Desafío Técnico

El desafío técnico se divide en 2 partes independientes **Backend** y**Frontend**.
El objetivo es consumir desde el backend la API de beneficios de Sportclub brindada a continuación* y construir diferentes soluciones basadas en este requerimiento.

Endpoints:
*Get beneficios: https://api-beneficios.dev.sportclub.com.ar/api/beneficios
*Detalle del beneficio: https://api-beneficios.dev.sportclub.com.ar/api/beneficios/8

Diseño: se puede tomar como referencia esta landing*, sera responsabilidad del desarrollador crear un diseño autentico
*https://beneficios.sportclub.com.ar/

Parte 1: Backend Python

Objetivo
Construir una api-rest que actúe como intermediario (pasamanos) entre la API de Sportclub y el frontend creado por el candidato.

Requerimientos
1. **Tecnologías:**
- Python 3.12.
- Framework Flask, Django o FastApi.
- Testing con unittest o pytest.
- No se requiere conexión a bases de datos.

2. **Endpoints a crear:**
  - `GET /api/beneficios`:
    - Consume la API de Sportclub y devuelve la lista de beneficios.

  - `GET /api/beneficios/:id`:
- Consume la API de Sportclub y devuelve los detalles de un beneficio. Si el beneficio no existe en la API, devolver un error 404 con un mensaje claro.
Implementar logging, traceback claro y utilizando las librerías nativas de Python.

3. **Notas adicionales:**
- Manejar errores como tiempos de espera agotados, datos corruptos, o API caída.
- Validar y normalizar los datos recibidos de la API.

Considerar principios de arquitectura limpia (Clean Architecture) para organizar el proyecto.
Testing: Cubrir al menos el 90% de los endpoints con tests unitarios y de integración.

Parte 2: Frontend

Construir una aplicación de frontend que consuma la API y muestre los beneficios.
Esta app debe constar con 3 componentes principales, caja de busqueda, vista de resultados, y detalle de los beneficios

Descripcion funcional

En la vista de caja de búsqueda, se debe poder ingresar el producto a buscar y al enviar el formulario navegar a la vista de Resultados de búsqueda.
Luego, al hacer click sobre uno de ellos, debería navegar a la vista de Detalle de Producto.

Requerimientos

1. **Tecnologías:**
   - React
   - Herramientas adicionales para usabilidad, SEO y performance son bienvenidas.
   - Librerías complementarias como axios o zustand para manejo de datos.
   - Implementar testing con una herramienta como Jest, Playwright o Vitest.

2. **Vistas principales:**
   - **Lista de Beneficios**:
     - Mostrar una lista de beneficios recuperados desde la API creada por el candidato.
     - Cada beneficio debe mostrar:
       - Nombre del beneficio.
       - Descripción breve.
       - Imagen asociada.
       - Estado (activo o inactivo).
     - Agregar un buscador para filtrar por nombre y estado.
   - **Detalle del Beneficio**:
- Al seleccionar un beneficio, mostrar una vista detallada con toda la información del beneficio.
Agregar funcionalidad para marcar como "Favorito" y guardar este estado localmente en IndexedDB o LocalStorage.

3. **URLs esperadas:**
   - `/beneficios`: Muestra la lista de beneficios.
- `/beneficios/:id`: Muestra el detalle de un beneficio.

4. **Notas adicionales:**
   - Manejar estados de carga y errores.
- Implementar un diseño responsivo.
- Implementar paginacion
- Implementar carga diferida (lazy loading) para imágenes y recursos no críticos.
Usabilidad:
Debe ser totalmente accesible (cumplir estándares de WCAG 2.1 AA).
Testing: Cubrir al menos el 80% de las funcionalidades clave con tests.

Entregable
Repositorio en GitHub con instrucciones para instalar ,ejecutar y correr los test.

Documentación:
Incluir README detallado que explique la arquitectura utilizada y cómo se solucionaron problemas de performance y escalabilidad.

Estandarización:
Cumplir con linters y formateadores

Despliegue:
Ambas aplicaciones deberán estar dockerizadas respectivamente indicando cómo levantarlas.
