

# **UIT2739 – FULL STACK DEVELOPMENT**

## **A PROJECT REPORT**

*on*

### **Resource Sharing Portal**

*Submitted by*

**Ashwin Kumar S – 3122 22 5002 014**

**Aswath Karthik – 3122 22 5002 015**

**Dunya Syed Hassan – 3122 22 5002 032**



**Department of Information Technology**

**Sri Sivasubramaniya Nadar College of Engineering**

**(An Autonomous Institution, Affiliated to Anna University)**

**Rajiv Gandhi Salai (OMR), Kalavakkam – 603 110**

**NOVEMBER 2025**

**SRI SIVASUBRAMANIYA NADAR COLLEGE OF  
ENGINEERING**



**Department of Information Technology**

**CERTIFICATE**

**Certified that this project titled “Resource Sharing Portal” is the bonafide work of “Ashwin Kumar S– 3122 22 5002 014, Aswath Karthik – 3122 22 5002 015, Dunya Syed Hassan – 3122 22 5002 032”, and is submitted for project review on 24<sup>th</sup> November 2025.**

**Place : Kalavakkam**

**Date : 24/11/2025**

**Internal Examiner**

## TABLE OF CONTENTS

S. No	Title	Page No.
1	Project Overview	1
2	Project Requirements	2
2.1	Wireframes/ Screenshots	2
2.2	Functional Requirements and Use Cases	5
3	Technical Details	11
3.1	Tech Stack	11
3.2	Architecture Diagram	13
3.3	Design Patterns	15
4	Conclusion	17

# 1. Project Overview

The Student Resource Sharing Portal is a flexible, community-driven platform built to help students exchange academic materials with ease. Instead of relying on scattered group chats or inconsistent cloud folders, the portal centralizes everything—notes, assignments, textbooks, reference links, presentations, code snippets, and even multimedia files—into one organized, searchable hub.

Under the hood, the system runs on a clean Node.js/Express backend paired with a fast React + Vite frontend. Students can sign up, authenticate securely, and immediately begin contributing resources. The backend manages user identities, stores metadata for all uploaded resources, and handles file uploads smoothly through dedicated middleware. MongoDB serves as the storage backbone, keeping track of users, files, and access permissions.

On the front end, the UI is intentionally simple and familiar. Students can upload new resources, browse contributions from their peers, or quickly search for something specific—maybe last semester’s lab manual or an important formula sheet right before an exam. Components like, category filters, and responsive layouts make the platform feel intuitive rather than overwhelming.

## 2. Project Requirements

The Requirements section defines the essential capabilities, features, and constraints that the Student Resource Sharing Portal must deliver to satisfy user needs and project objectives. It outlines both functional and non-functional specifications.

