Documentación del Código

Este código es una aplicación React que carga, muestra y permite buscar datos en un formato JSON con paginación. A continuación, se explica cada sección y función de la aplicación:

1. Dependencias y Estado Inicial

javascript

Copiar código

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

import axios from "axios";

import "bootstrap/dist/css/bootstrap.min.css";

import "bootstrap/dist/js/bootstrap.bundle.min.js";
```

React: La librería principal para crear la interfaz de usuario.

axios: Se usa para hacer solicitudes HTTP y obtener datos desde un servidor.

Bootstrap: Utilizado para aplicar estilos y componentes de interfaz de usuario de manera rápida y consistente.

2. Estado Inicial de la Aplicación

javascript

Copiar código

```
const [data, setData] = useState([]);  // Datos recibidos desde el backend

const [currentPage, setCurrentPage] = useState(1);  // Página actual

const itemsPerPage = 10;  // Número de elementos por página

const [searchTerm, setSearchTerm] = useState("");  // Término de búsqueda
```

data: Almacena los datos que se obtienen de la API.

currentPage: Controla la página actual para la paginación.

itemsPerPage: Define la cantidad de elementos que se mostrarán por página (10 por defecto).

searchTerm: El término que el usuario ingresa para buscar dentro de los datos.

3. Efecto de Carga de Datos

javascript

Copiar código

```
useEffect(() => {
```

```
  axios
    .get("http://localhost:5000/data")
    .then((response) => setData(response.data.children || []))
    .catch((error) => console.error("Error fetching data:", error));
}, []);
```

useEffect: Se utiliza para realizar la solicitud HTTP cuando el componente se monta. Se hace una petición GET a la API local para obtener los datos.

setData: Actualiza el estado data con la respuesta de la API. Si no se encuentra la propiedad children en los datos, se establece un arreglo vacío.

4. Función de Búsqueda

javascript

Copiar código

```
const handleSearch = (event) => {
  setSearchTerm(event.target.value);
};
```

Esta función maneja el cambio en el campo de búsqueda y actualiza el estado searchTerm con el valor ingresado por el usuario.

5. Filtrado de Datos

javascript

Copiar código

```
const filterData = () => {
  if (!searchTerm) return data;
  return data.filter((item) =>
    JSON.stringify(item).toLowerCase().includes(searchTerm.toLowerCase())
  );
};
```

filterData: Filtra los datos de acuerdo con el término de búsqueda. Si no se ingresa un término de búsqueda, se retornan todos los datos. La búsqueda se realiza en los datos convertidos a una cadena JSON.

6. Resaltar el Término de Búsqueda

javascript

Copiar código

```
const highlightText = (text) => {
  if (!text) return text;
  const regex = new RegExp(`(${searchTerm})`, "gi");
  return text.split(regex).map((part, index) =>
    part.toLowerCase() === searchTerm.toLowerCase() ? (
      <span key={index} style={{ backgroundColor: 'yellow', fontWeight: 'bold' }}>{part}</span>
    ) : (
      part
    )
  );
};
```

highlightText: Resalta el término de búsqueda dentro del texto. Si el término de búsqueda se encuentra dentro del texto, se envuelve con una etiqueta con un fondo amarillo y texto en negrita.

7. Renderizado de la Metadata del Documento

javascript

Copiar código

```
const renderDocumentMetadata = (metadata) => {
  return (
    <div className="mb-4">
      <h5 className="text-primary">Metadata del Documento</h5>
      <div>
        {metadata.year && <p><strong>Año:</strong> {highlightText(metadata.year)}</p>}
        {metadata.expeditionDate && <p><strong>Fecha de Expedición:</strong> {highlightText(metadata.expeditionDate)}</p>}
        {metadata.publishDate && <p><strong>Fecha de Publicación:</strong> {highlightText(metadata.publishDate)}</p>}
        {metadata.isParsed && <p><strong>¿Está Parseado?:</strong> {highlightText(metadata.isParsed ? "Sí" : "No")}</p>}
        {metadata.name && <p><strong>Nombre:</strong> {highlightText(metadata.name)}</p>}
```

