```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function ListSources() {
 const [globalSources, setGlobalSources] = useState([]);
 const [privateSources, setPrivateSources] = useState([]);
 const [selectedSource, setSelectedSource] = useState(null);
 const [documents, setDocuments] = useState([]);
 const [message, setMessage] = useState('');
 const [isLoading, setIsLoading] = useState(false);
 const [selectedFile, setSelectedFile] = useState(null);
 const [uploadProgress, setUploadProgress] = useState(0);

 const refreshData = async () => {
  await fetchSources();
  if (selectedSource) {
   await fetchDocuments(selectedSource);
  }
 };

 useEffect(() => {
  refreshData();
 }, []);

 const fetchSources = async () => {
  setIsLoading(true);
  setMessage('');

  try {
   const response = await axios.get('/chatbot1/list-sources/', {
    headers: {
     'Content-Type': 'application/json',
     'X-CSRFToken': getCookie('csrftoken'),
    },
    withCredentials: true
   });

   if (response.status === 200) {
```

```
      setGlobalSources(response.data.global_sources || []);
      setPrivateSources(response.data.private_sources || []);
    } else {
      setMessage('An error occurred while fetching the sources.');
    }
  } catch (error) {
    console.error('Error during source fetching:', error);
    setMessage(error.response?.data?.error || 'An error occurred while fetching the
sources.');
  } finally {
    setIsLoading(false);
  }
 };

 const fetchDocuments = async (source) => {
  setIsLoading(true);
  setMessage('');
  setSelectedSource(source);

  try {
    const response = await axios.get(` /chatbot1/list-documents/${source}/` , {
      headers: {
        'Content-Type': 'application/json',
        'X-CSRFToken': getCookie('csrftoken'),
      },
      withCredentials: true
    });

    if (response.status === 200 && response.data.documents) {
      setDocuments(response.data.documents.documents.map(doc => ({
        ...doc,
        path: doc.path || ` /chatbot1/media/documents/${source}/${doc.filename}`
      })));
      setMessage(response.data.documents.documents.length === 0 ? 'No documents
found.' : '');
    } else {
      setMessage('An error occurred while fetching the documents.');
    }
```

```
  } catch (error) {
    console.error('Error during document fetching:', error);
    setMessage(error.response?.data?.error || 'An error occurred while fetching the
documents.');
  } finally {
    setIsLoading(false);
  }
 };

 const getCookie = (name) => {
  let cookieValue = null;
  if (document.cookie && document.cookie !== '') {
    const cookies = document.cookie.split(';');
    for (let i = 0; i < cookies.length; i++) {
      const cookie = cookies[i].trim();
      if (cookie.substring(0, name.length + 1) === `${name}=`) {
        cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
        break;
      }
    }
  }
  return cookieValue;
 };

 const handleSyncSource = async () => {
  try {
    const response = await axios.post(`/chatbot1/sync-source/${selectedSource}/`, {}, {
      headers: {
        'Content-Type': 'application/json',
        'X-CSRFToken': getCookie('csrftoken'),
      },
      withCredentials: true
    });
    alert(response.data.message);
    fetchDocuments(selectedSource);
  } catch (error) {
    console.error('Error syncing source:', error);
    console.error('Error details:', error.response?.data);
```

```
    alert(error.response?.data?.error || 'An error occurred during synchronization. Please check the server logs for more details.');
   }
  };

  const handleFileChange = (event) => {
   setSelectedFile(event.target.files[0]);
  };

  const handleUpload = async () => {
  if (!selectedFile || !selectedSource) {
   setMessage('Please select a file and a source to upload.');
   return;
  }

  setIsLoading(true);
  setUploadProgress(0);
  const formData = new FormData();
  formData.append('file', selectedFile);

  try {
   const response = await axios.post(`/chatbot1/upload-document/${selectedSource}/`, formData, {
    headers: {
     'Content-Type': 'multipart/form-data',
     'X-CSRFToken': getCookie('csrftoken'),
    },
    withCredentials: true,
    onUploadProgress: (progressEvent) => {
     const percentCompleted = Math.round((progressEvent.loaded * 100) / progressEvent.total);
     setUploadProgress(percentCompleted);
    }
   });

   console.log('Full API Response:', response.data);

   if (response.data.status === 'PENDING') {
```

```
    setMessage(`Upload initiated. Task ID: ${response.data.task_id}. Checking status...`);
    pollUploadStatus(selectedSource, response.data.task_id);
   } else {
    setMessage(`Status: ${response.data.status}\nMessage:
${response.data.message}\nFull Response: ${JSON.stringify(response.data.full_response,
null, 2)}`);
    if (response.data.status === 'SUCCESS') {
     setSelectedFile(null);
     await fetchDocuments(selectedSource);
    }
   }
  } catch (error) {
   console.error('Error uploading document:', error);