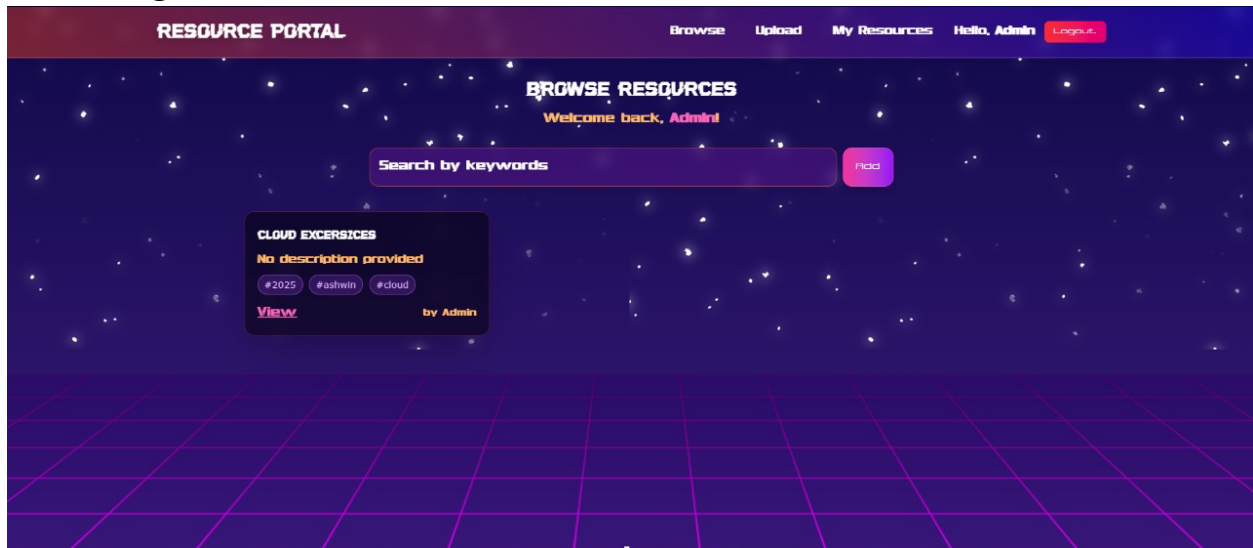
### 2.1 Wireframes/ Screenshots

#### 2.1.1 Wireframes

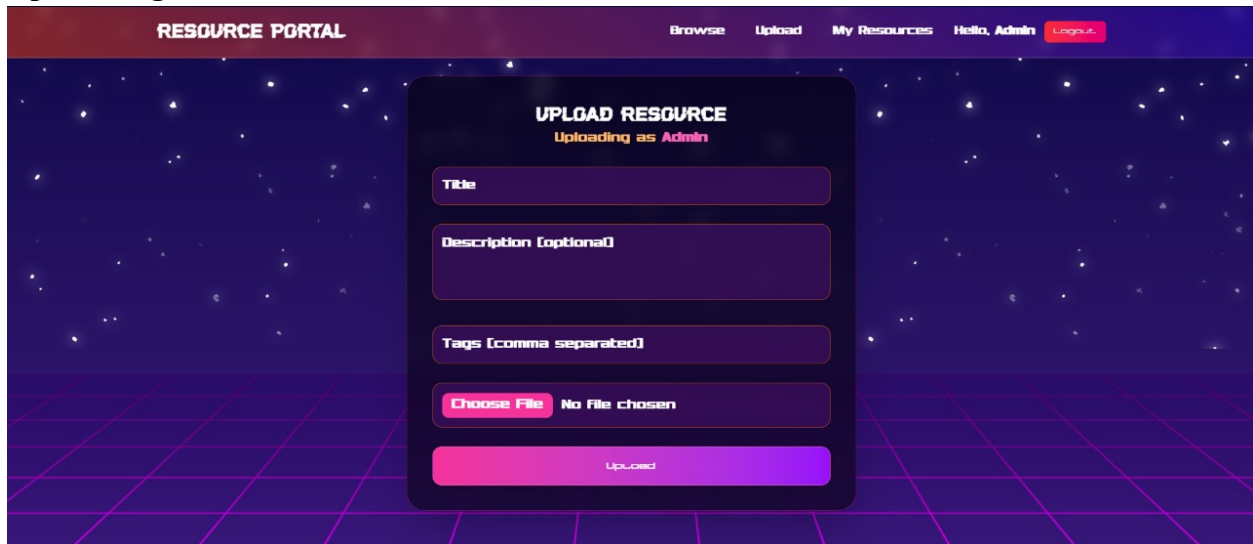


## 2.1.2 Screenshots

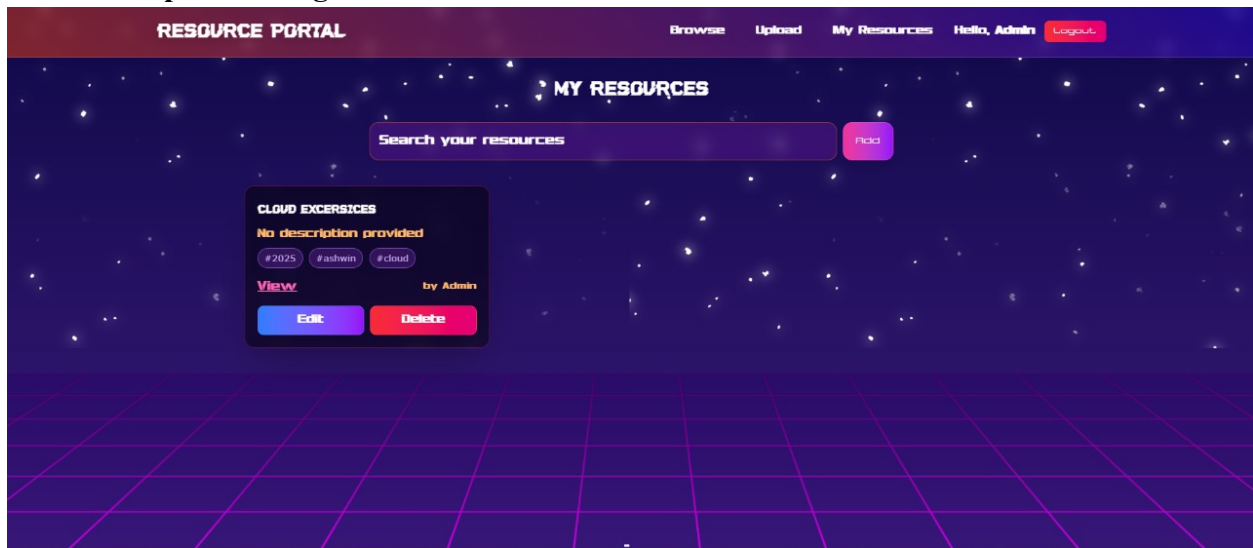
### Browse Page



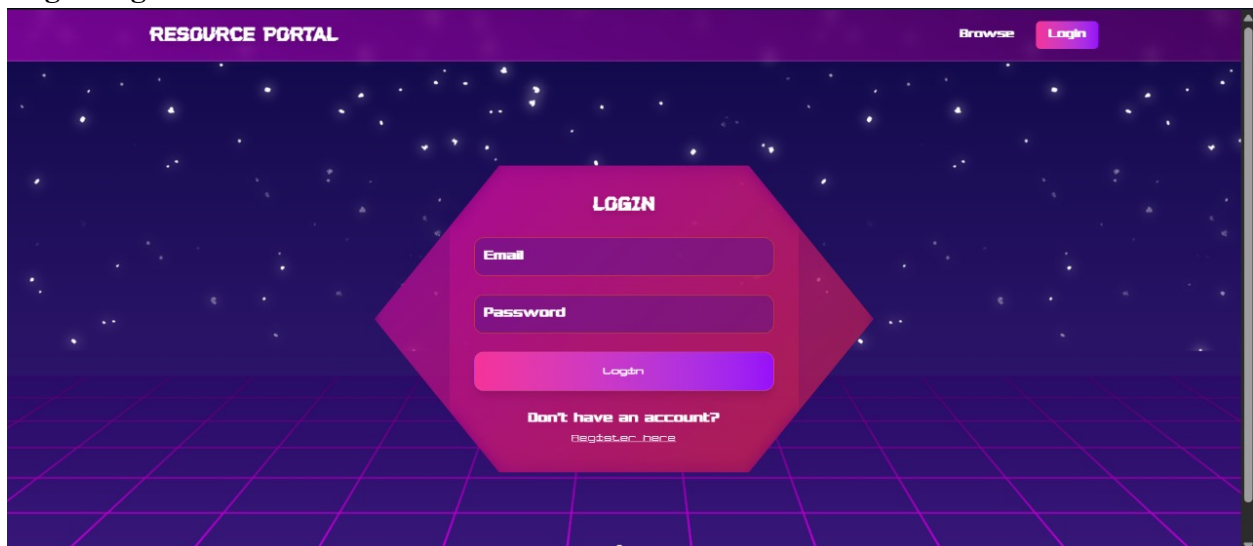
### Upload Page



## Resource Updation Page



## Login Page



## 2.2 Requirements and Use Cases

### 2.2.1 Functional Requirements:

#### 1. User Authentication

- The system must allow students to register with basic credentials (username, email, password).
- The system must allow existing users to log in.
- The backend must generate and validate JWT tokens for authentication.
- Only authenticated users should be able to upload, delete, or manage their resources.

#### 2. Resource Management

- The system must allow users to upload various types of resources (PDF, images, ZIP files, notes, etc.).
- The system must save uploaded resource metadata in the database, including:
  - Title
  - Description
  - File path
  - Uploaded by
  - Timestamp
- Students must be able to view all uploaded resources.
- Users must be able to view only their own uploaded resources under **“My Resources.”**
- The system must allow users to delete their own uploaded resources.

#### 3. Browsing & Searching

- The application must display a list/grid of available resources.
- Students must be able to search resources by keyword.
- The system must fetch resources from the backend using REST APIs.

#### **4. Role-Based Access & Authorization**

- Only logged-in users may upload and delete files.
- Non-authenticated users may only browse public resources (based on the current code structure, browsing is unrestricted).

#### **5. User Interface**

- The frontend must provide:
  - A navigation bar
  - Pages for Login, Register, Browse, Upload, My Resources
- A ResourceCard component must display:
  - Title
  - Description snippet
  - File type
  - Download/view link

#### **6. File Handling**

- The backend must process file uploads using dedicated middleware (upload.js).
- Uploaded files must be stored in the server's filesystem (Multer-based implementation).

