***15/5/25***

Project Goal:

To create a web-based and cloud-integrated platform where students can access syllabus, notes, previous exam papers, and other study material in a structured way. The portal will initially be developed for the Computer Science (CS) department of your college, covering the following courses:

1. B.Sc. (CS), 2. B.C.A. (Science), 3. M.Sc. (CS), 4. M.Sc. (CA)

Key Features:

* Structured Access to Study Materials: Organized by course, subject, year, and semester.
* Search Bar:
  + Quick access to materials.
  + Users should be able to search for any subject and get matching results.
  + Partial search feature (e.g., "cyb" shows "Cyber Security").
  + Displays a list of matching subjects with:
    - * 📚 Subject Name
      * 🎓 Course Name
      * 📅 Year & Semester
  + Clicking a search result should take users to the Study Material for that subject.
* Login/Signup: Separate login options for students and admins (teachers).
* Student Features:
  + Access to study materials.
  + View PDF before downloading.
  + Save materials in their account.
  + Personal storage space for saving their own notes and materials.
* Admin (Teacher) Features:
  + All student features (viewing, saving, downloading, storing).
  + Ability to add materials to respective courses.
* Cloud-Based Infrastructure:
  + Database: MongoDB Atlas (cloud-based NoSQL database).1
  + Cloud Storage: To be decided between Firebase Storage and AWS S3 for storing study materials.

Technology Stack:

1. Frontend: React with Vite
   * React for building the dynamic user interface with a component-based architecture.
   * Vite as a fast and efficient build tool for a streamlined development experience.2
2. Backend: Python with Flask
   * Python as the backend language for building the API.
   * Flask as a lightweight and flexible microframework for creating the API endpoints.3
3. Database: MongoDB Atlas
   * A cloud-based NoSQL database for storing application data (users, courses, materials, etc.).
4. Cloud Storage:

To be decided, with a slight inclination towards Firebase Storage for easier initial integration with the frontend and a generous free tier for a college project. However, AWS S3 is also a viable option offering scalability and industry-standard features.4

Development Steps (General Outline):

1. Detailed Requirements Gathering: Further refine user stories and content structure.
2. UI/UX Design: Create wireframes and mockups of key pages.
3. Backend Setup (Python/Flask):
   * Set up the Flask project structure.
   * Define API endpoints for user authentication, fetching materials, search functionality, and admin material upload.
   * Integrate with MongoDB Atlas.
4. Frontend Setup (React/Vite):
   * Create the React project using Vite.
   * Develop UI components for login, material listing, search, personal space, and admin dashboard.
   * Implement routing and state management.
   * Integrate with the backend API endpoints.
5. Cloud Storage Integration (Firebase Storage or AWS S3):
   * Set up the chosen cloud storage service.
   * Implement backend logic for handling file uploads by admins and retrieving file URLs for users.
   * Potentially integrate frontend for direct uploads with Firebase Storage (with appropriate security rules).
6. Implement Core Features:
   * Login/Signup functionality.
   * Displaying study materials based on course and subject.
   * Search bar implementation (frontend UI and backend API logic).
   * Personal storage space for users.
   * Admin material upload functionality.
   * PDF viewing capability on the frontend.
7. Testing: Thoroughly test all features and user flows.
8. Deployment: Deploy the frontend and backend to suitable hosting platforms.

Considerations for You (Working Alone):

* Start Simple: Focus on the core functionalities first (e.g., displaying materials, basic search, user login).
* Iterative Development: Build the project in smaller, manageable increments.5
* Prioritize Learning: This is a great opportunity to learn these technologies. Don't be afraid to experiment and seek out resources.
* Time Management: Plan your time effectively to cover all aspects of the project.

----------------------------------------------------------------------------------------------------------------

1. **Really understand what each button and page will do.** Imagine using the website yourself – what would you click, and what should happen next?
2. **Sketch out the basic look of the website and the important pages.** It doesn't have to be perfect, just rough drawings of where things will go (like the search bar, lists of materials, login boxes).
3. **Decide how the information (like student details, course names, and study materials) will be stored.** You've already chosen MongoDB, so think about the different "buckets" or collections you'll need.
4. **Plan the basic "rules" for how the website will work.** For example, who can log in (students and teachers), what can each person do, and how will the search actually find things.
5. **Get your "tools" ready.** Make sure you have Python, Flask, React, Vite, and you can connect to MongoDB Atlas.

1. **Break down the big project into smaller, bite-sized tasks.** Instead of "build the whole website," think "build the login page," "make the search bar work," etc.

--------------------------------------------------------------------------------------------------------------------------------------

 **Detailed Requirements Gathering:** We can go deeper into the user stories and the structure of the content.

 **UI/UX Design (Sketching):** We can start with some basic wireframes for the key pages.

 **Backend Setup:** We can begin setting up your Flask project and connecting to MongoDB Atlas.

 **Frontend Setup:** We can get your React project up and running with Vite.

--------------------------------------------------------------------------------------------------------------------------------------

***18/5/2025***

GitHub repository (https://github.com/sakshinde14/sp.git).

Completed Features:

* Project Setup:
  + You have a React frontend and a Flask backend.
  + You're using MongoDB for data storage (although we haven't fully integrated it into the course selection flow yet).
* User Authentication:
  + Students can sign up and log in.
  + Admins can log in.
  + You have basic authentication routes in your Flask backend.
* Course Selection Flow (Basic Structure):
  + The user can select a course.
  + After selecting a course, the user can select a year.
  + After selecting a year, the user can select a semester.
  + After selecting a semester, the subjects for that course, year, and semester are displayed.
* API Endpoints:
  + You have Flask API endpoints to:
    - * Get a list of courses.
      * Get years for a course.
      * Get semesters for a course and year.
      * Get subjects for a course, year, and semester.
* React Components:
  + You have React components to:
    - * Display the list of courses (CourseList).
      * Display the list of years (YearList).
      * Display the list of semesters (SemesterList).
      * Display the list of subjects (SubjectList).
* UI Layout:
  + You have a basic dashboard layout with:
    - * Top navigation.
      * A search bar.
      * Conditional rendering of components based on the course selection flow.
* Basic Styling:
  + You've implemented a light purple theme using CSS.

***Areas for Further Development/Next Steps:***

* Search Functionality: Implement the search bar to actually filter subjects or study materials.
* Study Material Display:
  + Fetch and display actual study materials (e.g., links, files) for the selected subjects. This will likely require storing study material information in your database and creating new API endpoints and React components.
* Data Persistence:
  + Currently, the course, year, semester, and subject data is hardcoded in the Flask backend. You'll need to store this data in your MongoDB database to make it dynamic and manageable.
* Error Handling and Feedback:
  + Improve error handling and provide better feedback to the user (e.g., loading indicators, success messages, more informative error messages).
* UI Enhancements:
  + Refine the UI/UX to make it more user-friendly and visually appealing.
* Responsiveness:
  + Ensure the UI is responsive and works well on different screen sizes.
* Testing:
  + Write tests for both the frontend and backend to ensure the application is working correctly and to prevent regressions.
* Security:
  + Implement proper security measures, especially around authentication and data handling.

--------------------------------------------------------------------------------------------------------------------------------------