Objetivo general

Desarrollar un componente en React que consuma un API protegido mediante API Key, aplicando los hooks useState y useEffect para manejar estados, efectos y peticiones asíncronas.

Objetivos específicos

- Utilizar **useState** y **useEffect** para gestionar estados y efectos en React.
- Consumir el **API de productos** usando **API Key** y negociación de contenido **JSON/XML**.
- Implementar **paginación** y **selector de tamaño de página** dinámico.
- Aplicar **ordenamiento del lado del cliente** por campos como *name* o precio.
- Mostrar el **detalle de producto** y su vista **Raw** según el formato seleccionado.
- Incorporar **skeletons**, **mensajes de error** y **estado vacío** para mejorar la experiencia de usuario.

Instrucciones

1) Requisitos técnicos

- Usar **useState** y **useEffect**.
- Endpoints (Lab 8) con **API Key** y **negociación de contenido**:
- **GET /products?page=&limit=**
- **GET /products/:id**
- Headers obligatorios:
    - **x-api-key: <TU_API_KEY>**
    - **Accept: application/json o application/xml** (según selección del usuario).

2) Página única

A) Controles superiores

- **Selector de formato**: **<select>** para JSON/XML que cambie **Accept** y dispare nueva carga.
- **Selector de tamaño de página (page size)**: **<select>** con opciones (p. ej. 6, 12, 24, 48). Cambiarlo ajusta **limit** y **reinicia page=1**.
- **Ordenamiento (lado cliente)**: **<select>** con opciones:
    - **name:asc**, **name:desc**
    - **price:asc**, **price:desc**

  El ordenamiento **no** se envía al servidor; **se aplica en memoria** sobre los datos recibidos.

B) Listado de productos (cards)

- Presentar en **grid responsivo**.
- Cada **card** debe mostrar **name** y **sku** (obligatorio).
- Al hacer clic en una card, abrir la **sección de detalle** (ver C).

C) Detalle de producto

- Mostrar los **campos restantes** disponibles en un modal (p. ej. description, price, category, image, etc.).
- Incluir un **toggle "Raw"** que muestre la **respuesta cruda** del endpoint de detalle:
    - Si JSON → pretty-print.

- Si XML → texto plano.

Debe poder alternarse entre **vista amigable** y **Raw** sin volver a pedir al servidor.

D) Paginación

- Controles **Anterior / Siguiente** y etiqueta **page / totalPages**.
- Al cambiar **page**, **volver a consultar /products?page=&limit=**.

E) Estados de UI

- **Cargando (listado y detalle)**: Mostrar **skeletons**
- **Vacío (listado)**:Si la página actual no trae ítems, mostrar “No hay productos para mostrar.”
- **Error**: Mensaje claro y permitir **Reintentar**.

Rubrica de evaluación

La evaluación del laboratorio se realizará bajo los siguientes criterios:

Criterio	1 – Insuficiente	3 – Aceptable	5 – Excelente
Implementación de useState y useEffect	Los estados o efectos no se usan correctamente o hay recargas infinitas.	Se utilizan para manejar datos, paginación y formato, pero con dependencias parciales o sin cleanup.	useState y useEffect correctamente implementados, con dependencias precisas.
Consumo de API y negociación de contenido	No se envían headers obligatorios (x-api-key, Accept) o las peticiones fallan.	Las solicitudes funcionan para un formato (JSON o XML).	Implementa correctamente la **negociación de contenido** entre JSON/XML con selector funcional y parsing apropiado.
Paginación y tamaño de página	La paginación no cambia resultados o tiene errores de límites.	La paginación básica funciona y cambia correctamente entre páginas.	Paginación fluida y confiable, **sin errores de sincronización**, con **selector de tamaño de página** que reinicia page=1 correctamente.
Ordenamiento cliente	No hay ordenamiento o no se refleja en la interfaz.	Se aplica ordenamiento básico (por un campo).	Implementa ordenamiento completo por **name** y **sku**, y opcionalmente **price**; aplica sobre los datos correctamente sin perder el estado de paginación.
Detalle de producto y vista Raw	No muestra detalle o no cambia entre formatos.	Muestra detalle, pero sin vista Raw o con datos incompletos.	Detalle completo con **toggle entre vista amigable y Raw** (JSON/XML) correctamente presentado y accesible.
Estados de UI (carga, error, vacío)	No hay indicación visual o se usa solo “Loading…”.	Implementa mensajes básicos de carga y error.	Muestra **skeletons de carga**, **mensajes claros de error** con role="alert" y **empty state** accesible.

Entrega

Cada estudiante deberá entregar un archivo comprimido (.zip) con el proyecto completo, siguiendo estas instrucciones:

1. Un archivo denominado enlaces.txt que contenga
    a. URL del repositorio público en GitHub:

Fecha límite: Viernes 24 de octubre 5:00 p.m.
Forma de entrega: El archivo .zip debe subirse a la plataforma TEC Digital, en el espacio correspondiente al curso.
Portafolio del curso: Este laboratorio forma parte del portafolio individual del curso.

Todo el trabajo presentado deberá ser **original y producto del esfuerzo del estudiante**. Cualquier evidencia de **plagio total o parcial**, **uso indebido de código ajeno**, o la **utilización excesiva de herramientas de inteligencia artificial** sin un aporte real del estudiante será considerada una **falta grave a la integridad académica**.

En tales casos, el laboratorio recibirá una **nota de 0 (cero)** y se procederá con la **inclusión del incidente en el expediente académico** del estudiante, conforme a la normativa institucional vigente del Tecnológico de Costa Rica.

El uso de tecnologías de apoyo (incluyendo IA) solo será aceptado si se emplea de manera responsable y con **criterio académico**, como herramienta de consulta o asistencia, y **no como sustituto del trabajo propio** en la redacción del código, la documentación o los entregables evaluados.