**Explicación Detallada de “src/index.js”:**

Importaciones

*import React from "react";*

*import ReactDOM from "react-dom/client";*

*import App from "./App";*

* import React from "react";: Importa la biblioteca React, la cual es esencial para trabajar con JSX y crear componentes de React. Sin esta importación, no podrías utilizar la sintaxis JSX ni las funcionalidades de React.
* import ReactDOM from "react-dom/client";: Importa el módulo ReactDOM desde el paquete react-dom. Este módulo proporciona métodos específicos para manipular el DOM en una aplicación React. En este caso, usamos ReactDOM.createRoot que es el punto de entrada para todas las aplicaciones React en React 18.
* import App from "./App";: Importa el componente principal App desde el archivo App.jsx. Este componente será la raíz de nuestra aplicación y contendrá todos los demás componentes y la lógica de navegación.

Crear el Punto de Entrada

*const root = ReactDOM.createRoot(document.getElementById("root"));*

* const root = ReactDOM.createRoot(document.getElementById("root"));:
  + document.getElementById("root"): Selecciona el elemento del DOM con el id root. Este elemento debe existir en el HTML de la aplicación (normalmente en public/index.html) y es donde React montará la aplicación.
  + ReactDOM.createRoot: Este método crea una raíz para la aplicación React en el elemento seleccionado. Es una nueva forma de inicializar React en la versión 18 y superior, reemplazando el método ReactDOM.render.

Renderizar el Componente Principal

*root.render(*

*<React.StrictMode>*

*<App />*

*</React.StrictMode>*

*);*

* root.render: Llama al método render del nodo raíz creado anteriormente para renderizar la aplicación.
* <React.StrictMode>: Un componente envolvente que ayuda a detectar problemas potenciales en la aplicación. Solo se aplica en modo desarrollo y no afecta el rendimiento en producción.
  + Beneficios:
    - Identifica componentes con efectos secundarios inseguros.
    - Avisa sobre métodos de ciclo de vida obsoletos.
    - Detecta descargas inesperadas de estado.
* <App />: El componente principal de la aplicación que se va a renderizar. Este es el punto de entrada de la aplicación React y contendrá todos los componentes y la lógica de navegación.

¿Qué Pasaría si...?

* Faltan Importaciones:
  + React: Si omites importar React, no podrás utilizar JSX, y el compilador lanzará un error.
  + ReactDOM: Si no importas ReactDOM, no podrás renderizar tu aplicación en el DOM, y verás un error en el navegador.
  + App: Si no importas App, no tendrás un componente principal para renderizar, lo que resultará en un error de módulo no encontrado.
* Elemento DOM No Encontrado:
  + document.getElementById("root"): Si el elemento con id root no existe en el HTML, document.getElementById("root") devolverá null, y ReactDOM.createRoot(null) lanzará un error. Asegúrate de que public/index.html contenga <div id="root"></div>.
* Errores en JSX:
  + <React.StrictMode>: Si se usa incorrectamente o se omite, la aplicación perderá las verificaciones adicionales en modo desarrollo, lo que podría dificultar la identificación de problemas.
  + <App />: Si hay un error en la definición del componente App, o si App no se importa correctamente, el renderizado fallará, y verás un error en la consola del navegador.

Este archivo index.js es crucial porque configura y monta toda la aplicación React en el DOM, proporcionando el punto de entrada inicial desde donde se despliega el resto de la aplicación.

**Explicación Detallada de “src/App.jsx”:**

Importaciones:

*import React from "react";*

*import { BrowserRouter as Router, Route, Routes } from "react-router-dom";*

*import Home from "./components/Home";*

*import CountryList from "./components/CountryList";*

*import CountryDetail from "./components/CountryDetail";*

*import About from "./components/About";*

*import NotFound from "./components/NotFound";*

*import EditCountry from "./components/EditCountry";*

* import React from "react";: Importa la biblioteca React, que es necesaria para crear componentes de React.
* import { BrowserRouter as Router, Route, Routes } from "react-router-dom";: Importa componentes necesarios de react-router-dom para manejar la navegación en una aplicación de una sola página (SPA).
  + BrowserRouter as Router: Renombra BrowserRouter a Router para facilitar su uso.
  + Route: Componente que define una ruta específica y el componente que se debe renderizar para esa ruta.
  + Routes: Componente que agrupa múltiples componentes Route.
* import ...: Importa componentes específicos que serán renderizados en diferentes rutas. Cada importación trae un componente desde el archivo correspondiente en la carpeta components.

Componente App

*function App() {*

*return (*

*<Router>*

*<div className="App">*

*<Routes>*

*<Route path="/" element={<Home />} />*

*<Route path="/countries" element={<CountryList />} />*

*<Route path="/countries/:id" element={<CountryDetail />} />*

*<Route path="/about" element={<About />} />*

*<Route path="/countries/edit/:id" element={<EditCountry />} />*

*<Route path="\*" element={<NotFound />} />*

*</Routes>*

*</div>*

*</Router>*

*);*

*}*

* function App() { ... }: Define el componente funcional App, que es el componente principal de la aplicación.
* return ( ... );: El componente App devuelve el JSX que define la estructura de la aplicación.

