Project Report for "Intra College Social Networking Application"

This report outlines the comprehensive plan for developing a web-based social networking and academic organization application for a college community. The project will be built by a beginner team with a focus on a clear roadmap, a well-defined technology stack, and a modular approach to development.

---

1. Functional Requirements 🎯

The application will fulfill a dual purpose: a social networking platform for communication and a functional tool for academic organization.

User Roles and Authentication

- User Registration: A custom sign-up form will allow users to register with an email, password, and their designated role (Student or Teacher).
- Student Profile: Profiles will include academic details like course and major, along with public social information such as a bio, followers/following count, and recent posts.
- Teacher Profile: Profiles will show subjects taught and will have the capability to be designated as an "Admin" or "Club Head" for specific permissions.

Social and Communication Features

- News Feed: A unified feed for both roles, displaying posts, notices, and events.
- Post Content: Users can post text, images, and videos. An AI integration will provide suggestions for writing posts.
- Post Interactions: Users can react to posts with a variety of emojis, comment on them, and share them.

- Direct Messaging (DM): A crucial feature for real-time communication, supporting one-on-one and group chats between students and teachers.
- Follower System: A standard follower/following system will allow students and teachers to build their network within the college.

Academic and Organizational Features

- "Your Course" Hub: A personalized section for students to view assignments, notes, and timetables specific to their course and major.
- Assignment Management:
    - Teacher Functionality: Teachers can create assignments by selecting their course and major from a list, adding assignment details, and setting a due date.
    - Student Functionality: Students can view assignments specific to their registered course and major. They can submit their work and mark the assignment as "submitted."
- "Ask a Doubt" Tab: A dedicated section where students can submit queries. Queries will be routed to a teacher's inbox for a direct response.
- Notices and Events: A centralized page will list all official notices and events. Both students and teachers can post new events, while only designated users (like a Teacher with Admin privileges or a Club Head) can post official notices.

---

2. Technology Stack 🛠️

The tech stack is designed for a team of beginners, prioritizing ease of use, maintainability, and scalability.

- Frontend: Next.js with JavaScript and Tailwind CSS. Next.js offers a great framework for building a fast, modern web application and can handle server-side rendering for better performance. Tailwind CSS simplifies styling and ensures a consistent design.

- Backend: Node.js with Express.js. This is a great choice for building a RESTful API, especially for a JavaScript-centric team.
- Database: PostgreSQL hosted on Supabase. Supabase provides a managed PostgreSQL database, eliminating the complexity of setting up and managing a server. It also offers a seamless integration with its storage solution.
- Authentication: Clerk. Clerk handles all user authentication, which is a major benefit for a beginner team as it manages user sessions, password hashing, and login/sign-up components out of the box.
- Cloud Storage: Supabase Storage. This is an easy-to-use alternative to AWS S3 and integrates directly with your Supabase database.
- Deployment: Vercel for the frontend and Render for the backend API. Both platforms are known for their ease of use and Git-based deployment workflows.
- AI: Gemini API or OpenAI API. Used for features like AI-powered post suggestions.

---

3. API Routes and Functionality 📡

The backend API will be the backbone of the application, handling all data operations. Here are some of the key routes:

Authentication & Profile

- `POST /api/auth/register`: Creates a new user in the database after successful registration with Clerk, storing their role, course, and major.
- `GET /api/users/:userId`: Fetches a user's complete profile details, including their bio, post history, and follower/following counts.
- `PUT /api/users/:userId/profile`: Updates a user's profile information.

Social & Communication

- `GET /api/feed`: Retrieves the main feed with posts from all users.
- `POST /api/posts`: Creates a new post with associated media.
- `POST /api/posts/:postId/like`: Adds an emoji reaction to a post.
- `GET /api/posts/:postId/comments`: Fetches comments for a specific post.
- `POST /api/posts/:postId/comment`: Adds a new comment.
- `POST /api/follow/:userId`: Initiates a follow request.
- `POST /api/unfollow/:userId`: Removes a follower.
- `POST /api/chat/start`: Creates a new chat.
- `POST /api/chat/:chatId/message`: Sends a message in a chat.

