Task Management System: Software Engineering Assessment

Objective: Build a complete **Task Management System** that allows users to register, log in, and perform full management (create, view, edit, delete) of their personal tasks.

Choose Your Track: Full-Stack or Mobile Focus

To allow you to demonstrate your best skills, you have the option to pick **one** of the two tracks below. Both tracks include working with the required Node.js backend API.

Track	Focus	Requirements
Track A: Full-Stack Engineer	Backend & Web Frontend	Complete the **Backend API (Node.js)** and the **Web Frontend (Next.js)** sections.
Track B: Mobile Engineer	Backend & Mobile App	Complete the **Backend API (Node.js)** and the **Mobile App (Flutter)** sections.

Part 1: Mandatory Backend API (Node.js + TypeScript)

(Must be completed for both tracks)

The API must use **Node.js**, **TypeScript**, and an **SQL Database** (with Prisma/TypeORM) to serve data securely.

1. User Security (Authentication)

- Implement **Login, Registration, and Logout**.
- Use **JWT (JSON Web Tokens)** for security:
  - **Access Token** (short-lived) for accessing protected routes.
  - **Refresh Token** (long-lived) to securely get a new Access Token when the old one expires.
- **Hash passwords** using `bcrypt` before storing them.
  - *Required Endpoints:* `/auth/register`, `/auth/login`, `/auth/refresh`, `/auth/logout`.

2. Task Management (CRUD)

- Implement all required operations for tasks. Tasks must belong to the logged-in user.
- The main task list endpoint (`GET /tasks`) must include **pagination** (loading tasks in batches), **filtering** (by status), and **searching** (by title).
  - *Required Endpoints:* `/tasks` (GET/POST), `/tasks/:id` (GET/PATCH/DELETE), `/tasks/:id/toggle`.

3. Technical Requirements

- Use **TypeScript** throughout.
- Use an **ORM** (Object-Relational Mapper) like **Prisma** or **TypeORM**.
- Implement proper **validation** and clear **error handling** with standard HTTP status codes (e.g., 400, 401, 404).

Track A: Web Frontend (Next.js + TypeScript)

(Choose this track to focus on web development)

Build a responsive web application using **Next.js (App Router)** and **TypeScript**.

1. Authentication

- Create **Login and Registration pages** that connect to the backend API.
- Handle the logic for storing the Access Token and using the Refresh Token to stay logged in.

2. Task Dashboard

- Display the list of tasks retrieved from the backend.
- Implement **filtering** and **searching** features.
- Design must be **responsive** (works well on both desktop and mobile screens).

3. CRUD Functionality

- Implement UI and forms to **Add, Edit, Delete, and Toggle** the status of tasks.
- Show simple **notifications** (toasts) for successful operations.

Track B: Mobile App (Flutter)

(Choose this track to focus on cross-platform mobile development)

Build a mobile application using **Flutter** for both Android and iOS (focusing on Android for the final APK deliverable).

1. Authentication

- Create **Login and Registration screens**.
- Securely store the Access Token and Refresh Token using a tool like `flutter_secure_storage`.
- Implement logic to **automatically refresh the token** when it expires before retrying an API call.

2. Task Dashboard

- Display the task list using **efficient rendering** (`ListView.builder`).
- Include **pull-to-refresh** functionality to easily update the list.

3. Architecture & State Management

- Structure your code into clear layers: **UI, State, Repositories, and Services/API layer.**
- Use a modern state management solution like **Riverpod, Bloc/Cubit, or Provider.**

4. CRUD & Error Handling

- Implement all **Add, Edit, Delete, and Toggle** task functionalities.
- Display friendly **Snackbars or Dialogs** for common errors (like 401 Unauthorized or 500 Server Error).