Prueba Técnica Full Stack

Descripción del proyecto

Desarrollar una aplicación de gestión de tareas **"Task Manager"** que permita a los usuarios:

- Crear, leer, actualizar y eliminar tareas.
- Visualizar la lista de tareas en una interfaz intuitiva y moderna.
- Marcar tareas como completadas o pendientes.

Requerimientos técnicos

1. Backend

1. **Recursos principales Tarea** (`Task`):
    - ID único generado automáticamente.
    - Título (`title`) - Texto obligatorio.
    - Descripción (`description`) - Texto opcional.
    - Estado (`completed`) - Booleano, por defecto `false`.
    - Fecha de creación (`createdAt`) - Fecha generada automáticamente.
2. **Endpoints requeridos**
    - `POST /api/tasks`:
        - Crea una nueva tarea.
        - Valida que el campo `title` esté presente.
    - `GET /api/tasks`:
        - Devuelve la lista de tareas.
        - Debe incluir la opción de filtrar por estado (`completed` o `pending`).
    - `GET /api/tasks/:id`:
        - Devuelve los detalles de una tarea específica.
    - `PUT /api/tasks/:id`:
        - Permite actualizar los campos de una tarea.
    - `DELETE /api/tasks/:id`:
        - Elimina una tarea.
3. **Requerimientos adicionales para el backend**
    - Usa **MongoDB** con **Mongoose** para la base de datos.
    - Implementa validaciones con `express-validator`.
    - Documenta los endpoints con **Swagger**.
    - Manejo de errores estructurado con respuestas claras (códigos 400, 404, 500).

---

2. Frontend

1. **Funcionalidades**
    - Pantalla principal con una lista de tareas.
        - Cada tarea debe mostrar: título, estado (completada o pendiente) y fecha de creación.
        - Botones para editar o eliminar una tarea.
    - Formulario para agregar una nueva tarea.
    - Función para marcar tareas como completadas o pendientes desde la lista.
    - Filtro para visualizar solo tareas completadas, pendientes o todas.
    - Diseño responsivo para desktop y móvil.
2. **Requerimientos técnicos para el frontend**
    - Usa **React.js** con **Chakra UI** o **Tailwind CSS**.
    - Maneja el estado global con **Context API** o una librería como Redux.
    - Integra la API del backend para todas las operaciones.
    - Maneja los errores de la API y muestra mensajes claros al usuario.

---

3. Despliegue

1. **Backend**:
    - Despliega el backend en **Render** o **Railway**.
2. **Frontend**:
    - Despliega el frontend en **Vercel** o **Netlify**.
3. **Documentación**:
    - Proporciona un archivo `README.md` con:
        - Enlace a la aplicación desplegada.
        - Pasos para instalar y ejecutar el proyecto localmente.
        - Detalles de configuración (por ejemplo, variables de entorno).

Criterios de evaluación

1. **Backend**:
    - Correcta implementación de la API REST y las validaciones.
    - Uso adecuado de MongoDB y Mongoose.
    - Documentación clara en Swagger.
2. **Frontend**:
    - Interfaz limpia, funcional y responsiva.
    - Correcta integración con el backend.
    - Manejo eficiente del estado y las operaciones CRUD.

3. **Código**:
    - Estructura clara y organizada del proyecto.
    - Uso de buenas prácticas en ambos lados (frontend y backend).
4. **Extras opcionales**:
    - Implementa autenticación con JWT para proteger los endpoints del backend.
    - Escribe pruebas unitarias para el backend (con Jest) y el frontend (con React Testing Library).
    - Agrega animaciones para mejorar la experiencia del usuario.

Entregables

1. Repositorio en GitHub (uno o dos repos separados para frontend y backend).
2. URL de la aplicación desplegada.
3. Documentación clara en el `README.md`.