Requirements Document: ToDo List Application

Overview

The ToDo List application should allow users to manage their tasks efficiently. It should include features to create, update, view, and delete tasks, along with categorization, due dates, and priority settings. The backend should be developed in **Go**, and **PostgreSQL** should be used as the database.

Goals

- Provide an intuitive backend API to handle task management operations.
- Create a design that supports future feature enhancements like task sharing and notifications (**Note: No need to develop any of the future features now**)

Functional Requirements

User Management

- **Sign Up**: Users should be able to create an account with an email and password.
- **Login**: Authenticate users via email and password.
- **Profile Management**: Users may update their profile details if required.

Task Management

- **Create Task**:
    - Fields include:
        - Title (required)
        - Description (optional)
        - Due Date (optional)
        - Priority (Low, Medium, High; default is Medium)
        - Category (optional, e.g., Work, Personal)
        - Status (default: Pending)
- **View Tasks**:
    - Provide options to filter tasks by:
        - Due Date
        - Priority
        - Category
        - Status
- **Update Task**: Allow modification of any task attribute.
- **Delete Task**: Option to mark tasks as deleted or remove them permanently.

Non-Functional Requirements

Performance

- Aim for an API response time of <500ms for common requests.

Security

- Use hashed passwords (e.g., bcrypt).
- Ensure secure data transmission using HTTPS.
- Implement authentication (e.g., JWT) to protect APIs.

Scalability

- Design the database schema and APIs to support future features like task sharing or collaborative lists. (**Note: No need to develop any of the future features now**)

Usability

- Provide clear and detailed API documentation (e.g., using Swagger).
- Ensure APIs follow RESTful principles.

Technical Requirements

Backend

- Language: Go (version >= 1.22).
- Framework: Use a lightweight framework such as Fiber or Gin.
- Key Libraries:
  - Database: `pgx` or `gorm` for PostgreSQL interaction.
  - Authentication: `jwt-go` for token management.
  - Validation: `validator` for input validation.

Database

- PostgreSQL Schema:
  - **Users Table**:
    - `id`: Primary Key
    - `email`: Unique, Not Null
    - `password_hash`: Not Null
    - `created_at`: Timestamp
  - **Tasks Table**:
    - `id`: Primary Key
    - `user_id`: Foreign Key (Users Table)
    - `title`: Not Null
    - `description`: Text
    - `due_date`: Timestamp
    - `priority`: Enum (Low, Medium, High)
    - `category`: Text
    - `status`: Enum (Pending, Completed)
    - `created_at`: Timestamp

        - `updated_at`: Timestamp

API Endpoints

Method	Endpoint	Description
POST	/signup	Create a new user
POST	/login	Authenticate a user
GET	/tasks	Get all tasks for a user
POST	/tasks	Create a new task
PUT	/tasks/:id	Update a task
DELETE	/tasks/:id	Delete a task

Instructions for the Candidate

As part of this task, feel free to reach out for any doubts or clarifications on the approach for designing and implementing this application.