Introduction to AJAX What is AJAX?

1. AJAX stands for **Asynchronous JavaScript and XML**.
2. Allows web pages to update asynchronously by exchanging small amounts of data with the server behind the scenes.
3. Results in faster and more dynamic web applications without reloading the entire page.

- **Use Cases of AJAX:**
  1. Dynamic content loading (e.g., social media feeds).
  2. Form validation and submission without page reload.
  3. Auto-suggestions in search bars (e.g., Google search).
  4. Data fetching for dashboards (e.g., charts and analytics).
- **How AJAX Works:**
  1. **Event Occurs:** A user action triggers an event (e.g., button click).
  2. **XMLHttpRequest (XHR) Object:** JavaScript creates an XHR object.
  3. **Request Sent to Server:** Asynchronously sends a request.
  4. **Server Processes Request:** The server processes the request and returns data.
  5. **JavaScript Processes Response:** Updates the webpage without reloading.

Implement a simple example demonstrating HTTP Request and Response using AJAX to fetch data from an XML file and display it on a webpage.

Project Structure:

```
ajax-xml-demo/
│
├── index.html      # Main HTML file with UI elements
├── app.js          # JavaScript file with AJAX code
└── data.xml        # Sample XML data file
```

1. HTML File (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AJAX HTTP Request and Response with XML</title>
</head>
<body>
    <h1>AJAX Example: Fetching XML Data</h1>
```

```
    <!-- Button to trigger AJAX request -->
    <button onclick="fetchXMLData()">Load Student Data</button>

    <!-- Container to display the response -->
    <div id="output"></div>

    <!-- Link to external JavaScript file -->
    <script src="app.js"></script>
</body>
</html>
```

Explanation:

- The **button** triggers the fetchXMLData() function when clicked.
- The with id="output" will display the server response.

2. JavaScript File (app.js):

```
// Function to handle AJAX request and response
function fetchXMLData() {
    // Create a new XMLHttpRequest object
    const xhr = new XMLHttpRequest();

    // Initialize a GET request to fetch the XML file
    xhr.open('GET', 'data.xml', true);

    // Event listener for when the request completes successfully
    xhr.onload = function() {
        // Check if the HTTP status code is 200 (OK)
        if (xhr.status === 200) {
            // Parse the XML response
            const xmlDoc = xhr.responseXML;

            // Extract 'student' elements from the XML
            const students =
xmlDoc.getElementsByTagName('student');

            // Prepare HTML output
            let output = '<h2>Student List</h2>';
```

```
            // Iterate through each student and extract data
            for (let i = 0; i < students.length; i++) {
                const name =
students[i].getElementsByTagName('name')[0].textContent;
                const age =
students[i].getElementsByTagName('age')[0].textContent;
                output += `<p>Name: ${name}, Age: ${age}</p>`;
            }

            // Display the output in the HTML element with ID
'output'
            document.getElementById('output').innerHTML = output;
        } else {
            // Handle errors for non-200 status codes
            document.getElementById('output').innerHTML =
`<p>Error: ${xhr.status}</p>`;
        }
    };

    // Event listener for handling network errors
    xhr.onerror = function() {
        console.error('Request Error...');
        document.getElementById('output').innerHTML = '<p>Failed
to load data. Please try again.</p>';
    };

    // Send the request to the server
    xhr.send();
}
```

Explanation:

- Creates an **AJAX** request using XMLHttpRequest().
- Sends a **GET request** to data.xml.
- Parses the **XML response** and dynamically updates the webpage.
- Implements **error handling** for network issues.

3. XML File (data.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<students>
    <student>
        <name>John Doe</name>
        <age>22</age>
    </student>
    <student>
        <name>Jane Smith</name>
        <age>24</age>
    </student>
    <student>
        <name>Mike Johnson</name>
        <age>20</age>
    </student>
</students>
```

Explanation:

- The XML file contains a list of <student> elements.
- Each <student> has <name> and <age> child elements.

How to Run:

1. Save all files in the same directory (ajax-xml-demo).
2. Open index.html in a browser.
3. Click the **Load Student Data** button to see the XML data displayed.

Expected Output:

```
Student List
Name: John Doe, Age: 22
Name: Jane Smith, Age: 24
Name: Mike Johnson, Age: 20
```

Implement a simple example demonstrating HTTP Request and Response using AJAX to fetch data from a JSON file and display it on a webpage.

