Project Overview: Collaborative Task Management App

The trainees will build a **Task Management App** where users can:

1. Create an account.
2. Join or create teams.
3. Add tasks to team projects.
4. Assign tasks to specific team members.
5. Set due dates, track status (in-progress, completed), and leave comments on tasks.
6. Filter tasks by project or team member, and manage notifications for deadlines.

The app will have **user authentication** and **authorization** to control access based on user roles (e.g., admin, team member). **Multiple tables** in Xata will include **Users**, **Teams**, **Projects**, **Tasks**, and **Comments**.

Core Features:

1. **User Authentication and Authorization**:
    - **Signup/Login** with JWT authentication.
    - Password hashing (e.g., using bcrypt).
    - Different roles (Admin, Team Member).
    - Protected routes for team-specific and admin-specific tasks.
2. **Teams and Projects**:
    - Users can create or join teams.
    - Teams have multiple projects.
    - Each project can contain multiple tasks assigned to team members.
3. **Task Management**:
    - Create, update, and delete tasks.
    - Assign tasks to team members.
    - Set due dates and status (in-progress, completed).
    - Add comments to tasks for communication.
4. **Notification System**:
    - Notify users of upcoming task deadlines or project updates.

5. **Testing**:
    - Unit tests (for API endpoints and services).
    - Integration tests (ensure all components work together).
    - Mock Xata responses in tests to simulate database interactions.

Project Structure:

Backend (Node/Express + TypeScript + Xata)

- **Tables/Models**:
    - **User**: { id, name, email, password, role }
    - **Team**: { id, name, description, adminId (foreign key to User) }
    - **Project**: { id, name, teamId (foreign key to Team) }
    - **Task**: { id, description, status, dueDate, projectId (foreign key to Project), assignedToId (foreign key to User) }
    - **Comment**: { id, content, taskId (foreign key to Task), userId (foreign key to User) }
- **Routes**:
    - **/auth/register**: Register a user (Admin or Team Member).
    - **/auth/login**: Login with JWT generation.
    - **/teams**: Create, update, join, leave a team.
    - **/projects**: CRUD operations for projects within a team.
    - **/tasks**: CRUD for tasks, task assignment, status updates.
    - **/comments**: Add comments on tasks.
- **Controllers**:
    - **AuthController**: Handle registration, login, and JWT authentication.
    - **TeamController**: Handle team management (create, join, leave teams).
    - **ProjectController**: Handle CRUD operations for projects.
    - **TaskController**: Task CRUD, assignment, filtering by status, and member.
    - **CommentController**: Handle adding/viewing comments.
- **Middleware**:
    - JWT Authentication middleware for protected routes.

- Role-based access control (e.g., Admin can manage teams).

Frontend (React/Next.js + TypeScript)

- **Pages/Components**:
  - **Login/Signup** pages with form validation.
  - **Dashboard** (Shows list of teams and projects for the logged-in user).
  - **Task Board**: Display tasks in a kanban-style board (drag and drop).
  - **Task Details**: View and edit tasks, add comments.
  - **Team Management**: Manage teams (create team, invite members).
- **API Calls** (from the front-end):
  - **useEffect** to fetch the list of teams, projects, tasks.
  - **Axios** or **Fetch** for API requests (protected routes should include JWT in headers).
  - Update state on task creation, update, and deletion.

Database (Xata)

- **Table Relationships**:
  - **User - Team**: Many-to-many (a user can be in many teams, a team can have many users).
  - **Team - Project**: One-to-many (a team can have multiple projects).
  - **Project - Task**: One-to-many (a project can have multiple tasks).
  - **Task - User**: One-to-one (a task is assigned to a single user).
  - **Task - Comment**: One-to-many (a task can have many comments).
- **Xata Queries**:
  - Create relational records (e.g., a task assigned to a user).
  - Fetch tasks for a team and filter by project or user.
  - Fetch comments for a task.

Testing:

- **Backend Tests**:
  - **Jest** for unit tests (e.g., test authentication, task creation).
  - **Supertest** for API testing (test protected routes, verify token middleware).

    - Mock the Xata API with testing utilities for database operations.
- **Frontend Tests**:
    - **React Testing Library** for component testing (e.g., forms, task board).
    - **Jest** for unit testing logic (e.g., task filtering, state management).
    - Integration tests for end-to-end user flows (e.g., logging in, creating tasks).

Project Milestones:

Week 1: Database Schema & Basic Backend API

- Set up the Xata database with multiple tables and relationships.
- Create user registration and login functionality.
- Build basic API routes for teams and projects.

Week 2: Frontend Setup & Task Management

- Set up the frontend with React or Next.js.
- Build forms for task creation and the dashboard for displaying teams/projects.
- Implement task CRUD functionality and task assignment.

Week 3: Comments & Notifications

- Add comments to tasks and build the notification system (for task deadlines).
- Display notifications in the UI and send reminders.

Week 4: Testing & Final Touches

- Write unit and integration tests for both frontend and backend.
- Ensure the project is fully functional and tested.
- Final project presentation.

Learning Objectives:

- **TypeScript**: Understanding types, interfaces, and type-checking for both frontend and backend.
- **Node/Express**: Handling routes, middleware, and authorization with JWT.
- **Xata**: Managing multi-table relationships and performing queries on related data.
- **Testing**: Writing and executing unit, integration, and end-to-end tests.

- **Team Collaboration**: Use Git for version control, code reviews, and resolving merge conflicts.

Bonus Features (for Extra Credit):

- Implement a **task filtering system** by priority, assignee, or due date.
- Add a **real-time feature** using websockets (e.g., real-time task updates).