Prueba Técnica

Objetivo

Desarrollar una aplicación web que permita crear y listar usuarios, utilizando un backend simulado creado por ti.

No buscamos lógica compleja, sino una ejecución profesional, bien diseñada, organizada y visualmente limpia.

Tecnologías permitidas

- Frontend: React, Vue, Angular o HTML/JS/CSS (a tu elección).
- Mock backend (debes crearlo tú mismo):
    - json-server (local)
    - MockAPI.io (en la nube)

Diseño del esquema

Como parte de la prueba, debes definir y construir una base de datos simulada (usuarios) con los siguientes campos:

Campo	Tipo	Requerido	Validación mínima
id	number	✅	Autogenerado por el sistema
nombre	string	✅	No vacío
email	string	✅	Formato válido
edad	number	✅	Mayor a 0

Si usas:

- json-server: debes crear un archivo db.json con este esquema.
- MockAPI.io: debes crear la colección usuarios y definir estos campos desde su panel.

Este paso forma parte **esencial del proyecto**: la prueba evalúa tu capacidad para estructurar datos y exponerlos como un recurso REST válido.

Requisitos funcionales

1. Listar usuarios (GET /usuarios)
2. Agregar usuario nuevo (POST /usuarios)
3. Formulario con validación visual:
    - Nombre requerido
    - Email válido
    - Edad positiva

Calidad esperada

- Diseño moderno y limpio
- Código modular, mantenible
- Manejo de errores, carga, éxito
- Buen uso de Git y documentación
- Profesionalismo en presentación

Entrega

Repositorio en GitHub con:

- Código fuente frontend
- Backend simulado (archivo db.json o link de MockAPI)

README con:

- Pasos para ejecutar backend
- Pasos para ejecutar frontend
- Explicación técnica y capturas (opcional)

Criterios de Evaluación

Obligatorios:

- Manejo de errores en UI (por ejemplo, si la API falla).
- Interfaz responsive (adaptada a móvil).
- Código modular, limpio y mantenible.
- Buen uso de Git: commits descriptivos y ordenados.

Opcionales / Bonus:

- Editar usuarios (PUT /usuarios/:id)
- Eliminar usuarios (DELETE /usuarios/:id)
- Implementar con Node.js un pequeño backend en vez de usar MockAPI/json-server.
- Uso de pruebas unitarias básicas (Jest/Vitest).
- Deployment (por ejemplo, Vercel o Netlify /etc.)

Colaboración y Crecimiento (reflexión requerida)

Imagina que estás trabajando en equipo y este proyecto será parte de algo más grande. Para evaluar tu mentalidad colaborativa y visión a largo plazo, incluye un archivo llamado reflexion.md en el repositorio, con las siguientes respuestas:

1. **¿Cómo organizarías tu trabajo si esto fuera parte de un equipo ágil?**
   Describe brevemente cómo te coordinarías con otros miembros, cómo dividirías tareas y qué herramientas usarías (por ejemplo, Jira, Trello, GitHub Projects).

Si este proyecto formara parte de un equipo ágil, me organizaría aplicando prácticas de metodologías como Scrum, participando en reuniones clave para mantener una coordinación constante con el equipo. Utilizaría Jira como herramienta principal para gestionar el backlog, estimar tareas con story points y planificar sprints de forma estructurada. Además, emplearía Trello como apoyo visual para organizar los avances por categoría o tipo de tarea, facilitando una vista rápida del progreso individual y de equipo. La combinación de estas herramientas permitiría una gestión eficiente y colaborativa del desarrollo.

2. **¿Qué decisiones técnicas tomaste y por qué?**
   Explica por qué elegiste ciertas librerías, estructura de carpetas, validaciones, o arquitectura.

Para el desarrollo del backend opté por el framework Express, ya que permite levantar un servidor de manera rápida y flexible. Utilicé el ORM Sequelize para facilitar la interacción con la base de datos mediante un enfoque orientado a objetos, junto con dotenv para manejar las variables de entorno de forma segura. También integré CORS, configurado de forma abierta para permitir solicitudes desde localhost durante el desarrollo. En cuanto a validaciones, aproveché las capacidades integradas de Sequelize a través de la opción validate, y utilicé transacciones para asegurar la integridad de los datos en operaciones críticas. Para el frontend elegí Angular, un framework robusto que permite estructurar el código de forma modular y organizada, separando claramente vistas, lógica y servicios. La arquitectura general del proyecto sigue el modelo cliente-servidor, y tanto en el backend como en el frontend se aplicó una estructura de carpetas basada en el principio de responsabilidad única, facilitando el mantenimiento y la escalabilidad del código.

3. **¿Qué aprendiste al hacer esta prueba y qué mejorarías en una segunda versión?**

Al realizar esta prueba, reforcé la importancia de mantener una buena modularidad tanto en la organización del código como en la arquitectura general del proyecto. También valoré aún más el uso adecuado de ramas en el control de versiones; en una próxima versión, optaría por separar el desarrollo en ramas independientes para backend y frontend, lo que permitiría una mejor gestión de permisos, revisiones y despliegues. Además, identifiqué que trabajar con múltiples versiones de Node puede generar conflictos, especialmente si se utiliza nvm en ciertos entornos; por ello, mejoraría el proceso estableciendo desde el inicio una versión única y fija de Node para todo el equipo, asegurando así consistencia y evitando errores en entornos locales.

Este archivo será valorado como parte del profesionalismo en la entrega.