </div>
        </Col>
      </Row>
    </Container>
  );
};

export default AdminReports;
```

LIMITATIONS

While **Chetan College Forum** successfully achieves its primary objectives of fostering a collaborative academic environment, there are certain limitations in the current version of the system:

1. **Lack of Multi-Language Support:** The platform currently operates solely in English. Given the diverse linguistic background of students in many colleges, this limits accessibility for those who might prefer discussing complex academic concepts in their native regional languages (e.g., Hindi, Marathi).
2. **Absence of Private Messaging (Direct Chat):** The system currently focuses on public discussions via Topics and Comments. It does not yet offer a real-time private messaging (DM) feature. Students wishing to discuss sensitive matters or collaborate one-on-one must rely on external communication tools, which breaks the continuity of the platform experience.
3. **Basic Content Recommendation Algorithm:** The current "Popular" and "Latest" feeds rely on simple sorting logic based on timestamps and view counts. The platform lacks an intelligent, AI-driven recommendation engine that could suggest topics based on a student's specific reading history, joined Spaces, or past upvoting behavior.
4. **Limited File Sharing Capabilities:** While users can upload images (avatars) via Cloudinary, the platform does not yet support the direct uploading and sharing of heavy academic documents, such as PDF notes, past year question papers, or project zip files, which are essential for a complete academic resource hub.

FUTURE SCOPE OF CHETAN COLLEGE FORUM

The current implementation of the Chetan College Forum serves as a robust foundation. However, there is significant potential for expansion and enhancement in future iterations:

1. **Real-Time Chat & Collaboration:** Integrating **Socket.io** to enable real-time features such as private messaging between students and group chats within specific "Spaces." This would allow for instant study group coordination without leaving the platform.
2. **Mobile Application Development:** Developing a native mobile application using **React Native**. Since the backend is already built as a RESTful API (Node.js/Express), it can easily serve a mobile app, providing students with push notifications for exam alerts, replies, and trending polls directly on their smartphones.
3. **AI-Powered Content Moderation & Tagging:** Integrating AI libraries (like TensorFlow.js) to automatically detect and flag inappropriate content or spam. Additionally, AI could be used to auto-suggest tags for new topics based on the content text, ensuring better categorization and searchability.
4. **Academic Resource Repository:** Expanding the "Spaces" feature to include a dedicated **Document Repository**. This would allow students to upload, organize, and download lecture notes, eBooks, and research papers, turning the forum into a complete Learning Management System (LMS) component.
5. **Gamification and Reputation System:** Enhancing the current "Top Contributors" feature by introducing badges (e.g., "Problem Solver," "Verified Tutor") and a reputation point system. This would incentivize students to provide high-quality answers and remain active on the platform.

CONCLUSION

The **Chetan College Forum** project successfully addresses the critical need for a centralized, structured, and interactive digital community for students. By bridging the gap between social networking and academic collaboration, the platform provides a dedicated space where knowledge can be shared, validated, and preserved.

The development process demonstrated the power and flexibility of the **MERN Stack** (MongoDB, Express.js, React.js, Node.js). The use of **React** ensured a dynamic, single-page user interface that is responsive across devices, while **Redux** provided efficient state management for complex features like real-time polling and authentication. On the backend, **Node.js** and **MongoDB** proved to be a highly scalable combination, capable of handling diverse data types—from unstructured discussion text to structured user profiles.

Key features such as **Spaces** for categorization, **Polls** for community opinion, and the **Upvote/Downvote** mechanism for quality control have created a democratized environment where the best content naturally rises to the top. The implementation of secure authentication using **JWT** and **Bcrypt** ensures that the platform remains safe and trustworthy for all students.

In conclusion, Chetan College Forum is not just a discussion board; it is a scalable, modern solution for campus interaction. With its solid architectural foundation and clear path for future upgrades, it is poised to become an indispensable tool for student success and peer-to-peer learning.

BIBLIOGRAPHY

The following resources, documentation, and literature were referred to during the design, development, and implementation of this project:

Books:

1. *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node* by Vasan Subramanian.
2. *JavaScript: The Good Parts* by Douglas Crockford.
3. *Software Engineering: A Practitioner's Approach* by Roger S. Pressman.

Official Documentation & Websites: 4. **React.js Documentation** - https://react.dev 5. **Node.js Documentation** - https://nodejs.org/en/docs 6. **MongoDB Manual** - https://www.mongodb.com/docs 7. **Redux Toolkit** - https://redux-toolkit.js.org 8. **Bootstrap (React-Bootstrap)** - https://react-bootstrap.github.io 9. **MDN Web Docs (Mozilla)** - https://developer.mozilla.org

Tools & Libraries: 10. **Cloudinary (Image Management)** - https://cloudinary.com 11. **JWT (JSON Web Tokens)** - https://jwt.io 12. **Axios (HTTP Client)** - https://axios-http.com 13. **React Icons** - https://react-icons.github.io/react-icons 14. **Draw.io (System Diagrams)** - https://app.diagrams.net