Componente Router

*<Router>*

*<div className="App">*

*<Routes>*

*...*

*</Routes>*

*</div>*

*</Router>*

* <Router>: Componente que envuelve toda la aplicación y permite la navegación. Utiliza el historial de la API HTML5 para mantener la interfaz de usuario sincronizada con la URL.
* <div className="App">: Contenedor principal de la aplicación. La clase App se puede usar para aplicar estilos CSS específicos a este contenedor.

Componente Routes

*<Routes>*

*<Route path="/" element={<Home />} />*

*<Route path="/countries" element={<CountryList />} />*

*<Route path="/countries/:id" element={<CountryDetail />} />*

*<Route path="/about" element={<About />} />*

*<Route path="/countries/edit/:id" element={<EditCountry />} />*

*<Route path="\*" element={<NotFound />} />*

*</Routes>*

* <Routes>: Contenedor que agrupa todos los componentes Route. Este componente asegura que solo una ruta se renderice a la vez.
* <Route path="/" element={<Home />} />: Define una ruta específica.
  + path="/": URL que corresponde a esta ruta. En este caso, la raíz del sitio (/).
  + element={<Home />}: Componente que se renderiza cuando la URL coincide con la ruta especificada.

Definición de Rutas

* <Route path="/" element={<Home />} />: Ruta principal que carga el componente Home.
* <Route path="/countries" element={<CountryList />} />: Ruta que carga el componente CountryList.
* <Route path="/countries/:id" element={<CountryDetail />} />: Ruta que carga el componente CountryDetail para un país específico identificado por su id.
  + :id: Parámetro dinámico en la URL que representa el ID del país.
* <Route path="/about" element={<About />} />: Ruta que carga el componente About.
* <Route path="/countries/edit/:id" element={<EditCountry />} />: Ruta que carga el componente EditCountry para editar un país específico identificado por su id.
* <Route path="\*" element={<NotFound />} />: Ruta comodín que carga el componente NotFound para cualquier URL no especificada, actuando como una página 404.

Exportación del Componente App

*export default App;*

* export default App;: Exporta el componente App como la exportación predeterminada del módulo, permitiendo que otros módulos lo importen usando import App from './App';.

¿Qué Pasaría si...?

* Rutas Incorrectas:
  + URL Incorrecta: Si una ruta especificada no coincide con ninguna URL, se renderizará el componente NotFound, lo que proporciona una experiencia de usuario más robusta al manejar errores de navegación.
  + Sin Rutas Definidas: Si no hay rutas definidas en <Routes>, la aplicación no tendrá contenido dinámico y solo mostrará el contenedor vacío.
* Componentes Faltantes:
  + Importaciones Incorrectas: Si se omite importar un componente necesario, el código fallará en tiempo de compilación con un error de "módulo no encontrado".
  + Exportaciones Incorrectas: Si los componentes no están correctamente exportados desde sus archivos respectivos, no podrán ser importados y utilizados en App.jsx.
* Errores en JSX:
  + Errores de Sintaxis: Si hay errores de sintaxis en el JSX, el compilador mostrará un error y la aplicación no se renderizará.
  + Componentes Mal Definidos: Si los componentes pasados a element no están correctamente definidos o contienen errores, no se renderizarán como se espera.

Este archivo App.jsx gestiona la estructura principal de la aplicación y define cómo los diferentes componentes se renderizan en función de la URL actual, proporcionando una navegación fluida y una mejor experiencia de usuario.

**Explicación Detallada de “src/api.js”:**

Importación de Axios

Javascript:

*import axios from "axios";*

* import axios from "axios";: Esto importa la biblioteca axios, que se usa para realizar solicitudes HTTP desde el navegador. axios ofrece una interfaz más sencilla y características adicionales en comparación con la función nativa fetch.

Función fetchCountries

Javascript:

*export const fetchCountries = async () => {*

*try {*

*const response = await axios.get("http://45.236.130.191/paises.php");*

*return response.data;*

*} catch (error) {*

*console.error("Error fetching countries:", error);*

*return [];*

*}*

*};*

* export const fetchCountries = async () => { ... };: Define una función asíncrona llamada fetchCountries que realiza una solicitud GET para obtener una lista de países desde una API externa.
  + async: Palabra clave que indica que la función es asíncrona y puede usar await dentro de ella.
* try { ... } catch (error) { ... }: Utiliza un bloque try-catch para manejar posibles errores al realizar la solicitud.
  + try { ... }: Intenta ejecutar el bloque de código. Si ocurre un error, se ejecutará el bloque catch.
* const response = await axios.get("http://45.236.130.191/paises.php");: Realiza una solicitud GET a la URL especificada y espera a que se complete la solicitud.
  + await: Espera a que la promesa devuelta por axios.get() se resuelva antes de continuar.
  + axios.get("http://45.236.130.191/paises.php"): Realiza una solicitud HTTP GET a la URL proporcionada.
  + response: Contiene la respuesta de la solicitud, incluyendo los datos obtenidos de la API.
* return response.data;: Devuelve los datos obtenidos de la API.
  + response.data: Accede a los datos de la respuesta de la API.
* catch (error) { ... }: Si ocurre un error durante la solicitud, se captura y se maneja dentro de este bloque.
  + catch (error) { ... }: Captura el error y ejecuta el bloque de código dentro de él.
  + console.error("Error fetching countries:", error);: Muestra un mensaje de error en la consola para ayudar en la depuración.
  + return [];: Devuelve un array vacío en caso de error para asegurar que la función siempre devuelve un valor consistente.