#### **7. Database Integration**

- MongoDB must store user information.
- MongoDB must store details of uploaded resources.
- The backend must connect through db.js.

### **2.2.2 Non-Functional Requirements**

#### **1. Performance**

- Search and resource listing should return results quickly even with many uploads.
- File uploads must complete within a reasonable time under typical network conditions.

- API responses should be optimized for low latency.

## **2. Scalability**

- The system should support increasing numbers of users and resources.
- The architecture should allow easy extension—adding categories, tags, ratings, etc.

## **3. Security**

- Passwords must be securely hashed before storage.
- Authentication must use JWT tokens.
- File uploads must be validated to prevent malicious file types.
- Only authorized users should modify or delete resources.

## **4. Usability**

- The UI must be clean, minimal, and intuitive.
- Navigation should be simple for students under exam stress or tight deadlines.
- Pages should work on both desktop and mobile browsers.

## **5. Reliability & Availability**

- The backend must handle failures gracefully (e.g., DB or network issues).
- Uploaded files should remain accessible without corruption.

## **6. Maintainability**

- Code should follow modular structure (routes, middleware, models separated).
- Components in the frontend should be reusable and logically separated.

## **7. Compatibility**

- The portal must run on modern browsers (Chrome, Firefox, Edge).
- API communication must follow REST standards (JSON format).

### 2.2.3 Use Case Description

#### Use Case 1: Register

**Actor:** Student

**Description:** A new student creates an account to use the portal.

**Steps:**

1. Student enters name, email, password.
2. System validates input.
3. User data is stored in MongoDB.
4. Confirmation message shown.

#### Use Case 2: Log In

**Actor:** Student

**Description:** Student logs into their account to access upload and management features.

**Steps:**

1. User enters credentials.
2. Backend verifies credentials.
3. JWT token is returned.
4. User is redirected to the Browse page.

#### Use Case 3: Upload Resource

**Actor:** Authenticated Student

**Description:** Student uploads a new learning resource.

**Steps:**

1. User navigates to Upload Page.
2. User enters title, description, selects a file.
3. File is uploaded through middleware.
4. Metadata is stored in MongoDB.

5. Resource appears in Browse + My Resources pages.

#### **Use Case 4: Browse Resources**

**Actor:** Any student

**Description:** User views all shared resources.

**Steps:**

1. User opens Browse page.
2. System fetches all resources via API.
3. Resources displayed as cards.

#### **Use Case 5: Search Resources**

**Actor:** Student

**Description:** User searches for specific notes or files.

**Steps:**

1. Student enters a keyword.
2. Frontend sends search query to API.
3. Matching resources returned.

#### **Use Case 6: View My Resources**

**Actor:** Authenticated Student

**Description:** User sees only the resources they uploaded.

**Steps:**

1. User navigates to My Resources.
2. Backend filters by `uploadedBy`.
3. List displayed.

### **Use Case 7: Delete Resource**

**Actor:** Authenticated Student

**Description:** User removes a previously uploaded resource.

**Steps:**

1. User clicks Delete on a resource they uploaded.
2. Backend verifies user's ownership.
3. Resource is removed from DB and file storage.

## **3. Technical Details**

### **3.1 Tech Stack**

#### **Stack used: MERN Stack**

The Student Resource Sharing Portal is developed using the MERN stack, which comprises the following core technologies:

#### **MongoDB Atlas (Cloud):**

Provides a cloud-hosted NoSQL database that efficiently stores user information, resource metadata, and access permissions. Its scalability makes it ideal for dynamic and growing academic resource collections.

#### **Express.js:**

Acts as the backend web application framework, managing API endpoints, server-side authentication, and file upload logic, which ensures secure and reliable handling of user actions.

#### **React.js (with Vite):**

Powers the frontend, allowing for a modern, interactive, and highly responsive user interface. Vite further accelerates development by providing fast refreshes and optimized builds.

#### **Node.js:**

Serves as the runtime environment for the backend, leveraging JavaScript's asynchronous capabilities for robust processing of file uploads, downloads, and multi-user operations.

### **Supporting Infrastructure & Deployment**

#### **Cloudinary/Amazon S3:**

All uploaded files, such as documents and multimedia, are managed and delivered through Cloudinary or Amazon S3, assuring high availability and efficient media handling.

**Render:**

Enables automated deployment and scaling of the Node.js/Express backend, ensuring consistent performance.

