Trabajo Práctico integrador grupal de presentación obligatoria.

El presente Trabajo Práctico Integrador tiene como finalidad la aplicación y consolidación de los conocimientos adquiridos durante la cursada de Programación Visual. A lo largo de este proyecto, se abordará el desarrollo progresivo de una aplicación web funcional, utilizando React, Vite, JavaScript, React Router DOM, y todos los temas vistos en la materia.

Es importante aclarar que la metodología de evaluación para este Trabajo Práctico Integrador se realizará de forma continua y progresiva, alineada con el carácter evolutivo del desarrollo del proyecto.

Las evaluaciones se realizarán de manera periódica, con una frecuencia semanal, y estarán relacionadas con los avances presentados de acuerdo a los enunciados y requerimientos que el profesor a cargo de la comisión exija, por ejemplo, defensa oral con cámara y micrófono activo.

1. Introducción y Objetivo del Proyecto

El objetivo de este Trabajo Práctico Final es integrar la totalidad de los conceptos estudiados durante el cuatrimestre para construir un **Panel de Control de Clientes** en tiempo real. La aplicación dejará de utilizar datos simulados en memoria local y se conectará de forma asrincrónica a un servidor real en la nube a través de la API pública **FakeStoreAPI**. La aplicación debe estar estructurada en componentes funcionales y manejar correctamente los eventos en React y la navegación entre vistas.

Configuración del proyecto

Crear el proyecto utilizando Vite con plantilla de React.

Utilizar JSX para estructurar la interfaz.

Usar React Router Dom

Usar framework de interfaces de usuario

2. Tecnologías e Integración Requerida

Para aprobar el proyecto se evaluará la coexistencia y correcta implementación de los siguientes pilares:

- **Entorno:** React de nivel profesional inicializado mediante **Vite**.
- **Estética y Diseño:** Maquetado responsivo y adaptado al sistema de diseño de algún framework de interface de usuario **(MUI o React Bootstrap)**.
- **Enrutamiento:** Navegación entre pantallas sin recarga del navegador mediante **React Router**.
- **Estado Global:** Control de accesos y perfiles mediante **Context API** combinado con **LocalStorage**.
- **Asincronismo:** Peticiones HTTP estructurales utilizando la sintaxis moderna de async/await mediante fetch o la biblioteca Axios.

3. Estructura Obligatoria de Directorios

El proyecto debe ordenarse bajo una arquitectura limpia y modular de carpetas. No se aceptarán componentes sueltos ni mezclas de responsabilidades:

```
src/
├── components/
│   ├── layout/          # Componentes de estructura fijos (Header, Nav, Footer)
│   └── common/          # Componentes reutilizables pequeños (Formularios, Filas de Tabla, etc.)
├── context/
│   └── AdminContext.jsx   # Estado global del Administrador
├── views/
│   ├── Login.jsx          # Pantalla de acceso del Administrador
│   ├── Dashboard.jsx      # Panel principal con métricas globales
│   ├── ListaClientes.jsx   # Vista de Clientes con consumo de API (GET /users)
│   └── DetalleCliente.jsx  # Ficha profunda del Cliente por ID (GET /users/:id)
├── App.jsx              # Enrutador central y declaración de Providers
└── main.jsx
```

4. Requisitos Funcionales por Módulo

Módulo A: Gestión del Estado Global y Autenticación (Context API + LocalStorage)

Deberán crear el archivo src/context/AdminContext.jsx para administrar quién está operando el sistema.

1. **El Estado:** Al iniciar la aplicación por primera vez, el estado del administrador debe ser null.

2. **Pantalla de Login (/login):** Si el estado es null, el sistema de rutas de React Router debe bloquear el acceso a la aplicación y redirigir obligatoriamente al usuario a la pantalla de Login. El formulario solicitará:
    - Nombre del Administrador.
    - Sector de la Empresa (Menú desplegable con las opciones: "Soporte" o "Gerencia").