Función getLocalCountries

Javascript:

*export const getLocalCountries = () => {*

*const savedCountries = localStorage.getItem("countries");*

*return savedCountries ? JSON.parse(savedCountries) : [];*

*};*

* export const getLocalCountries = () => { ... };: Define una función para obtener la lista de países almacenados en localStorage.
  + const savedCountries = localStorage.getItem("countries");: Obtiene el valor asociado a la clave countries desde localStorage.
    - localStorage.getItem("countries"): Obtiene el valor almacenado bajo la clave countries en localStorage.
  + return savedCountries ? JSON.parse(savedCountries) : [];: Si hay datos guardados bajo la clave countries, los analiza como JSON y los devuelve. Si no hay datos, devuelve un array vacío.
    - savedCountries ? JSON.parse(savedCountries) : []: Utiliza un operador ternario para comprobar si savedCountries tiene un valor. Si lo tiene, lo convierte de una cadena JSON a un objeto JavaScript usando JSON.parse(savedCountries). Si no, devuelve un array vacío.

Función saveLocalCountries

Javascript:

*export const saveLocalCountries = (countries) => {*

*localStorage.setItem("countries", JSON.stringify(countries));*

*};*

* export const saveLocalCountries = (countries) => { ... };: Define una función para guardar una lista de países en localStorage.
  + localStorage.setItem("countries", JSON.stringify(countries));: Guarda los datos bajo la clave countries en localStorage.
    - localStorage.setItem("countries", JSON.stringify(countries)): Utiliza localStorage.setItem para guardar los datos bajo la clave countries. Convierte el objeto countries a una cadena JSON usando JSON.stringify(countries) antes de guardarlo.

¿Qué Pasaría si...?

* Errores en la Solicitud HTTP:
  + URL Incorrecta: Si la URL de la API es incorrecta o el servidor no está disponible, la solicitud fallará y se ejecutará el bloque catch, mostrando un mensaje de error en la consola y devolviendo un array vacío.
  + Omisión de await: Si await se omite, axios.get devolverá una promesa sin esperar a que se resuelva, lo que puede causar problemas al intentar acceder a los datos de la respuesta.
* Errores en localStorage:
  + Datos Mal Formateados: Si localStorage.getItem("countries") devuelve un valor que no es una cadena JSON válida, JSON.parse lanzará un error. Este caso no está manejado en la función getLocalCountries, lo que puede causar que la aplicación falle.
  + Omisión de JSON.stringify: Si JSON.stringify(countries) no se usa en saveLocalCountries, localStorage almacenará el valor como una cadena [object Object], lo que no es útil y causará errores al intentar recuperar los datos.
* Problemas Generales:
  + Exportación Incorrecta: Si alguna de las funciones no se exporta correctamente usando export, no estarán disponibles para ser importadas y utilizadas en otros archivos, lo que puede llevar a errores de importación.

Este archivo api.js centraliza las funciones para interactuar con la API y localStorage, permitiendo obtener y almacenar datos de manera organizada y manejable.

**Explicación Detallada de “src/components/Home.jsx”:**

Importaciones

*import React from "react";*

*import { Link } from "react-router-dom";*

*import "../components/styles/main.css";*

* import React from "react";: Importa la biblioteca React. React es necesario para trabajar con JSX y para crear componentes React. Sin esta importación, el archivo no podría utilizar JSX ni las funcionalidades de React.
* import { Link } from "react-router-dom";: Importa el componente Link del paquete react-router-dom. Link se usa para crear enlaces de navegación interna en una aplicación React con enrutamiento, reemplazando el uso de etiquetas <a> que recargarían la página.
* import "../components/styles/main.css";: Importa el archivo CSS principal que contiene los estilos generales para este componente. Esto asegura que los estilos definidos en main.css se apliquen a los elementos dentro del componente Home.

Componente Home

*function Home() {*

*return (*

*<div>*

*<h1>Bienvenido a mi Página Web de Países y Capitales</h1>*

*<nav>*

*<ul>*

*<li>*

*<Link to="/countries">Lista de Países</Link>*

*</li>*

*<li>*

*<Link to="/about">Acerca de</Link>*

*</li>*

*</ul>*

*</nav>*

*</div>*

*);*

*}*

* function Home() { ... }: Define un componente funcional llamado Home. Los componentes funcionales son una forma de definir componentes en React usando funciones.
* <div>: Contenedor principal del componente Home. Todos los elementos del componente estarán dentro de este <div>.
* <h1>Bienvenido a mi Página Web de Países y Capitales</h1>: Encabezado principal de la página de inicio. Este texto se muestra en un tamaño grande para dar la bienvenida a los usuarios.
* <nav>: Contenedor de navegación que agrupa los enlaces de navegación.
  + <ul>: Lista desordenada que contiene los elementos de la navegación.
    - <li>: Elemento de la lista que contiene un enlace de navegación.
      * <Link to="/countries">Lista de Países</Link>: Enlace de navegación que lleva a la página de lista de países. to="/countries" define la ruta a la que el enlace debe navegar.
      * <Link to="/about">Acerca de</Link>: Enlace de navegación que lleva a la página de "Acerca de". to="/about" define la ruta a la que el enlace debe navegar.