**Vercel:**

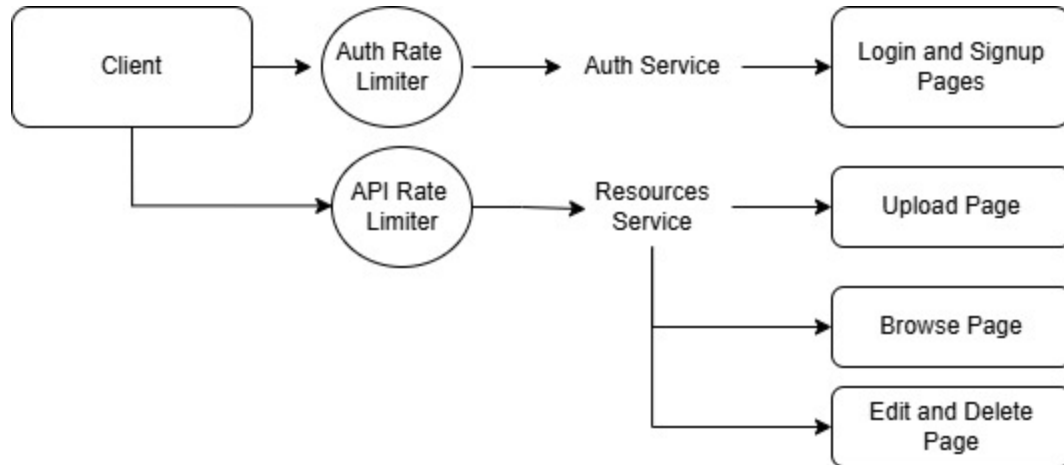
Facilitates serverless deployment of the React frontend, providing fast, reliable global delivery and seamless integration with code repositories.

**Advantages of using MERN Stack**

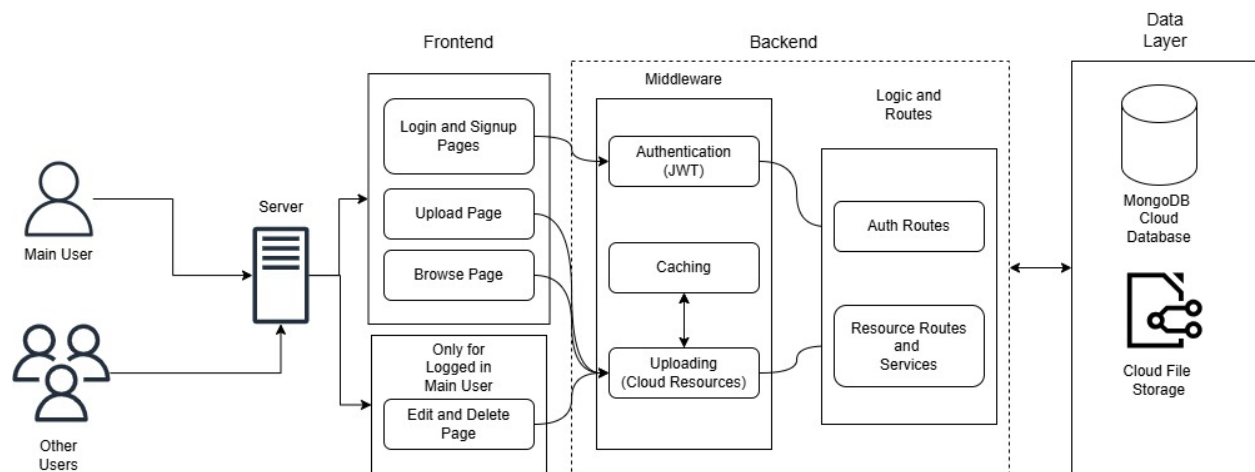
- **Unified Language:** MERN leverages JavaScript for both client and server development, streamlining the workflow and simplifying maintenance.
- **Modularity & Reusability:** React's component-based architecture and the modularity of Node.js ecosystems support rapid feature development and cleaner codebase management.
- **Performance & Scalability:** The non-blocking, event-driven model of Node.js and the scalability of MongoDB Atlas ensure the system performs well under high load or during resource-intensive operations.
- **Modern Development Practices:** Cloud-first storage (MongoDB Atlas, S3/Cloudinary) and serverless deployment (Vercel, Render) help the platform scale with institutional needs while reducing operational overhead.
- **Community and Support:** Each part of the MERN stack is open-source, backed by large, active communities, which guarantees ongoing support, access to robust libraries, and frequent security updates.

