■ Project Assignment: CollabHub – Team Collaboration Platform

■ Project Overview

CollabHub is a full-stack web application for team collaboration and project management. The platform will allow users to manage projects, assign tasks, communicate in real time, and share files. The goal of this project is to practice end-to-end development using Next.js (Frontend) and NestJS (Backend) with modern development practices.

■ Objectives

- Build a scalable backend with NestJS that exposes REST and GraphQL APIs.
- Develop a responsive frontend using Next.js with authentication, dashboards, and task boards.
- Implement real-time features (chat, notifications) using WebSockets.
- Handle file uploads and project-specific documents.
- Ensure secure authentication with JWT & OAuth (Google Sign-In).

■ Core Features

1. Authentication & Authorization

- Sign up / Login with email & password.
- Google login (OAuth).
- JWT authentication with refresh tokens.
- Role-based access: Admin, Project Manager, Team Member.

2. Project & Task Management

- Create, edit, and delete projects.
- Add team members to projects.
- Task board (Kanban style) with drag & drop between To-Do, In Progress, Done.
- Tasks should include: title, description, assignee, due date, priority.

3. Real-Time Chat

- Each project has a chatroom.
- WebSockets for instant message delivery.
- Show online/offline status.
- Typing indicators and read receipts.

4. File Sharing

- Upload documents, images, or PDFs within a project.

- Store files in AWS S3 (or local storage for dev).
- Preview images and PDFs in the frontend.

5. Dashboard & Analytics

- Project overview (active projects, members, tasks completed).
- Progress charts (task completion by status).
- Notifications for task assignment, deadlines, and chat messages.

■■ Tech Stack

Frontend (Next.js)

- Next.js 14 (App Router, SSR + CSR).
- TailwindCSS (UI styling).
- React Query (API data fetching & caching).
- NextAuth.js (authentication).
- Zustand or Redux Toolkit (state management).

Backend (NestJS)

- NestJS with modular architecture.
- PostgreSQL + TypeORM (database).
- Passport.js strategies (JWT + OAuth).
- WebSockets for real-time chat/notifications.
- Swagger for API documentation.

Others

- AWS S3 (file storage).
- Redis (caching + WebSocket scaling).
- Docker for containerization.

■ Database Schema (Core Tables)

• Users: id, name, email, passwordHash, role, avatar

• Projects: id, name, description, createdBy, members[]

• Tasks: id, title, description, status, priority, dueDate, assigneeId, projectId

• Messages: id, content, senderId, projectId, createdAt

• Files: id, fileName, url, uploadedBy, projectId

■ Deliverables

Phase 1: Backend (NestJS)

- Setup NestJS project with modules: Auth, Users, Projects, Tasks, Chat, Files.
- Implement authentication (JWT + OAuth).
- CRUD APIs for projects & tasks.
- WebSocket events for chat.
- File upload API (AWS/local).
- Swagger API documentation.

Phase 2: Frontend (Next.js)

- Setup Next.js project with Tailwind.
- Authentication (NextAuth with JWT & Google).
- Project list & details page.
- Task board (drag & drop).
- Real-time chat UI (WebSocket integration).
- File upload UI with preview.
- Dashboard with charts.

■ Evaluation Criteria

- Code quality (clean, modular, documented).
- Proper use of Next.js & NestJS features.
- Secure authentication & authorization.
- Working real-time chat and notifications.
- User-friendly, responsive UI.
- Proper database design with relationships.