Exportación

*export default Home;*

* export default Home;: Exporta el componente Home como la exportación predeterminada del módulo. Esto permite que otros archivos importen este componente usando import Home from './Home';.

¿Qué Pasaría si...?

* Faltan Importaciones:
  + React: Sin importar React, no podrías utilizar JSX, lo que resultaría en errores de sintaxis y el componente no se renderizaría.
  + Link: Sin importar Link de react-router-dom, los enlaces de navegación no funcionarían correctamente y causaría errores al intentar utilizar <Link>.
  + CSS: Sin importar el archivo CSS, los estilos definidos en main.css no se aplicarían, lo que podría resultar en una apariencia no deseada del componente.
* Errores en JSX:
  + Falta de Cierre de Etiquetas: JSX requiere que todas las etiquetas estén correctamente cerradas. Si falta una etiqueta de cierre, resultará en un error de sintaxis.
  + Errores de Enrutamiento: Si los valores to en los enlaces no coinciden con las rutas definidas en react-router-dom, los enlaces no funcionarán correctamente y podrían llevar a páginas no encontradas.

Este archivo Home.jsx define el componente de la página de inicio de la aplicación, proporcionando una bienvenida al usuario y enlaces de navegación a otras secciones importantes de la aplicación.

**Explicación Detallada de “src/components/CountryList.jsx”:**

Importaciones

*import React, { useState, useEffect } from "react";*

*import { Link, useNavigate } from "react-router-dom";*

*import { fetchCountries } from "../api";*

*import "../components/styles/CountryList.css"*

* import React, { useState, useEffect } from "react";: Importa React y los hooks useState y useEffect. useState se utiliza para manejar el estado del componente, mientras que useEffect se utiliza para manejar efectos secundarios, como la carga de datos.
* import { Link, useNavigate } from "react-router-dom";: Importa los componentes Link y useNavigate de react-router-dom. Link se usa para crear enlaces de navegación interna, y useNavigate es un hook que permite la navegación programática.
* import { fetchCountries } from "../api";: Importa la función fetchCountries del archivo api.js, que se utiliza para obtener la lista de países desde la API.
* import "../components/styles/CountryList.css": Importa el archivo CSS específico para aplicar estilos al componente CountryList.

Funciones Auxiliares

*const getLocalCountries = () => {*

*const savedCountries = localStorage.getItem("countries");*

*return savedCountries ? JSON.parse(savedCountries) : [];*

*};*

*const saveLocalCountries = (countries) => {*

*localStorage.setItem("countries", JSON.stringify(countries));*

*};*

* getLocalCountries: Función que obtiene los países guardados en el localStorage. Si existen datos, los parsea de JSON a un objeto JavaScript; de lo contrario, retorna un array vacío.
* saveLocalCountries: Función que guarda la lista de países en el localStorage, convirtiéndola a JSON antes de almacenarla.

Componente CountryList

*function CountryList() {*

*const [countries, setCountries] = useState([]);*

*const [searchTerm, setSearchTerm] = useState("");*

*const [newCountry, setNewCountry] = useState({*

*id: '',*

*nombre: '',*

*capital: '',*

*datos: {*

*poblacion: '',*

*idioma: '',*

*moneda: ''*

*}*

});

const navigate = useNavigate();

* const [countries, setCountries] = useState([]);: Define el estado countries y su función actualizadora setCountries. Inicialmente, countries es un array vacío.
* const [searchTerm, setSearchTerm] = useState("");: Define el estado searchTerm y su función actualizadora setSearchTerm. Inicialmente, searchTerm es una cadena vacía.
* const [newCountry, setNewCountry] = useState({ ... });: Define el estado newCountry y su función actualizadora setNewCountry, que maneja la información del nuevo país a agregar.
* const navigate = useNavigate();: Define la función navigate usando el hook useNavigate para permitir la navegación programática.

useEffect para Cargar Datos

*useEffect(() => {*

*const loadCountries = async () => {*

*const apiCountries = await fetchCountries();*

*const localCountries = getLocalCountries();*

*const allCountries = [*

*...apiCountries,*

*...localCountries.filter(*

*(localCountry) => !apiCountries.some(apiCountry => apiCountry.id === localCountry.id)*

*)*

*];*

*setCountries(allCountries);*

*};*

*loadCountries();*

*}, []);*

* useEffect(() => { ... }, []);: Hook que ejecuta la función pasada como argumento cuando el componente se monta. La dependencia vacía [] asegura que se ejecute solo una vez.
* const loadCountries = async () => { ... };: Función asíncrona que carga los países desde la API y localStorage.
  + const apiCountries = await fetchCountries();: Llama a fetchCountries para obtener países desde la API.
  + const localCountries = getLocalCountries();: Llama a getLocalCountries para obtener países desde el localStorage.
  + const allCountries = [ ... ];: Combina las dos listas de países, filtrando duplicados basados en el id.
  + setCountries(allCountries);: Actualiza el estado countries con la lista combinada de países.