## 3.2 Architecture Diagram

### 3.2.1 Components and Service Diagram



### 3.2.2 Architecture Diagram



### 3.2.2 Architecture Description

The system is designed using a client–server architecture that consists of three primary layers: the Frontend, Backend, and Data Layer. Middleware components are integrated within the backend to support authentication, caching, and file upload operations.

## **Frontend:**

The frontend provides an interactive interface for users to browse, upload, and manage content. It includes:

- **Login and Signup Pages** – Allow users, especially the main user, to securely register and log into the system.
- **Upload Page** – Enables authenticated users to upload cloud resources such as files.
- **Browse Page** – Lets all users (including other public users) view available resources.
- **Edit and Delete Page** – Accessible exclusively to the main logged-in user for managing their uploaded content.

User interactions with these pages are directed to the server for backend processing.

## **Backend:**

The backend handles core application logic, security, and communication with the data layer. It consists of:

### **Middleware Modules:**

- **Authentication (JWT)** – Validates user sessions via JSON Web Tokens, ensuring secure access control for restricted pages.
- **Caching** – Improves performance by temporarily storing frequently accessed data.
- **Uploading (Cloud Resources)** – Manages and routes file uploads to cloud storage services.

### **Logic and Routing:**

- **Auth Routes** – Handle user authentication processes (login, signup, token validation).

- **Resource Routes and Services** – Manage file uploads, retrievals, updates, and deletions with proper permission checks.

## **Data Layer:**

The system relies on cloud infrastructure to persist and manage user and application data:

- **MongoDB Cloud Database** – Stores user profiles, authentication data, and metadata about uploaded resources.
- **Cloud File Storage** – Stores actual user-uploaded files efficiently and securely.

## **Users:**

Two types of users interact with the system:

- **Main User** – Has full privileges including upload, edit, and delete operations.
- **Other Users** – Can browse and view shared content but cannot modify it.

## **Data Flow Summary:**

1. Users interact with frontend pages.
2. All requests are processed by the backend server.
3. Middleware performs authentication and caching as needed.
4. Database queries or cloud storage operations occur through backend service routes.
5. Responses are returned to the user interface to update the experience.

## **3.3 Design Patterns**

This project adopts several well-established design patterns to ensure a clean architecture, maintainable code, and scalable functionality:

### **3.3.1 Strategy Pattern**

The Strategy Pattern is utilized for resource editing logic on the frontend and backend. When a resource (document) requires operations such as edit or delete, different strategies are applied depending on user actions. For example, the UI presents options to edit or delete documents, and upon selection, the correct strategy (edit or delete function) is invoked. This allows for flexible future expansion if document types or modification actions increase.

### **3.3.2 Data Access Object (DAO) Pattern**

The Data Access Object Pattern is applied in the backend to handle database operations. All resource and user data interactions with MongoDB are funneled through DAO classes or modules. This abstraction cleanly separates the logic that accesses the database from the core business logic, making code easier to maintain, debug, and scale. It simplifies the swapping or upgrading of data sources in the future without altering the entire application logic.

### **3.3.3 Chain of Responsibility Pattern**

The Chain of Responsibility Pattern underpins the middleware system in the Express.js backend. User requests (such as uploads and authentication) pass through a series of middleware layers—such as authentication, validation, rate limiting, file parsing, and error handling. Each middleware addresses a specific concern before passing control to the next, resulting in modular, testable, and easily maintainable request processing flows.



## 4. Conclusion

The Student Resource Sharing Portal successfully demonstrates how a lightweight, modern web application can streamline academic collaboration within a student community. By integrating secure user authentication, an efficient resource upload system, and a clean, responsive interface, the platform eliminates the common difficulties associated with scattered file-sharing methods such as messaging groups or personal cloud links. The system allows students to easily contribute, access, and manage a wide variety of learning materials in one central hub, ultimately encouraging knowledge sharing and peer support.

From a technical perspective, the project showcases a well-organized full-stack architecture built using Node.js, Express, MongoDB, and React with Vite. Modular backend design, RESTful APIs, reusable frontend components, and consistent state management collectively ensure maintainability and future scalability. The system is flexible enough to be expanded with additional features such as course-based categorization, commenting, upvoting, or even real-time chat, depending on the needs of the academic environment.

Overall, the portal meets its primary objective: to provide students with a reliable, intuitive, and secure platform for exchanging educational resources. With further enhancement and broader deployment, the application has the potential to evolve into a valuable academic tool that strengthens collaboration and enhances the overall learning experience.