3. **Persistencia con LocalStorage:** Al presionar "Ingresar", se activará la función global del contexto para guardar los datos. El AdminProvider deberá utilizar un useEffect para almacenar en tiempo real una copia de esta sesión en el localStorage del navegador. Al presionar **F5 (recargar página)**, la aplicación debe leer el almacenamiento local y evitar que el usuario deba loguearse nuevamente.
4. **Cierre de Sesión:** En la barra del encabezado (<Header />), se deberá mostrar el nombre y el sector del administrador conectado, junto a un botón de "Cerrar Sesión" que borre los datos del localStorage, limpie el contexto y devuelva la aplicación al estado de /login.

Módulo B: Vista de Clientes (/clientes) y Consumo de API

Esta sección representará la tabla principal de trabajo de la empresa. Deberá consumir el recurso asincrónico: **https://fakestoreapi.com/users**

1. **Interfaz Gráfica:** Los datos remotos deben estructurarse utilizando los componentes de tablas profesionales, por ejemplo: (<Table>, <TableHead>, <TableRow>, etc.) o, en su defecto, tarjetas (<Card>) perfectamente alineadas en una cuadrícula (<Grid>). Cada cliente debe listar su ID, Nombre Completo, Email, Teléfono y Ciudad.
2. **Buscador:** Se añadirá una barra de búsqueda que filtre de forma dinámica en la pantalla a los clientes basándose en su apellido (lastname) o en su ciudad (city).
3. **Control de Estados de Carga:** Es obligatorio programar la experiencia de usuario controlando tres estados:
    - *Carga:* Mientras el servidor responde, se mostrará la animación <CircularProgress /> o componentes <Skeleton />.
    - *Éxito:* Renderizado inmediato de la información en la tabla.
    - *Error:* Si la API falla (por ejemplo, al simular una desconexión de red), se capturará el fallo con un bloque try/catch y se informará visualmente mediante un componente <Alert severity="error">.

Módulo C: Formulario de Alta de Clientes

Dentro de la misma pantalla de clientes, o mediante un componente modular integrado, se dispondrá un formulario para dar de alta nuevos usuarios en la base de datos externa.

1. **Petición POST:** Al enviar el formulario, la aplicación disparará una petición HTTP de tipo POST hacia https://fakestoreapi.com/users transmitiendo el objeto de datos estructurado.

2. **Feedback Visual:** Al recibir la respuesta exitosa del servidor remoto (Código 200 o 201), la aplicación deberá capturar el ID asignado por la API y notificar al administrador mediante el componente <Snackbar> o un <Alert> de éxito temporal.

Módulo D: Apartado de Complejidad - Ficha Profunda del Cliente (/clientes/:id)

Al hacer clic en el botón "Ver Ficha Completa" de cualquier fila de la tabla, la aplicación viajará a una ruta dinámica.

1. **Uso de useParams:** El componente <DetalleCliente /> capturará el ID de la URL y realizará un segundo fetch asincrónico específico hacia https://fakestoreapi.com/users/:id.
2. **Tratamiento de Objetos Anidados:** Al recibir la respuesta de FakeStoreAPI, deberán desestructurar y renderizar de forma ordenada en pantalla la dirección completa del cliente mapeando sus propiedades internas (address.street, address.number, address.zipcode, address.city), además de sus credenciales de acceso de la base de datos (username y password).
3. **Lógica de Permisos Globales (Control por Contexto):**
    - Si el Administrador logueado en el Contexto pertenece al sector **"Soporte"**, en esta ficha solo podrá visualizar los datos del cliente.
    - Si el Administrador pertenece al sector **"Gerencia"**, la interfaz habilitará de forma exclusiva un botón rojo de **"Eliminar Cliente de la Base de Datos"**, el cual simulará una petición HTTP de tipo DELETE hacia la API.

5 Versionamiento

Subir el proyecto a un repositorio en GitHub, el nombre del repositorio debe ser **pv_tp_integrador_grupo999**, donde 999 es el número del grupo.

Todos los cambios deben subirse a través de ramas.

En el archivo README.md incluir la lista de integrantes del grupo con sus respectivos nombres de usuarios github.

Añadir breve descripción del proyecto.