I need a detailed written plan for a webapp I am creating, for a computer club where members share and collaborate on apps they are developing, or want to develop. The app will facilitate the collaboration by sharing information on apps being developed and will have information on skills members have. Information on the projects will be stored in a mongo database in the webapp and members can post info on it. Write access will be limited to members who must authenticate as users. The app will have a javascript frontend and backend and will be an express node js app. When the coding is completed, it should be possible to setup as a github repository with the code, to be deployed.

Below is a detailed written plan for developing your web application for a computer club, designed to facilitate collaboration on app development by allowing members to share and manage project information, showcase their skills, and collaborate effectively. The plan covers the requirements, technical architecture, development phases, database schema, security considerations, and deployment strategy, ensuring the app is ready to be set up as a GitHub repository for deployment.

Web Application Plan: Computer Club Collaboration Platform

1. Project Overview

The web application, tentatively named "CodeCollab", will serve as a platform for a computer club where members can:

- Share details about apps they are developing or want to develop.
- List their skills to facilitate collaboration.
- Collaborate on projects by posting updates, sharing resources, and communicating.
- Store project and member data in a MongoDB database.
- Restrict write access to authenticated members.
- Be built using a JavaScript-based tech stack (Node.js, Express.js for the backend, and a JavaScript frontend framework).
- Be deployable via a GitHub repository.

The app will prioritize usability, security, and scalability, with a focus on enabling collaboration and transparency within the computer club.

---

2. Functional Requirements

2.1 Core Features

- User Authentication and Authorization:
    - Users must register and log in to access write functionalities (e.g., posting project updates, editing profiles).
    - Role-based access: Only authenticated members can create/edit/delete project data or update their skill profiles.
    - Non-members can view public project information (read-only access).
- Member Profiles:
    - Each member can create a profile with:
        - Name, contact info (optional), and a short bio.
        - List of skills (e.g., JavaScript, Python, UI/UX design, MongoDB).
        - Links to personal portfolios or GitHub profiles (optional).

- Members can edit their own profiles.
- Project Management:
  - Members can create, update, and delete projects they own.
  - Project details include:
    - Title, description, and status (e.g., ideation, in progress, completed).
    - Required skills for collaboration.
    - Links to related resources (e.g., GitHub repos, design mockups).
    - Optional tags for categorization (e.g., web app, mobile app, AI).
  - Members can comment on projects to provide feedback or express interest in collaborating.
- Collaboration Features:
  - Members can join projects by sending a request to the project owner.
  - Project owners can accept or reject collaboration requests.
  - A discussion board or comment section for each project to facilitate communication.
- Search and Discovery:
  - Search for projects by title, tags, or required skills.
  - Search for members by skills or name to find potential collaborators.
- Public vs. Private Access:
  - Public users (non-members) can view project summaries and member skills (read-only).
  - Authenticated members have full access to create, edit, and delete content.

2.2 Non-Functional Requirements

- Performance: The app should handle up to 500 concurrent users with minimal latency (target: <2 seconds for page loads).
- Scalability: The architecture should support scaling to accommodate growing membership and project data.
- Security: Protect user data with secure authentication, encrypted passwords, and role-based access control.
- Usability: Intuitive UI with responsive design for desktop and mobile devices.
- Maintainability: Codebase should be modular, well-documented, and version-controlled in a GitHub repository.

3. Technical Architecture

3.1 Tech Stack

- Frontend:
  - Framework: React.js (for a dynamic, component-based UI).
  - Styling: CSS with Tailwind CSS or Material-UI for rapid development and responsive design.
  - State Management: Redux or React Context API for managing user sessions and app state.
  - HTTP Client: Axios for API requests to the backend.

- Backend:
    - Framework: Node.js with Express.js for building a RESTful API.
    - Authentication: JSON Web Tokens (JWT) for secure user authentication.
    - File Uploads: Multer for handling resource uploads (e.g., project documents, images).
- Database:
    - MongoDB (NoSQL) for flexible, schema-less storage of user and project data.
    - MongoDB Atlas for cloud-hosted database management.
- Version Control:
    - Git with a GitHub repository for source code management.
- Deployment:
    - Hosting: Heroku, Vercel, or AWS for deploying the app.
    - CI/CD: GitHub Actions for automated testing and deployment.