Filtrado de Países

*const filteredCountries = countries.filter(*

*(country) =>*

*country.nombre.toLowerCase().includes(searchTerm.toLowerCase()) ||*

*country.capital.toLowerCase().includes(searchTerm.toLowerCase())*

*);*

* filteredCountries: Filtra los países basados en el término de búsqueda searchTerm. Compara los nombres y capitales de los países con el término de búsqueda (ignorando mayúsculas y minúsculas).

Manejo de Nuevos Países

*const handleAddCountry = () => {*

*const newCountries = [...countries, { ...newCountry, id: Date.now() }];*

*setCountries(newCountries);*

*saveLocalCountries(newCountries);*

*setNewCountry({*

*id: '',*

*nombre: '',*

*capital: '',*

*datos: {*

*poblacion: '',*

*idioma: '',*

*moneda: ''*

*}*

*});*

*};*

* handleAddCountry: Función que maneja la adición de un nuevo país.
  + const newCountries = [...countries, { ...newCountry, id: Date.now() }];: Crea una nueva lista de países incluyendo el nuevo país con un id único generado por Date.now().
  + setCountries(newCountries);: Actualiza el estado countries con la nueva lista.
  + saveLocalCountries(newCountries);: Guarda la nueva lista de países en el localStorage.
  + setNewCountry({ ... });: Resetea el formulario de nuevo país.

Manejo de Eliminación de Países

*const handleDeleteCountry = (id) => {*

*const newCountries = countries.filter((country) => country.id !== id);*

*setCountries(newCountries);*

*saveLocalCountries(newCountries);*

*};*

* handleDeleteCountry: Función que maneja la eliminación de un país.
  + const newCountries = countries.filter((country) => country.id !== id);: Filtra los países excluyendo el país con el id proporcionado.
  + setCountries(newCountries);: Actualiza el estado countries con la lista filtrada.
  + saveLocalCountries(newCountries);: Guarda la lista filtrada en el localStorage.

Renderización

*return (*

*<div>*

*<button onClick={() => navigate("/")}>Ir hacia atrás</button>{" "}*

*<input*

*type="text"*

*value={searchTerm}*

*onChange={(e) => setSearchTerm(e.target.value)}*

*placeholder="Buscar país"*

*/>*

*<h2>Añadir Nuevo País</h2>*

*<input*

*type="text"*

*placeholder="Nombre"*

*value={newCountry.nombre}*

*onChange={(e) => setNewCountry({ ...newCountry, nombre: e.target.value })}*

*/>*

*<input*

*type="text"*

*placeholder="Capital"*

*value={newCountry.capital}*

*onChange={(e) => setNewCountry({ ...newCountry, capital: e.target.value })}*

*/>*

*<input*

*type="text"*

*placeholder="Población"*

*value={newCountry.datos.poblacion}*

*onChange={(e) => setNewCountry({*

*...newCountry,*

*datos: { ...newCountry.datos, poblacion: e.target.value }*

*})}*

*/>*

*<input*

*type="text"*

*placeholder="Idioma"*

*value={newCountry.datos.idioma}*

*onChange={(e) => setNewCountry({*

*...newCountry,*

*datos: { ...newCountry.datos, idioma: e.target.value }*

*})}*

*/>*

*<input*

*type="text"*

*placeholder="Moneda"*

*value={newCountry.datos.moneda}*

*onChange={(e) => setNewCountry({*

*...newCountry,*

*datos: { ...newCountry.datos, moneda: e.target.value }*

*})}*

*/>*

*<button onClick={handleAddCountry}>Añadir País</button>*

*<ul>*

*{filteredCountries.map((country, index) => (*

*<li key={index}>*

*<Link to={`/countries/${country.id}`}>{country.nombre}</Link>*

*<Link to={`/countries/edit/${country.id}`} className="btn edit-btn">Editar</Link>*

*<button onClick={() => handleDeleteCountry(country.id)} className="btn delete-button">Eliminar</button>*

*</li>*

*))}*

*</ul>*

*</div>*

*);*

* Botón "Ir hacia atrás": Navega a la página de inicio al hacer clic.
* Campo de búsqueda: Permite al usuario buscar países por nombre o capital.
* Formulario para añadir nuevos países: Incluye campos para nombre, capital, población, idioma y moneda, y un botón para añadir el nuevo país.
* Lista de países: Muestra los países filtrados con enlaces para ver y editar cada país, y un botón para eliminar países.

**Explicación Detallada de “src/components/CountryDetail.jsx”:**

Importaciones

*import React, { useState, useEffect } from "react";*

*import { useParams, useNavigate } from "react-router-dom";*

*import { fetchCountries } from "../api";*

*import "../components/styles/CountryDetail.css";*

* import React, { useState, useEffect } from "react";: Importa React junto con los hooks useState y useEffect. useState se usa para manejar el estado del componente, mientras que useEffect se utiliza para realizar efectos secundarios, como cargar datos cuando el componente se monta o se actualiza.
* import { useParams, useNavigate } from "react-router-dom";: Importa useParams y useNavigate de react-router-dom. useParams permite acceder a los parámetros de la URL, mientras que useNavigate es un hook que permite la navegación programática.
* import { fetchCountries } from "../api";: Importa la función fetchCountries desde el archivo api.js, que se utiliza para obtener la lista de países desde una API externa.
* import "../components/styles/CountryDetail.css";: Importa el archivo CSS específico para aplicar estilos al componente CountryDetail.

