Test Práctico Completo

Este test está diseñado evaluar habilidades en frontend (React - No relevantes), backend (Node.js con Express), y base de datos (PostgreSQL).

Objetivo

Desarrollar un sistema básico de gestión de tareas que permita a los usuarios:

- Autenticarse en la aplicación.
- Crear, leer, actualizar y eliminar tareas.
- Visualizar una lista de tareas en el frontend con filtros y paginación.

El sistema debe incluir:

- Un backend en Node.js con Express y conexión a PostgreSQL.
- Un frontend en React que consuma la API proporcionada por el backend.

Frontend (React)

1. **Pantallas:**
   - **Login:**
     - Formulario para ingresar username y password.
     - Al autenticarse, almacenar el token JWT en el almacenamiento local (localStorage).
   - **Lista de Tareas:**
     - Visualizar las tareas del usuario autenticado.
     - Filtros por estado (completadas, pendientes).
     - Paginación.
     - Botón para marcar una tarea como completada o eliminarla.
   - **Crear Tarea:**
     - Formulario para agregar una nueva tarea.
2. **Integración con el Backend:**
   - Usar fetch o axios para consumir los endpoints del backend.
   - Manejar correctamente los errores (ej., token inválido, validaciones, etc.).
3. **Estilo (no es relevante):**
   - Usar cualquier librería de UI (opcional) o CSS básico.
   - El diseño debe ser limpio y funcional.

Requisitos

Backend (Node.js con Express)

1. Endpoints:
   - POST /api/auth/login: Permite a un usuario autenticarse (no se requiere registro, usar credenciales predefinidas en el código).
     - Entrada: { username: "admin", password: "12345" }
     - Respuesta: { token: "jwt-token" }
   - GET /api/tasks: Retorna la lista de tareas con filtros y paginación.
     - Parámetros opcionales:
       - completed (booleano): Filtra tareas completadas/incompletas.
       - page (número): Página actual.
       - limit (número): Cantidad de tareas por página.
     - Respuesta: { tasks: [...], total: númeroTotalDeTareas }
   - POST /api/tasks: Crea una nueva tarea.
     - Entrada: { title: "Nueva tarea" }
   - PATCH /api/tasks/:id: Actualiza el estado completed de una tarea.
     - Entrada: { completed: true }
   - DELETE /api/tasks/:id: Elimina una tarea por su ID.
2. Base de Datos (PostgreSQL):
   - Configura una base de datos llamada task_management.
   - Tabla users:
     - id (PRIMARY KEY, SERIAL)
     - username (VARCHAR, UNIQUE)
     - password (VARCHAR)
   - Tabla tasks:
     - id (PRIMARY KEY, SERIAL)
     - title (VARCHAR, NOT NULL)
     - completed (BOOLEAN, DEFAULT FALSE)
     - user_id (FOREIGN KEY) (relación con users).
     - created_at (TIMESTAMP, DEFAULT NOW())
3. Autenticación:
   - Implementar autenticación con JWT.
   - Asegurar que los endpoints /api/tasks solo estén disponibles para usuarios autenticados.
4. Configuración:
   - Archivo .env para variables de entorno (puerto, credenciales de base de datos, clave JWT).
   - Script de migraciones para crear las tablas.

Requerimientos Técnicos

Backend

- Node.js
- Express.js
- PostgreSQL
- JWT para autenticación
- Librerías sugeridas: pg, jsonwebtoken, dotenv

Frontend

- React
- React Router para manejar las rutas
- Context API o Redux para manejar el estado global (opcional)
- Librerías sugeridas: axios para API calls

Criterios de Evaluación

1. **Funcionalidad:**
   - Todas las características requeridas están implementadas y funcionan correctamente.
2. **Calidad del Código:**
   - Código limpio, organizado y modular.
   - Uso adecuado de buenas prácticas (middlewares, validaciones, separación de lógica).
3. **Interfaz de Usuario:**
   - Interfaz funcional y amigable (no relevante).
4. **Eficiencia:**
   - Consultas a la base de datos optimizadas.
5. **Entrega Completa:**
   - Instrucciones claras para correr el proyecto (en un archivo README.md).

Entrega

- Subir el código a un repositorio público en GitHub.
- Incluir un archivo README.md con las instrucciones para instalar y correr el proyecto:
  - Configuración del entorno.
  - Credenciales de usuario predeterminadas para pruebas.