🧪 Technical Test – Fullstack Developer (React + Node.js)

🎯 Objective

Construir un sistema Fullstack que obtenga datos en tiempo real desde la API pública de **REE (Red Eléctrica de España)** — específicamente los datos de **Balance Eléctrico** —, los almacene en una base de datos **MongoDB**, y los exponga mediante una **API GraphQL**. Además, debe incluir un **frontend en React** que consuma esa API y muestre la información de forma clara e interactiva.

El sistema debe estar **contenedorizado con Docker** y contar con **testing y documentación adecuados**.

🔌 Data Source and API

Usar el siguiente endpoint público de REE:
📍 https://apidatos.ree.es/es/datos/balance/balance-electrico
Este endpoint proporciona información del balance eléctrico nacional: generación, demanda, importaciones/exportaciones, etc.

NOTA:
Las consultas al endpoint requieren el uso de parámetros de forma obligatoria, en caso de realizar una consulta sin parámetros el endpoint devuelve un error 500. La especificación del api se puede encontrar en https://www.ree.es/es/datos/apidatos.
Un ejemplo de consulta con parámetros sería: https://apidatos.ree.es/es/datos/balance/balance-electrico?start_date=2019-01-01T00:00&end_date=2019-01-31T23:59&time_trunc=day

Tu backend debe:

- Consultar la API de forma periódica.
- Almacenar los datos en MongoDB.
- Exponer la información mediante una API GraphQL para que el frontend la consuma.

📦 Backend Requirements

Mínimos requeridos:

- Usar **MongoDB** para guardar datos históricos y actuales del balance eléctrico.
- Crear una API **GraphQL** con al menos las siguientes queries:
    - Obtener datos de balance eléctrico por rango de fechas.
- Implementar:
    - Validación del schema.
    - Manejo de errores robusto.
    - Fallback elegante si la API de REE no está disponible.
- Contenerizar todo el backend con **Docker**.
- Escribir pruebas **unitarias e integradas** para funcionalidades clave (fetching, procesamiento, resolvers).

🖥️ Frontend Requirements (React)

Desarrollar una interfaz web con **React** que consuma la API GraphQL del backend y muestre la información de forma visual.

Mínimos requeridos:

- Crear una SPA (Single Page Application) con React.
- Conectar al backend a través de GraphQL (puedes usar Apollo Client).
- Visualizar al menos:
    - Datos de balance eléctrico para un rango de fechas.
    - Representación gráfica (por ejemplo, con **chart.js**, **recharts** o similar).
- Implementar manejo de errores (loading states, retries si falla la API).
- Estructura clara y componentes reutilizables.

✅ Testing

- Incluir pruebas significativas utilizando herramientas como **Jest**, **React Testing Library**, **Mocha** u otras.
- Cubrir:

- La ingesta de datos (backend).
- La API GraphQL.
- Componentes y lógica del frontend (React).

📤 Deliverables

1. Repositorio público en **GitHub** con todo el código fuente (frontend y backend).
2. Instrucciones para correr el proyecto:
   - Backend y frontend localmente.
   - Todo el sistema vía Docker / Docker Compose.
3. Un archivo README.md completo que incluya:
   - Descripción del pipeline de datos y modelo de datos.
   - Cómo ejecutar y testear el backend.
   - Cómo ejecutar y testear el frontend.
   - Cómo obtener y actualizar los datos de REE.
   - Consultas GraphQL de ejemplo y respuestas esperadas.
   - Capturas o gifs del frontend en funcionamiento.

🧠 Evaluation Criteria

- Arquitectura limpia y modular (backend y frontend).
- Uso correcto de MongoDB, Docker, GraphQL y React.
- Buena integración con la API pública de REE.
- Manejo robusto de errores y validaciones.
- Código bien estructurado y testeado.
- Interfaz clara, funcional y conectada al backend.
- Documentación útil y profesional.