Academic and Admin

- `GET /api/courses`: Fetches a list of courses and majors for the assignment creation dropdown.
- `POST /api/assignments`: (Teacher) Creates a new assignment for a specific course and major.
- `GET /api/assignments/student`: (Student) Retrieves assignments for the current user's registered course and major.
- `POST /api/assignments/:assignmentId/submit`: (Student) Submits an assignment.
- `GET /api/assignments/:assignmentId/submissions`: (Teacher) Views all submissions for an assignment.
- `POST /api/queries`: (Student) Submits a query to a specific teacher.
- `GET /api/queries/teacher`: (Teacher) Fetches queries directed to the current teacher.
- `POST /api/notices`: (Teacher/Admin) Creates a new notice or event.
- `GET /api/notices`: Retrieves a list of all notices and events.

---

4. Non-Functional Requirements ⚙️

These requirements define the quality and performance of the application.

- Scalability: The application must be scalable to accommodate a growing user base. Using a separate backend and a managed database will support this.
- Security: User data, especially passwords, will be encrypted. Access to API routes will be restricted based on the user's role.
- Usability: The user interface must be intuitive and easy for both students and teachers to navigate.
- Reliability: The application should be reliable, with minimal downtime. Using platforms like Vercel and Render helps ensure high uptime.
- Performance: The app should be fast and responsive, with quick loading times for the feed and other pages. This is achieved by using a modern framework like Next.js and optimizing API calls.

Frontend File Structure

A well-organized file structure is key for a growing application, especially with a beginner team. The `src` directory pattern is a best practice for separating source code from configuration files.

```
/my-college-app-frontend
|-- /src
|  |-- /app                     // Next.js App Router for routing
|  |  |-- /_components           // Private folder for components not directly routed
|  |  |  |-- /ui                // Reusable UI components (e.g., Button, Input)
|  |  |  |-- /shared            // Components shared across the app (e.g., Navbar)
|  |  |  |-- /auth              // Auth-related components (e.g., CustomSignUpForm)
|  |  |  |-- /dashboard         // Components for the teacher/student dashboards
|  |  |  |-- /posts             // Components for posts and comments
|  |  |  |-- /messaging         // Components for the chat UI
|  |  |-- /api                  // Next.js API Routes (if you want to proxy requests)
|  |  |-- /onboarding           // Route for role selection post-signup
|  |  |  |-- page.jsx
|  |  |-- /notices              // Route for the notices and events page
|  |  |  |-- page.jsx
|  |  |-- /assignments
|  |  |  |-- page.jsx           // Student assignment list page
|  |  |-- /teacher
|  |  |  |-- /assignments
|  |  |  |  |-- page.jsx        // Teacher assignment management page
|  |  |-- /profile
|  |  |  |-- [userId]           // Dynamic route for user profiles
|  |  |  |  |-- page.jsx
|  |  |-- /feed
|  |  |  |-- page.jsx
|  |  |-- /chat
|  |  |  |-- [chatId]           // Dynamic route for a specific chat
|  |  |  |  |-- page.jsx
|  |  |-- layout.jsx            // The root layout for the entire app
|  |  |-- page.jsx              // The main homepage/login page
|  |-- /lib                     // Utility functions and client-side data fetching
|  |  |-- api.js                // Functions to call your backend API
|  |  |-- useAuth.js            // Custom hook for authentication state
```

|-- .env.local
|-- next.config.js
|-- package.json

API Route Connections and Page Details

The Next.js frontend will use the `fetch` API to communicate with your Express.js backend. The data fetching can be done on the server-side in Next.js Server Components or on the client-side using a hook like `useState` with `useEffect`. For your application, it's best to use a mix of both.

