Proyecto Final: Gestor de Tareas Full Stack

Fecha Revisión/Consultas: Jueves 15/05/2025 19:00

Fecha Entrega: Domingo 18/05/2025 14:00

Objetivo

Desarrollar una aplicación completa de gestión de tareas con **React (Frontend)**, **Node.js y Express (Backend)**, **PostgreSQL o MongoDB (Base de Datos)**, y desplegarla en **Vercel u otro (Frontend)** y **Render u otro (Backend)**.

Requisitos del Proyecto

1. Funcionalidades del Gestor de Tareas

- **Registro e inicio de sesión** (JWT y bcrypt.js para autenticación).
- **CRUD de tareas**:
    - Crear una tarea con título, descripción, estado y fecha límite.
    - Editar una tarea.
    - Marcar una tarea como completada.
    - Eliminar una tarea.
- **Filtrado y búsqueda** de tareas por estado o fecha.
- **Conexión con PostgreSQL o MongoDB** mediante Sequelize ORM o Mongoose ADM.

Punto 1: Funcionalidades del Gestor de Tareas.

1.1. Autenticación de Usuarios

- **Registro de usuario** con nombre, email (único) y contraseña.
- **Inicio de sesión** con email y contraseña.
- **Protección de rutas privadas** (solo usuarios autenticados pueden acceder a la gestión de tareas).
- **Autenticación con JWT** para manejar sesiones seguras.
- **Contraseñas hasheadas** con bcrypt.js.

1.2. Gestión de Tareas (CRUD)

- **Crear tarea** con:
    - Título (obligatorio).
    - Descripción (opcional).

    - Estado (pendiente, en progreso, completada).
        - Una tarea nueva siempre inicia como "pendiente".
    - Fecha límite (opcional, validado).
- **Editar una tarea** (modificar título, descripción, estado o fecha límite).
    - Solo se puede marcar como "en progreso" si está en "pendiente".
    - No se puede volver a "pendiente" desde "en progreso" o "completada".
    - Una vez "completada", no se puede modificar (solo eliminar).
- **Marcar una tarea como completada**.
    - Solo se puede marcar como "completada" si está en "en progreso".
- **Eliminar una tarea**.
    - Solo si la tarea está en estado “Completada”

1.3. Filtrado y Búsqueda

- **Filtrar tareas** por estado (pendiente, en progreso, completada).
- **Filtrar por Fechas**
- **Buscar tareas** por título o palabra clave en descripción.

1.4. Conexión con Base de Datos

- Cada usuario solo podrá ver y gestionar sus propias tareas.
- **PostgreSQL** o MongoDB almacena los datos con relaciones entre usuarios y tareas.
- **Sequelize ORM o Mongoose** maneja las consultas y operaciones en la base de datos.

1.5. Flujo de Trabajo y Endpoints

Autenticación de Usuarios (Registro e Inicio de Sesión)

Acción	Método	Endpoint	Descripción
Registro de usuario	POST	/api/auth/register	Crea un nuevo usuario con nombre, email y contraseña encriptada.
Inicio de sesión	POST	/api/auth/login	Verifica las credenciales del usuario y devuelve un token JWT.
Obtener datos del usuario autenticado	GET	/api/auth/me	Retorna los datos del usuario basado en el token JWT.

Ejemplo de JSON para el registro (/api/auth/register)

{

```
 "name": "Juan Pérez",
 "email": "juan@example.com",
 "password": "123456"
}
```


Ejemplo de respuesta exitosa (/api/auth/login)


```
{
 "message": "Login exitoso",
 "token": "eyJhbGciOiJIUzI1NiIsInR..."
}
```


Gestión de Tareas (CRUD - Crear, Leer, Actualizar, Eliminar)

Acción	Método	Endpoint	Descripción
Crear tarea	POST	/api/tasks	Crea una nueva tarea asociada al usuario autenticado.
Obtener todas las tareas del usuario	GET	/api/tasks	Devuelve la lista de tareas del usuario autenticado.
Obtener una tarea específica	GET	/api/tasks/:id	Devuelve una tarea específica del usuario.
Actualizar tarea	PUT	/api/tasks/:id	Modifica los datos de una tarea existente.
Eliminar tarea	DELETE	/api/tasks/:id	Elimina una tarea del usuario.

Ejemplo de JSON para crear una tarea (/api/tasks)


```
{
 "title": "Completar el proyecto",
 "description": "Terminar la última parte del backend",
 "status": "pendiente",
 "dueDate": "2025-03-20"
}
```


Ejemplo de respuesta exitosa (/api/tasks)


```
{
 "message": "Tarea creada exitosamente",
 "task": {
```

```
  "id": 1,
  "title": "Completar el proyecto",
  "description": "Terminar la última parte del backend",
  "status": "pendiente",
  "dueDate": "2025-03-20",
  "userId": 3
 }
}
```


Filtrado y Búsqueda

Acción	Método	Endpoint	Descripción
Filtrar tareas por estado o fecha	GET	/api/tasks?status=completada	Devuelve solo las tareas que coincidan con el estado dado.
Buscar tareas por palabra clave	GET	/api/tasks?search=proyecto	Retorna tareas cuyo título o descripción contengan la palabra clave.

Ejemplo de URL para filtrar tareas completadas:

GET /api/tasks?status=completada

Ejemplo de URL para buscar "proyecto" en tareas:

GET /api/tasks?search=proyecto

Protección de Rutas con JWT

- Todas las rutas de tareas deben estar protegidas con autenticación **JWT**.
- Se debe incluir el **token JWT** en el **header de cada petición protegida**:

Ejemplo de header con JWT:


```
{
 "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR..."
}
```


2. Frontend: React (Vercel)

- Estructura con componentes reutilizables (por ejemplo, TaskCard, TaskList, etc.).
- React Router para navegación (Login, Dashboard, etc.).
- Estado global con Context API o Redux.

- Consumo de la API REST con Fetch o Axios.
- Diseño responsive con CSS Modules, Tailwind o Styled Components.

3. Backend: Node.js + Express (Render)

- API REST con Express.js (endpoints para usuarios y tareas).
- Autenticación con JWT y bcrypt.js.
- Manejo de errores y validaciones con Express Validator.
- CORS habilitado para permitir la conexión con el frontend.

4. Base de Datos: PostgreSQL (Render) o MongoDB (MongoDB Atlas)

- Modelo de usuario (id, nombre, email, contraseña hasheada).
- Modelo de tarea (id, título, descripción, estado, fecha límite, usuarioId).
- Relación entre usuarios y tareas (cada usuario gestiona sus propias tareas).
- ORM Sequelize o ODM Mongoose para interacción con la base de datos.

5. Repositorio en GitHub

- Repositorio público con documentación en README.
- Uso de Git con commits significativos.

6. Despliegue

- Backend en Render u otro.
- Base de Datos en Render u otro (PostgreSQL) MongoDB Atlas u otro (MongoDB).
- Frontend en Vercel u otro.
- Variables de entorno correctamente configuradas.

7. Criterios de Evaluación

Criterio	Puntos
Funcionalidad completa del CRUD	35
Autenticación segura (JWT + bcrypt.js)	25
Frontend bien estructurado y responsive	20
Conexión exitosa entre frontend y backend	10
Despliegue funcional	10
Total	**100**

8. Entrega Final

- Informe en .pdf del testeo de rutas en Postman
- Enlace al repositorio en GitHub con el código bien organizado.
- Enlaces al backend en Render y al frontend en Vercel.
- Video corto (máx. 5 min) explicando la funcionalidad de la aplicación.