Funciones Auxiliares

*const getLocalCountries = () => {*

*const savedCountries = localStorage.getItem("countries");*

*return savedCountries ? JSON.parse(savedCountries) : [];*

*};*

* const getLocalCountries = () => { ... }: Función que obtiene los países almacenados en localStorage. Si existen datos, los convierte de JSON a un objeto JavaScript con JSON.parse. Si no hay datos guardados, devuelve un array vacío.

Componente CountryDetail

*function CountryDetail() {*

*const { id } = useParams(); // Obtener el ID del país desde la URL*

*const [country, setCountry] = useState(null);*

*const navigate = useNavigate(); // Hook para la navegación*

* const { id } = useParams();: Utiliza el hook useParams para obtener el ID del país desde la URL. id es un parámetro dinámico definido en la ruta de la aplicación.
* const [country, setCountry] = useState(null);: Define el estado country para almacenar la información del país seleccionado. Inicialmente, country es null.
* const navigate = useNavigate();: Define la función navigate usando el hook useNavigate, permitiendo la navegación programática dentro de la aplicación.

Carga de Datos

*useEffect(() => {*

*const loadCountry = async () => {*

*const apiCountries = await fetchCountries();*

*const localCountries = getLocalCountries();*

const allCountries = [...apiCountries, ...localCountries];

// Agregar console.log para depuración

console.log("All countries:", allCountries);

const countryData = allCountries.find((country) => country.id === parseInt(id));

// Agregar console.log para depuración

console.log("Selected country ID:", id);

console.log("Selected country data:", countryData);

setCountry(countryData);

};

loadCountry();

}, [id]);

* useEffect(() => { ... }, [id]);: Hook que ejecuta la función pasada como argumento cuando el componente se monta o el valor de id cambia. Esto asegura que los datos del país se carguen correctamente cada vez que se accede a un país diferente.
* const loadCountry = async () => { ... }: Función asíncrona para cargar los datos del país.
* const apiCountries = await fetchCountries();: Llama a fetchCountries para obtener la lista de países desde la API.
* const localCountries = getLocalCountries();: Llama a getLocalCountries para obtener la lista de países desde el localStorage.
* const allCountries = [...apiCountries, ...localCountries];: Combina los países de la API con los del localStorage. Esto incluye todos los países en una sola lista.
* console.log("All countries:", allCountries);: Imprime la lista combinada de países en la consola para depuración.
* const countryData = allCountries.find((country) => country.id === parseInt(id));: Busca en la lista combinada el país cuyo id coincide con el parámetro id obtenido de la URL.
* console.log("Selected country ID:", id);: Imprime el ID del país seleccionado en la consola para depuración.
* console.log("Selected country data:", countryData);: Imprime los datos del país seleccionado en la consola para depuración.
* setCountry(countryData);: Actualiza el estado country con los datos del país encontrado.

Renderización

*if (!country) return <div>Loading...</div>; // Mostrar "Cargando..." mientras se obtienen los datos*

*return (*

*<div>*

*<button onClick={() => navigate(-1)}>Ir hacia atrás</button>{" "}*

*{/\* Botón para volver a la página anterior \*/}*

*<h1>{country.nombre}</h1>*

*<p>Capital: {country.capital}</p>*

*<p>Población: {country.datos.poblacion}</p>*

*<p>Idioma: {country.datos.idioma}</p>*

*<p>Moneda: {country.datos.moneda}</p>*

*</div>*

*);*

* if (!country) return <div>Loading...</div>;: Verifica si country es null. Si es así, muestra un mensaje de "Cargando..." mientras se obtienen los datos.
* <button onClick={() => navigate(-1)}>Ir hacia atrás</button>: Botón que permite al usuario volver a la página anterior utilizando la función navigate(-1).
* <h1>{country.nombre}</h1>: Muestra el nombre del país.
* <p>Capital: {country.capital}</p>: Muestra la capital del país.
* <p>Población: {country.datos.poblacion}</p>: Muestra la población del país.
* <p>Idioma: {country.datos.idioma}</p>: Muestra el idioma del país.
* <p>Moneda: {country.datos.moneda}</p>: Muestra la moneda del país.

**Explicación Detallada de “src/components/EditCountry.jsx”:**

Importaciones

*import React, { useState, useEffect } from "react";*

*import { useParams, useNavigate } from "react-router-dom";*

*import { fetchCountries, getLocalCountries, saveLocalCountries } from "../api"; Corrección de importaciones*

*import "../components/styles/main.css"; // Importar CSS general*

*import "../components/styles/EditCountry.css"; // Importar CSS específico para el componente*

* import React, { useState, useEffect } from "react";: Importa React junto con los hooks useState y useEffect para manejar el estado y los efectos secundarios en el componente.
* import { useParams, useNavigate } from "react-router-dom";: Importa los hooks useParams y useNavigate de react-router-dom para manejar los parámetros de la URL y la navegación programática.
* import { fetchCountries, getLocalCountries, saveLocalCountries } from "../api";: Importa funciones desde api.js para obtener y guardar datos de países.
* import "../components/styles/main.css"; y import "../components/styles/EditCountry.css";: Importa archivos CSS para aplicar estilos generales y específicos al componente.

