Prueba Técnica - Gestión de Tareas Full Stack

Objetivo

Desarrollar una aplicación de gestión de tareas que incluya tanto el backend como el frontend.
El candidato debe demostrar su capacidad para estructurar un backend eficiente y aplicar conceptos de desarrollo en el frontend.

Requisitos

Funcionalidad principal

La aplicación debe permitir:
Crear una nueva tarea.
Ver una lista de tareas pendientes y completadas.
Marcar una tarea como completada o pendiente.
Eliminar una tarea de la lista.
Filtrar la vista entre tareas completadas y pendientes.

Backend (NodeJs, Python, Java, o .NET Core)

Implementar un **API RESTful** con los siguientes endpoints:

- `POST /tasks` → Crear una nueva tarea.
- `GET /tasks` → Obtener la lista de tareas.
- `PATCH /tasks/{id}` → Marcar una tarea como completada o pendiente.
- `DELETE /tasks/{id}` → Eliminar una tarea.
  Usar una base de datos en memoria (SQLite o MongoDB).
  Aplicar principios de **limpieza de código** y **arquitectura en capas** (servicios, repositorios, controladores).
  Incluir manejo de errores y validaciones.
  Incluir pruebas unitarias en al menos un endpoint.

Frontend (Angular)

Consumir el API del backend y manejar los datos en el estado con RxJS.
Implementar un servicio para la gestión de tareas (CRUD).
Aplicar **lazy loading** para modularizar la aplicación.
Utilizar **ChangeDetectionStrategy.OnPush** en al menos un componente.
Permitir la interacción del usuario mediante una UI simple y funcional.
(Opcional) Implementar pruebas unitarias en al menos un servicio o componente.

Entrega

Subir el código en un repositorio (GitHub, GitLab, Bitbucket).
Incluir un **README.md** con instrucciones para ejecutar el backend y frontend.
Backend y frontend deben estar en carpetas separadas (`/backend` y `/frontend`).

Duración estimada: 3 - 4 horas

Criterios de Evaluación

Funcionalidad: Correcta implementación de CRUD y filtrado.
Código limpio y buenas prácticas: Separación de responsabilidades, modularización y validaciones.
Eficiencia y optimización: Uso de RxJS en Angular y arquitectura en capas en el backend.
Manejo de errores: Backend con respuestas adecuadas ante errores.
Pruebas: Valoradas positivamente si se incluyen.
Autogestión y documentación: README con instrucciones claras.