SOFTWARE ENGINEER ASSIGNMENT

ASSIGNMENT - 1

Problem Statement

Build a **real-time, full-stack web application** for managing tasks and projects. The app should allow multiple users to collaborate in real-time, manage tasks, and see updates instantly.

Core Features

1. Task Management

- Users can **create, edit, and delete tasks**.
- Tasks should include:
    - Title, Description, Status (To-Do, In Progress, Done), Priority, and Due Date.
- Each task belongs to a **project**.

---

2. Real-Time Collaboration

- Multiple users can collaborate on a project.
- When one user adds/edits a task, all other users see updates **instantly** (real-time).
- Use **WebSockets** (e.g., Socket.IO) to enable this functionality.

---

3. User Authentication

- Implement **JWT-based authentication**.
- Users can:
    - **Sign up** and **log in** securely.
    - Invite others to collaborate on their projects using email or a unique project link.

---

4. Project Dashboard

- Each user can view a dashboard with:
    - All their projects.
    - Tasks grouped by status (To-Do, In Progress, Done) with a Kanban-style layout.

---

5. Notifications

- Users receive **real-time notifications** when:
    - A task is created, edited, or deleted.
    - They are invited to collaborate on a project.

Bonus Features *(Optional)*

1. **Search & Filters**: Users can search for tasks by title, description, or priority.
2. **Due Date Reminders**: Send reminders when tasks are nearing their due dates.
3. **Role-Based Access Control**: Only task creators or project owners can edit/delete tasks.
4. **Activity Logs**: Maintain logs of all changes made to tasks (who changed what and when).
5. **Responsive UI**: Design the app to look great on mobile and desktop.

ASSIGNMENT - 2

Problem Statement

Build a **web application** that allows users to upload documents (PDFs or text files) and ask questions about the content. The system will use **Generative AI** to provide answers by reading and understanding the document.

Requirements

1. Core Features

1. **Document Upload**:
   - Users can upload documents (PDF or text files).
   - Extract the text content from the uploaded document.
2. **Question-Answering**:
   - Users can ask questions about the uploaded document.
   - Use a **Generative AI model** (e.g., GPT-4 via OpenAI API) to generate accurate answers based on the document content.
3. **Web UI**:
   - A clean and interactive frontend that allows:
     - Uploading a document.
     - Asking questions.
     - Displaying answers from the AI.
4. **Response Accuracy**:
   - Implement **prompt engineering** to ensure the AI answers questions based on the uploaded document content.

---

2. Bonus Features *(Optional)*

1. **Context Display**: Show the part of the document from which the AI-generated the answer.
2. **Save History**: Allow users to save previous Q&A sessions.
3. **Multi-Language Support**: Allow questions and answers in multiple languages.

Tech Stack

1. **Frontend**: React.js with a clean, responsive design (Tailwind CSS or Material UI).
2. **Backend**: Node.js + Express for API endpoints.
3. **Generative AI**: OpenAI GPT API (e.g., GPT-4).
4. **File Parsing**: Use libraries like `pdf-parse` or `Mammoth` to extract text from uploaded files.
5. **Storage**: Store documents and Q&A logs in a simple database like MongoDB.

Deliverables

1. A working web application (code repository with clear instructions to run locally).
2. Screenshots or a short demo video showcasing the functionality.
3. API documentation