Componente EditCountry

*function EditCountry() {*

*const { id } = useParams(); // Obtener el ID del país desde la URL*

*const [country, setCountry] = useState(null);*

*const [editCountry, setEditCountry] = useState({ nombre: '', capital: '', datos: { poblacion: '', idioma: '', moneda: '' } }); // Estado para editar país*

*const navigate = useNavigate(); // Hook para la navegación*

* function EditCountry() { ... }: Define el componente funcional EditCountry.
* const { id } = useParams();: Obtiene el ID del país desde los parámetros de la URL.
* const [country, setCountry] = useState(null);: Estado para almacenar los datos del país cargado.
* const [editCountry, setEditCountry] = useState({ nombre: '', capital: '', datos: { poblacion: '', idioma: '', moneda: '' } });: Estado para almacenar los datos del país que se está editando.
* const navigate = useNavigate();: Hook para obtener la función de navegación.

Efecto para Cargar Datos

*useEffect(() => {*

*const loadCountry = async () => {*

*const apiCountries = await fetchCountries();*

*const localCountries = getLocalCountries(); // Corrección de obtener países locales*

*const allCountries = [...apiCountries, ...localCountries]; // Combina países de la API y locales*

*const countryData = allCountries.find((country) => country.id === parseInt(id));*

*setCountry(countryData);*

*if (countryData) {*

*setEditCountry(countryData);*

*}*

*};*

*loadCountry();*

*}, [id]);*

* useEffect(() => { ... }, [id]);: Hook que se ejecuta cuando el componente se monta o el ID cambia. Carga los datos del país utilizando una función asíncrona.
* const loadCountry = async () => { ... }: Función asíncrona para cargar los datos del país.
* const apiCountries = await fetchCountries();: Obtiene los países de la API.
* const localCountries = getLocalCountries();: Obtiene los países almacenados localmente.
* const allCountries = [...apiCountries, ...localCountries];: Combina los países de la API y los locales.
* const countryData = allCountries.find((country) => country.id === parseInt(id));: Busca el país correspondiente al ID en la lista combinada.
* setCountry(countryData); y if (countryData) { setEditCountry(countryData); }: Actualiza el estado con los datos del país encontrado.

Manejo de Actualización del País

*const handleUpdateCountry = async () => { // Día 3: Marcado como 'async'*

*const apiCountries = await fetchCountries(); // Re-obtener países de la API para evitar desincronización*

*const localCountries = getLocalCountries(); // Obtener países locales*

*const updatedCountries = localCountries.map((c) =>*

*c.id === parseInt(id) ? { ...editCountry, id: parseInt(id) } : c*

*);*

*saveLocalCountries(updatedCountries);*

*navigate(`/countries/${id}`); // Redirigir a la vista de detalles del país*

*};*

* const handleUpdateCountry = async () => { ... }: Función asíncrona que maneja la actualización de los datos del país.
* const apiCountries = await fetchCountries();: Vuelve a obtener los países de la API.
* const localCountries = getLocalCountries();: Obtiene los países locales.
* const updatedCountries = localCountries.map((c) => ...);: Actualiza la lista de países locales con los datos editados.
* saveLocalCountries(updatedCountries);: Guarda la lista actualizada de países locales.
* navigate(/countries/${id});: Redirige a la vista de detalles del país editado.

Renderizado del Componente

*if (!country) return <div>Loading...</div>; // Mostrar "Cargando..." mientras se obtienen los datos*

*return (*

*<div>*

*<button onClick={() => navigate(-1)}>Ir hacia atrás</button> {/\* Botón para volver a la página anterior \*/}*

*<h1>Editar País</h1>*

*<input*

*type="text"*

*placeholder="Nombre"*

*value={editCountry.nombre}*

*onChange={(e) => setEditCountry({ ...editCountry, nombre: e.target.value })}*

*/>*

*<input*

*type="text"*

*placeholder="Capital"*

*value={editCountry.capital}*

*onChange={(e) => setEditCountry({ ...editCountry, capital: e.target.value })}*

*/>*

*<input*

*type="text"*

*placeholder="Población"*

*value={editCountry.datos.poblacion}*

*onChange={(e) => setEditCountry({ ...editCountry, datos: { ...editCountry.datos, poblacion: e.target.value } })}*

*/>*

*<input*

*type="text"*

*placeholder="Idioma"*

*value={editCountry.datos.idioma}*

*onChange={(e) => setEditCountry({ ...editCountry, datos: { ...editCountry.datos, idioma: e.target.value } })}*

*/>*

*<input*

*type="text"*

*placeholder="Moneda"*

*value={editCountry.datos.moneda}*

*onChange={(e) => setEditCountry({ ...editCountry, datos: { ...editCountry.datos, moneda: e.target.value } })}*

*/>*

*<button onClick={handleUpdateCountry}>Actualizar País</button> {/\* Botón para guardar los cambios \*/}*

*</div>*

*);*

*}*

* if (!country) return <div>Loading...</div>;: Muestra un mensaje de carga mientras se obtienen los datos del país.
* <button onClick={() => navigate(-1)}>Ir hacia atrás</button>: Botón que navega hacia la página anterior.
* <input type="text" ... />: Campos de entrada para editar el nombre, capital, población, idioma y moneda del país.
* <button onClick={handleUpdateCountry}>Actualizar País</button>: Botón que guarda los cambios realizados en los datos del país.