- **`app/sign-up/page.jsx`**: This is your **custom sign-up form**.
  - **Connection**: This page will not connect to a backend API directly for authentication. Instead, it will use Clerk's `useSignUp` hook to handle the user creation in Clerk's system.
  - **Logic**: After a successful sign-up with Clerk, the page will redirect the user to the `/onboarding` page to select their role and provide academic details.
- **`app/onboarding/page.jsx`**: This is the **post-sign-up form**.
  - **Connection**: This page will call your backend's `POST /api/auth/register` route.
  - **Logic**: It will send the user's Clerk `userId` along with their chosen role (`Student` or `Teacher`), `course`, and `major` to your backend. The backend will then store this information in the appropriate database tables.
- **`app/feed/page.jsx`**: This is the main **college feed**.
  - **Connection**: This page will call `GET /api/posts` to fetch all posts and display them. It will also connect to `POST /api/posts/:postId/like` and `POST /api/posts/:postId/comment` when a user interacts with a post.
  - **Logic**: Use a client component to handle post interactions like likes and comments. The feed itself could be a server component to fetch the initial data quickly.
- **`app/profile/[userId]/page.jsx`**: A **dynamic page for user profiles**.
  - **Connection**: This page will call `GET /api/users/:userId` to fetch a specific user's profile details. It will also use `PUT /api/users/:userId/profile` for updating a user's bio or club head status.
  - **Logic**: The page will display the user's photo, bio, followers/following count, and a list of their recent posts. It should have a conditional `Edit Profile`

button that only appears if the logged-in user's ID matches the profile's `userId`.

- **`app/assignments/page.jsx`** **(Student View)**: The **student assignment dashboard**.
  - **Connection**: This page will call `GET /api/assignments/student/:course/:major` to retrieve assignments specific to the student's course and major.
  - **Logic**: It will display a list of assignments with details like due date and status. Each assignment will have a button to "Submit" or "Mark as Submitted," which triggers a call to `POST /api/assignments/:assignmentId/submit`.
- **`app/teacher/assignments/page.jsx`** **(Teacher View)**: The **teacher assignment dashboard**.
  - **Connection**: This page will call `POST /api/assignments` to create new assignments and `GET /api/assignments/:assignmentId/submissions` to view student submissions. It can also call `GET /api/courses` to populate the course and major dropdowns in the assignment creation form.
  - **Logic**: It will have a form for creating new assignments and a list of assignments they've already posted.
- **`app/notices/page.jsx`**: The **notices and events page**.
  - **Connection**: Calls `GET /api/notices` to fetch a list of all notices and events.
  - **Logic**: It will display a list of official announcements and events. It should also have a "Create Notice/Event" button that is only visible to users with the 'Teacher' role, which triggers a `POST /api/notices` call.

Other Details to Look For 🕵️

- **State Management**: For local state (like form inputs), use React's `useState` hook. For global, shared state (like the current user's data), pass the data as props from a parent layout component. For complex state, consider a state management library like Zustand, as it's a lightweight and beginner-friendly option.
- **Data Fetching**: Use a custom `api.js` file in the `/lib` directory. This centralizes all your API calls, making them easier to manage and modify. Each function will simply `fetch` data from your Express backend URL.
- **Error Handling**: Implement `try...catch` blocks for all API calls to gracefully handle network errors or bad responses from the backend.

- **UI/UX**: Use **Tailwind CSS** to build a responsive, clean interface that works well on both desktop and mobile devices. Focus on making the navigation intuitive and the academic sections easy to use.

Backend API Design: Node.js with Express

The backend will be a RESTful API built with Node.js and Express.js. It will serve as the central hub for all data and business logic. The API will handle user authentication, data management, and serve as the intermediary between the Next.js frontend and the PostgreSQL database.

API Routes

Here are the API routes that you will build, categorized by functionality.

1. Authentication & Profile Management

- POST /api/auth/register: Creates a new user account.
  - **Body**: { email, password, role, course, major }
- POST /api/auth/login: Authenticates a user and returns a token.
  - **Body**: { email, password }
- GET /api/users/:userId: Retrieves a specific user's public profile, including their bio, followers/following count, and recent posts.
- PUT /api/users/:userId/profile: Updates a user's profile information (e.g., bio, club head status).

2. Social Features (Posts & Interactions)

- POST /api/posts: Creates a new post with text and media.
  - **Body**: { userId, content, mediaUrl, tags }
- GET /api/posts: Retrieves a list of all posts for the main feed.
- POST /api/posts/:postId/like: Adds a like/reaction from a user to a post.
- POST /api/posts/:postId/comment: Adds a new comment to a post.
- POST /api/follow/:userId: Allows a user to follow another user.
- DELETE /api/unfollow/:userId: Allows a user to unfollow another user.