3.2 System Architecture

- Client-Server Model:
    - The React frontend communicates with the Express backend via RESTful API endpoints.
    - MongoDB stores all persistent data (users, projects, comments).
- Authentication Flow:
    - Users register/login using email and password.
    - The backend issues a JWT upon successful login, stored in the client's local storage.
    - JWT is sent with API requests to authenticate write operations.
- File Storage:
    - Small files (e.g., project images, PDFs) are stored in MongoDB GridFS or a cloud storage service like AWS S3.
- Scalability:
    - Horizontal scaling of the Express server using a load balancer.
    - MongoDB sharding for large datasets.

4. Database Schema (MongoDB)

4.1 Collections

- Users:

javascript

```
{
  _id: ObjectId,
  email: String (unique, required),
  password: String (hashed, required),
  name: String (required),
  bio: String (optional),
  skills: [String] (e.g., ["JavaScript", "Python", "UI/UX"]),
  portfolioLinks: [{ name: String, url: String }] (optional),
  createdAt: Date,
```

updatedAt: Date
}

- Projects:

javascript
{
_id: ObjectId,
title: String (required),
description: String (required),
status: String (enum: ["ideation", "in_progress", "completed"]),
requiredSkills: [String],
tags: [String],
owner: ObjectId (ref: Users),
collaborators: [{ userId: ObjectId (ref: Users), status: String (enum: ["pending", "accepted"]) }],
resources: [{ name: String, url: String, fileId: ObjectId (optional, ref: GridFS)]],
createdAt: Date,
updatedAt: Date
}

- Comments:

javascript
{
_id: ObjectId,
projectId: ObjectId (ref: Projects),
userId: ObjectId (ref: Users),
content: String (required),
createdAt: Date,
updatedAt: Date
}

4.2 Indexes

- Create indexes on Users.email (unique) for fast login queries.
- Index Projects.title and Projects.tags for efficient search.
- Index Comments.projectId for quick retrieval of project comments.

5. API Endpoints

5.1 Authentication

- POST /api/auth/register: Register a new user (email, password, name).
- POST /api/auth/login: Authenticate user and return JWT.
- GET /api/auth/me: Get authenticated user's profile (requires JWT).

5.2 Users

- GET /api/users: List all users (public, read-only).
- GET /api/users/:id: Get user details by ID (public).
- PUT /api/users/:id: Update user profile (requires JWT, owner only).
- GET /api/users/search?query=skill: Search users by skills or name.

5.3 Projects

- GET /api/projects: List all projects (public).
- GET /api/projects/:id: Get project details, including comments (public).
- POST /api/projects: Create a new project (requires JWT).
- PUT /api/projects/:id: Update project details (requires JWT, owner only).
- DELETE /api/projects/:id: Delete a project (requires JWT, owner only).
- GET /api/projects/search?query=tag: Search projects by title or tags.

5.4 Collaboration

- POST /api/projects/:id/collaborate: Request to join a project (requires JWT).
- PUT /api/projects/:id/collaborate/:userId: Accept/reject collaboration request (requires JWT, owner only).

5.5 Comments

- GET /api/projects/:id/comments: Get all comments for a project.
- POST /api/projects/:id/comments: Add a comment to a project (requires JWT).
- DELETE /api/comments/:id: Delete a comment (requires JWT, owner only).

6. Development Phases

Phase 1: Project Setup (1-2 weeks)

- Set up GitHub repository with initial project structure:
- /codecollab
- ├── /client (React frontend)
- │ ├── /src
- │ │ ├── /components
- │ │ ├── /pages
- │ │ ├── /styles
- │ │ └── /utils
- ├── /server (Express backend)
- │ ├── /controllers
- │ ├── /models
- │ ├── /routes
- │ ├── /middleware
- │ └── /config
- ├── /docs (API documentation, setup instructions)
- ├── README.md
- ├── package.json

└── .gitignore

- Install dependencies (React, Express, MongoDB driver, JWT, Multer, etc.).
- Configure MongoDB Atlas and environment variables (.env for API keys, database URI).
- Set up ESLint and Prettier for code consistency.

Phase 2: Backend Development (3-4 weeks)

- Implement MongoDB schema and models using Mongoose.
- Build RESTful API endpoints with Express.js.
- Implement JWT authentication middleware.

- Add input validation (e.g., Joi or Express Validator).
- Write unit tests for API endpoints using Jest.