Exportación del Componente

*export default EditCountry;*

* export default EditCountry;: Exporta el componente EditCountry para que pueda ser utilizado en otras partes de la aplicación.

Este componente permite a los usuarios editar la información de un país. Se carga el país desde la API y el almacenamiento local, se actualizan los datos editados, y se guarda la nueva información en el almacenamiento local.

**Explicación Detallada de “src/components/About.jsx”:**

Importaciones

*import React from "react";*

*import { useNavigate } from "react-router-dom"; // Día 2: Para el botón "atrás".*

*import "../components/styles/main.css"; // Importar CSS general*

* import React from "react";: Importa la biblioteca React para utilizarla en la definición del componente.
* import { useNavigate } from "react-router-dom";: Importa el hook useNavigate de react-router-dom, que se utiliza para manejar la navegación programática.
* import "../components/styles/main.css";: Importa el archivo CSS general para aplicar estilos al componente.

Componente About

*function About() {*

*const navigate = useNavigate(); // Día 2: Hook para la navegación*

*return (*

*<div>*

*<button onClick={() => navigate("/")}>Ir hacia atrás</button>{" "}*

*{/\* Botón para volver a la página de inicio \*/}*

*<h1>Acerca del Proyecto</h1>*

*<p>*

*A continuación, presento un resumen detallado del progreso realizado en*

*el desarrollo del proyecto, siguiendo los requisitos establecidos y las*

*tareas realizadas cada día, junto con la ubicación de los archivos*

*correspondientes:*

*</p>*

* function About() { ... }: Define el componente funcional About.
* const navigate = useNavigate();: Usa el hook useNavigate para obtener la función navigate, que se utilizará para cambiar la ruta programáticamente.
* <button onClick={() => navigate("/")}>Ir hacia atrás</button>: Botón que, al ser clickeado, navega a la página de inicio ("/").
* <h1>Acerca del Proyecto</h1>: Encabezado principal del componente que describe el propósito del contenido.
* <p> ... </p>: Párrafo que introduce el resumen detallado del progreso del proyecto.

**Explicación Detallada de “src/components/NotFound.jsx”:**

Importaciones:

*import React from "react";*

*import "../components/styles/main.css";*

*import "../components/styles/NotFound.css";*

* import React from "react";: Este import es esencial para trabajar con React. Permite usar JSX (la sintaxis similar a HTML que React utiliza) y crear componentes.
* import "../components/styles/main.css";: Importa el archivo de estilos globales. Los estilos en main.css se aplican a todo el proyecto o a varios componentes.
* import "../components/styles/NotFound.css";: Importa el archivo de estilos específicos para el componente NotFound. Permite aplicar estilos personalizados a este componente en particular.

Definición del Componente:

*function NotFound() {*

*return (*

*<div>*

*...*

*</div>*

*);*

*}*

* function NotFound() { ... }: Define el componente funcional NotFound. Los componentes funcionales son funciones de JavaScript que devuelven JSX.

Renderizado del Componente:

*<h1>404 - No Encontrado</h1>*

*<p>La página que estás buscando no existe.</p>*

*<br />*

*<img*

*src="https://giphy.com/clips/ThisIsMashed-animation-gaming-cartoon-1hGu7ketlpuuWjwvMA"*

*alt="Not Found"*

*className="notfound-gif"*

*/>*

* <h1>404 - No Encontrado</h1>: Muestra un mensaje de error 404 usando un encabezado <h1>. Este encabezado tiene un impacto visual significativo y es adecuado para mensajes de error.
* <p>La página que estás buscando no existe.</p>: Proporciona un mensaje en un párrafo <p>. Ayuda a informar al usuario que la página solicitada no está disponible.
* <br />: Un salto de línea HTML que se usa para agregar un espacio entre el mensaje y la imagen, mejorando la separación visual.
* <img src="..." alt="Not Found" className="notfound-gif" />: Muestra una imagen GIF en la página de error.
  + src="https://giphy.com/clips/ThisIsMashed-animation-gaming-cartoon-1hGu7ketlpuuWjwvMA": La URL de la imagen GIF que se muestra. Es un recurso visual que proporciona un toque lúdico o informativo a la página de error.
  + alt="Not Found": Texto alternativo para la imagen. Es importante para la accesibilidad, describiendo la imagen para aquellos que no pueden verla.
  + className="notfound-gif": Aplica estilos específicos definidos en NotFound.css a la imagen. Esto puede incluir tamaños, márgenes, o cualquier otro estilo visual.

Exportación del Componente:

*export default NotFound;*

* export default NotFound;: Exporta el componente NotFound para que pueda ser importado y utilizado en otras partes de la aplicación, como en la configuración de rutas para mostrar una página 404 cuando una ruta no está definida.