   setMessage(`Error: ${error.response?.data?.message || 'An error occurred while
uploading the document.'}\nFull Error: ${JSON.stringify(error.response?.data, null, 2)}`);
  } finally {
   setIsLoading(false);
   setUploadProgress(0);
  }
};

const pollUploadStatus = async (source, taskId, maxAttempts = 10) => {
 for (let i = 0; i < maxAttempts; i++) {
  try {
   const response = await axios.get(`/chatbot1/check-upload-
status/${source}/${taskId}/`, {
    headers: {
     'X-CSRFToken': getCookie('csrftoken'),
    },
    withCredentials: true
   });

   console.log('Poll response:', response.data);

   if (response.data.status === 'SUCCESS') {
    setMessage(`Upload completed successfully.\nFull Response:
${JSON.stringify(response.data.full_response, null, 2)}`);
    await fetchDocuments(source);
```

```
      return;
    } else if (response.data.status === 'ERROR') {
      setMessage(`Upload failed.\nError: ${response.data.message}\nFull Response:
${JSON.stringify(response.data.full_response, null, 2)}`);
      return;
    }

    setMessage(`Upload still in progress. Checking again in 5 seconds...\nCurrent status:
${response.data.status}`);
    await new Promise(resolve => setTimeout(resolve, 5000));
   } catch (error) {
    console.error('Error polling upload status:', error);
    setMessage(`Error checking upload status: ${error.message}`);
    return;
   }
  }

  setMessage('Upload status check timed out. The upload may still be in progress.');
};

  const renderSourceList = (sources, title) => (
   <div>
    <h2 style={{ color: '#444', marginTop: '20px' }}>{title}</h2>
    {sources.length > 0 ? (
     <ul style={{ listStyleType: 'none', padding: 0 }}>
      {sources.map((source) => (
       <li key={source} style={{ marginBottom: '10px', border: '1px solid #ccc', padding:
'10px', borderRadius: '4px' }}>
        <h3 style={{ margin: '0' }}>
         <a href="#" onClick={(e) => { e.preventDefault(); fetchDocuments(source); }} style={{
color: '#007bff', textDecoration: 'none' }}>
          {source}
         </a>
        </h3>
       </li>
      ))}
     </ul>
    ) : (
```

```
      <p>No {title.toLowerCase()} found.</p>
    )}
  </div>
 );

 const renderDocuments = () => (
  <div>
   <h2 style={{ color: '#444', marginTop: '20px' }}>Documents in {selectedSource}</h2>
   {documents.length > 0 ? (
    <>
     <p>Total documents: {documents.length}</p>
     <ul style={{ listStyleType: 'none', padding: 0 }}>
      {documents.map((document, index) => (
       <li key={document.id || index} style={{ marginBottom: '10px', border: '1px solid #ccc',
padding: '10px', borderRadius: '4px' }}>
        <h4 style={{ margin: '0' }}>{document.filename}</h4>
        {document.path && (
         <p style={{ margin: '5px 0 0', fontSize: '0.9em', color: '#666' }}>
          Path: <a href={document.path} target="_blank" rel="noopener noreferrer" style={{
color: '#007bff' }}>{document.path}</a>
         </p>
        )}
       </li>
      ))}
     </ul>
    </>
   ) : (
    <p>No documents available in this source.</p>
   )}

   <div style={{ marginTop: '20px' }}>
    <input type="file" onChange={handleFileChange} />
    <button onClick={handleUpload} disabled={isLoading || !selectedFile} style={{
marginLeft: '10px', padding: '10px', backgroundColor: '#28a745', color: 'white', border:
'none', borderRadius: '4px', cursor: 'pointer' }}>
     {isLoading ? 'Uploading...' : 'Upload Document'}
    </button>
   </div>
```

```
    {uploadProgress > 0 && uploadProgress < 100 && (
      <div style={{ marginTop: '10px' }}>
        <progress value={uploadProgress} max="100"></progress>
        <span>{uploadProgress}%</span>
      </div>
    )}

    <button onClick={() => setSelectedSource(null)} style={{ marginTop: '20px', padding:
'10px', backgroundColor: '#007bff', color: 'white', border: 'none', borderRadius: '4px', cursor:
'pointer', marginRight: '10px' }}>
      Back to Sources
    </button>
    <button onClick={handleSyncSource} style={{ marginTop: '20px', padding: '10px',
backgroundColor: '#28a745', color: 'white', border: 'none', borderRadius: '4px', cursor:
'pointer' }}>
      Sync Source
    </button>
  </div>
 );

 return (
  <div style={{ padding: '20px', backgroundColor: 'white', maxWidth: '800px', margin: '0 auto'
}}>
    <h1 style={{ textAlign: 'center', color: '#333' }}>List of Sources</h1>
    {isLoading ? (
      <p style={{ textAlign: 'center' }}>Loading...</p>
    ) : (
      selectedSource ? renderDocuments() : (
        <>
          {renderSourceList(globalSources, 'Global Sources')}
          {renderSourceList(privateSources, 'Private Sources')}
          {globalSources.length === 0 && privateSources.length === 0 && (
            <p style={{ textAlign: 'center' }}>No sources found.</p>
          )}
        </>
      )
    )}
```

```
    {message && <p style={{ marginTop: '20px', color: 'red' }}>{message}</p>}
   </div>
  );
}

export default ListSources;
```