```
      {metadata.description && <p><strong>Descripción:</strong>
{highlightText(metadata.description)}</p>}
    </div>
  </div>
 );
};
```

Esta función recibe los metadatos de un documento y los muestra, aplicando el resaltado del término de búsqueda a cada propiedad.

8. Renderizado de Contenido de un Bloque

javascript

Copiar código

```
const renderBlockContent = (block) => {
 return (
  <div className="mb-4">
   <div>
    {block.GUID && <p><strong>GUID:</strong> {highlightText(block.GUID)}</p>}
    {block.validDate && <p><strong>Fecha Válida:</strong> {highlightText(block.validDate)}</p>}
    {block.children && block.children.length > 0 && (
     <div className="mt-3">
      {block.children.map((child, idx) => (
       <p key={idx} className="mb-2">
        {highlightText(child.text)}
       </p>
      ))}
     </div>
    )}
   </div>
  </div>
 );
};
```

renderBlockContent: Muestra los datos de un bloque, incluyendo sus propiedades como GUID y validDate. Si el bloque tiene hijos, también se muestran.

9. Renderizado de Páginas de Datos

javascript

Copiar código

```
const renderPage = (data) => {
  return data.map((node, index) => {
    if (node.tag === "documentMetadata") {
      return renderDocumentMetadata(node.attributes);
    } else if (node.tag === "block") {
      return renderBlockContent(node);
    } else {
      return null;
    }
  });
};
```

renderPage: Itera sobre los datos filtrados y renderiza diferentes tipos de contenido, dependiendo de las etiquetas (tags) de los nodos (por ejemplo, documentMetadata o block).

10. Paginación

javascript

Copiar código

```
const handlePageChange = (newPage) => {
  setCurrentPage(newPage);
};
```

handlePageChange: Esta función permite cambiar la página actual cuando el usuario hace clic en los botones de paginación (anterior o siguiente).

11. Cálculo de Índices de Paginación

javascript

Copiar código

```
const startIndex = (currentPage - 1) * itemsPerPage;
```

const currentData = filterData().slice(startIndex, startIndex + itemsPerPage);

Se calculan los índices de inicio y fin para los datos de la página actual. El slice es utilizado para seleccionar solo los elementos que corresponden a la página actual.

12. Renderizado Final

javascript

Copiar código

```
return (
  <div className="container-fluid px-3 mt-4" style={{ minHeight: '100vh', display: 'flex', flexDirection: 'column' }}>
    <div className="card shadow-lg border-0" style={{ flex: '1 0 auto' }}>
      <div className="card-header bg-primary text-white d-flex justify-content-between align-items-center">
        <h1 className="h4 mb-0 text-center" style={{ fontSize: '18px' }}>Visualizador de JSON con Paginación</h1>
        <input
          type="text"
          placeholder="Buscar..."
          className="form-control"
          style={{ width: '300px' }}
          value={searchTerm}
          onChange={handleSearch}
        />
      </div>
      <div className="card-body" style={{ fontSize: '14px', overflowY: 'auto', maxHeight: 'calc(100vh - 200px)', padding: '20px' }}>
        {data.length > 0 ? renderPage(currentData) : (
          <div className="d-flex justify-content-center align-items-center">
            <div className="spinner-border text-primary" role="status">
              <span className="visually-hidden">Cargando...</span>
            </div>
          </div>
```

```
      )}
    </div>
    <div className="card-footer d-flex justify-content-between" style={{ backgroundColor: '#f8f9fa', padding: '10px 20px' }}>
      <button
        className="btn btn-outline-primary"
        disabled={currentPage === 1}
        onClick={() => handlePageChange(currentPage - 1)}
      >
        Anterior
      </button>
      <span className="text-muted" style={{ fontSize: '14px' }}>
        Página {currentPage} de {Math.ceil(filterData().length / itemsPerPage)}
      </span>
      <button
        className="btn btn-outline-primary"
        disabled={currentPage === Math.ceil(filterData().length / itemsPerPage)}
        onClick={() => handlePageChange(currentPage + 1)}
      >
        Siguiente
      </button>
    </div>
  </div>
 </div>
);
```

Renderizado final: Estructura la interfaz utilizando Bootstrap para mostrar el contenido de los datos. También incluye el formulario de búsqueda y los controles de paginación. Se muestra un spinner de carga mientras se obtienen los datos.