3. Academic Features

- GET /api/courses: Retrieves a list of all available courses and majors.
- POST /api/assignments: **(Teacher only)** Creates a new assignment for a specific course and major.
  - **Body**: { teacherId, course, major, title, details, dueDate }
- GET /api/assignments/student/:course/:major: **(Student only)** Retrieves a list of assignments for a student's specific course and major.

- POST /api/assignments/:assignmentId/submit: **(Student only)** Allows a student to submit an assignment.
    - **Body**: { studentId, submissionDetails }
- GET /api/assignments/:assignmentId/submissions: **(Teacher only)** Retrieves a list of submissions for a specific assignment.

4. Queries & Notices

- POST /api/queries: **(Student only)** Allows a student to ask a query.
    - **Body**: { studentId, teacherId, queryText }
- GET /api/queries/teacher/:teacherId: **(Teacher only)** Retrieves a list of queries directed to a specific teacher.
- POST /api/notices: **(Teacher/Admin only)** Creates a new notice.
    - **Body**: { authorId, title, content }
- GET /api/notices: Retrieves a list of all notices.

5. Direct Messaging (DM)

- POST /api/chat/start: Creates a new one-on-one or group chat.
- POST /api/chat/:chatId/message: Sends a new message in a chat.
- GET /api/chat/:chatId/messages: Retrieves the message history for a specific chat.
- GET /api/chats/:userId: Retrieves a list of all chats for a user.

Database Schema: PostgreSQL

The database schema is designed to be relational, ensuring data integrity and efficient querying.

1. users Table

This is the central table for all users.

- id (PK, UUID): Unique ID for each user.
- email (UNIQUE): User's email address.
- password (VARCHAR): Hashed password.
- role (ENUM): 'Student' or 'Teacher'.
- createdAt (TIMESTAMP)

2. profiles Table

Stores public profile information, linked to the users table.

- userId (FK, UUID): Links to users.id.
- username (VARCHAR): Display name.
- bio (TEXT): User's short biography.
- pfp_url (VARCHAR): URL of the profile picture.
- followingCount (INTEGER)
- followersCount (INTEGER)

3. students Table

Stores student-specific academic information.

- userId (PK, UUID): Links to users.id.
- course (VARCHAR): Student's registered course.
- major (VARCHAR): Student's major.

4. teachers Table

Stores teacher-specific academic information.

- userId (PK, UUID): Links to users.id.
- subjects (VARCHAR[]): An array of subjects taught.

5. posts Table

Stores all user-generated content for the feed.

- id (PK, UUID): Unique ID for the post.
- authorId (FK, UUID): Links to users.id.
- content (TEXT)
- mediaUrl (VARCHAR): URL to the image/video.
- createdAt (TIMESTAMP)

6. likes Table

Records user reactions to posts.

- id (PK, UUID)
- postId (FK, UUID): Links to posts.id.
- userId (FK, UUID): Links to users.id.
- reactionType (VARCHAR): 'like', 'love', etc.

7. assignments Table

Stores assignments created by teachers.

- id (PK, UUID)
- teacherId (FK, UUID): Links to users.id.
- course (VARCHAR): Target course.
- major (VARCHAR): Target major.
- title (VARCHAR)
- details (TEXT)
- dueDate (TIMESTAMP)

8. submissions Table

Stores student submissions for assignments.

- id (PK, UUID)
- assignmentId (FK, UUID): Links to assignments.id.
- studentId (FK, UUID): Links to users.id.
- submissionDetails (TEXT)
- submittedAt (TIMESTAMP)

9. queries Table

Stores student-teacher queries.

- id (PK, UUID)
- studentId (FK, UUID)
- teacherId (FK, UUID)
- queryText (TEXT)
- answered (BOOLEAN)

10. notices_events Table

Stores notices and events created by teachers/admins.

- id (PK, UUID)
- authorId (FK, UUID)
- type (ENUM): 'Notice' or 'Event'.
- title (VARCHAR)
- content (TEXT)