Phase 3: Frontend Development (3-4 weeks)

- Create React components for:
    - User authentication (login/register forms).
    - User profile page (view/edit skills, bio).
    - Project listing and detail pages.
    - Search functionality for users and projects.
    - Collaboration request and comment forms.
- Implement responsive design with Tailwind CSS or Material-UI.
- Integrate Axios for API calls and Redux/Context for state management.
- Write unit tests for components using React Testing Library.

Phase 4: Integration and Testing (2-3 weeks)

- Connect frontend and backend via API calls.
- Test authentication and authorization flows.
- Perform end-to-end testing with tools like Cypress.
- Conduct usability testing with sample users (club members).
- Fix bugs and optimize performance.

Phase 5: Deployment and Documentation (1-2 weeks)

- Deploy backend to Heroku/Vercel/AWS and frontend to Vercel or Netlify.
- Set up MongoDB Atlas for production.
- Configure GitHub Actions for CI/CD.
- Write comprehensive README.md with:
    - Project overview.
    - Setup instructions (local and production).
    - API documentation (e.g., using Swagger or Postman).
    - Contribution guidelines for club members.
- Create API documentation in /docs.

7. Security Considerations

- Authentication: Use bcrypt for password hashing and JWT for session management.
- Authorization: Implement middleware to restrict write access to authenticated users and project owners.
- Data Validation: Sanitize all user inputs to prevent injection attacks (e.g., MongoDB query injection).
- CORS: Restrict API access to the frontend domain.
- HTTPS: Enforce HTTPS in production for secure communication.
- Rate Limiting: Use Express Rate Limit to prevent abuse of API endpoints.
- File Uploads: Validate file types and sizes for uploaded resources to prevent malicious uploads.

8. Deployment Strategy

- GitHub Repository:

- Create a public or private repository on GitHub (e.g., github.com/computer-club/codecollab).
- Use branches (main for production, develop for ongoing work, feature branches for specific tasks).
- Include a .gitignore to exclude sensitive files (e.g., .env, node_modules).

- CI/CD:
  - Use GitHub Actions to run tests and deploy to Heroku/Vercel/AWS on push to main.
  - Example workflow: Lint, test, build, and deploy.
- Production Setup:
  - Backend: Deploy on Heroku with environment variables for MongoDB URI, JWT secret, etc.
  - Frontend: Deploy on Vercel with automatic scaling.
  - Database: Use MongoDB Atlas with backups enabled.
- Monitoring:
  - Use tools like New Relic or AWS CloudWatch for performance monitoring.
  - Set up error tracking with Sentry.

9. Future Enhancements
- Add real-time collaboration features using WebSockets (e.g., Socket.IO for live chat).
- Implement notifications for collaboration requests and comments.
- Add support for project milestones and task tracking.
- Integrate with GitHub API to automatically pull project updates from repositories.
- Add an admin role for club moderators to manage content.

10. Timeline and Milestones
- Total Duration: ~10-15 weeks (depending on team size and expertise).
- Milestones:
  - Week 1-2: Project setup and repository initialization.
  - Week 3-6: Backend development and API implementation.
  - Week 7-10: Frontend development and UI/UX design.
  - Week 11-13: Integration, testing, and bug fixing.
  - Week 14-15: Deployment, documentation, and final testing.

11. Deliverables
- A fully functional web application deployed to a hosting platform.
- A GitHub repository containing:
  - Source code for frontend and backend.
  - Comprehensive README with setup and deployment instructions.
  - API documentation in /docs.
  - Automated CI/CD pipeline configuration.
- MongoDB Atlas database with sample data for testing.
- User guide for club members to use the platform.

12. Assumptions and Risks

- Assumptions:
  - Club members have basic technical knowledge to use the platform.
  - MongoDB Atlas free tier is sufficient for initial deployment.
  - Team has experience with JavaScript, React, and Node.js.
- Risks:
  - Scope creep if additional features are requested mid-development.
  - Security vulnerabilities if input validation or authentication is not robust.
  - Performance issues if database queries are not optimized.
  - Mitigation: Follow agile development with regular feedback from club members, prioritize security, and optimize queries with indexing.

This plan provides a comprehensive roadmap for developing the CodeCollab web application. The modular architecture, clear division of tasks, and focus on security and scalability ensure the app meets the computer club's needs. The GitHub repository will be structured for easy collaboration and deployment, enabling club members to contribute and maintain the codebase. If you need specific code snippets, wireframes, or further details on any section, let me know!