Project Structure:

```
ajax-json-demo/
│
├── index.html      # Main HTML file with UI elements
├── app.js          # JavaScript file with AJAX code
└── data.json       # Sample JSON data file
```

1. HTML File (index.html):

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AJAX HTTP Request and Response with JSON</title>
</head>
<body>
    <h1>AJAX Example: Fetching JSON Data</h1>

    <!-- Button to trigger AJAX request -->
    <button onclick="fetchJSONData()">Load Student Data</button>

    <!-- Container to display the response -->
    <div id="output"></div>

    <!-- Link to external JavaScript file -->
    <script src="app.js"></script>
</body>
</html>
```

Explanation:

- The **button** triggers the fetchJSONData() function when clicked.
- The with id="output" will display the server response.

2. JavaScript File (app.js):

```
// Function to handle AJAX request and response
```

```
function fetchJSONData() {
    // Create a new XMLHttpRequest object
    const xhr = new XMLHttpRequest();

    // Initialize a GET request to fetch the JSON file
    xhr.open('GET', 'data.json', true);

    // Event listener for when the request completes successfully
    xhr.onload = function() {
        // Check if the HTTP status code is 200 (OK)
        if (xhr.status === 200) {
            // Parse the JSON response
            const students = JSON.parse(xhr.responseText);

            // Prepare HTML output
            let output = '<h2>Student List</h2>';

            // Iterate through each student and extract data
            students.forEach(student => {
                output += `<p>Name: ${student.name}, Age:
${student.age}</p>`;
            });

            // Display the output in the HTML element with ID
'output'
            document.getElementById('output').innerHTML = output;
        } else {
            // Handle errors for non-200 status codes
            document.getElementById('output').innerHTML =
`<p>Error: ${xhr.status}</p>`;
        }
    };

    // Event listener for handling network errors
    xhr.onerror = function() {
        console.error('Request Error...');
        document.getElementById('output').innerHTML = '<p>Failed
to load data. Please try again.</p>';
    };

    // Send the request to the server
    xhr.send();
```

```
}
```

Explanation:

- Creates an **AJAX** request using XMLHttpRequest().
- Sends a **GET request** to data.json.
- Parses the **JSON response** and dynamically updates the webpage.
- Implements **error handling** for network issues.

3. JSON File (data.json):

```
[
    {
        "name": "John Doe",
        "age": 22
    },
    {
        "name": "Jane Smith",
        "age": 24
    },
    {
        "name": "Mike Johnson",
        "age": 20
    }
]
```

Explanation:

- The JSON file contains an **array** of **student objects**.
- Each **object** has name and age **properties**.

How to Run:

1. Save all files in the same directory (ajax-json-demo).
2. Open index.html in a browser.
3. Click the **Load Student Data** button to see the JSON data displayed.

Expected Output:

```
Student List
Name: John Doe, Age: 22
Name: Jane Smith, Age: 24
Name: Mike Johnson, Age: 20
```

AJAX can be seamlessly integrated into a **React.js** app to fetch data asynchronously and update the UI dynamically. In a React app, you typically use **AJAX** through **JavaScript fetch API**, **Axios**, or even the classic **XMLHttpRequest**

When to Use AJAX in a React App:

- **Fetching Data:** Load data from APIs or external files when a component mounts.
- **Form Submissions:** Send form data to a server without reloading the page.
- **Dynamic Updates:** Fetch new data based on user interactions (e.g., search, filters).

Example: Using AJAX (Fetch API) in a React App

Project Structure:

```
ajax-react-demo/
│
├─ public/
│    └─ data.json       # Sample JSON data file
│
└─ src/
     ├─ App.js          # Main React component
     └─ index.js        # React DOM render file
```

1. Sample JSON File (public/data.json):

```
[
    { "name": "John Doe", "age": 22 },
    { "name": "Jane Smith", "age": 24 },
    { "name": "Mike Johnson", "age": 20 }
]
```

2. React Component (src/App.js):

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

const App = () => {
    const [students, setStudents] = useState([]);
    const [error, setError] = useState(null);

    // Function to fetch data using AJAX (Fetch API)
    const fetchStudentData = async () => {
        try {
            const response = await fetch('/data.json'); //
Fetching from public folder
            if (!response.ok) {
                throw new Error(`HTTP error! Status:
${response.status}`);
            }
            const data = await response.json();
            setStudents(data);
        } catch (err) {
            setError(err.message);
        }
    };

    // useEffect to call the fetch function when the component
mounts
    useEffect(() => {
        fetchStudentData();
    }, []);

    return (
        <div>
            <h1>AJAX in React: Fetching Student Data</h1>
            <button onClick={fetchStudentData}>Reload
Data</button>

            {error && <p style={{ color: 'red' }}>Error:
{error}</p>}

            {students.length > 0 ? (
```

```
                <div>
                    <h2>Student List</h2>
                    {students.map((student, index) => (
                        <p key={index}>
                            Name: {student.name}, Age:
{student.age}
                        </p>
                    ))}
                </div>
            ) : (
                <p>No student data available.</p>
            )}
        </div>
    );
};

export default App;
```

If you have XML file instead of JSON use the below code:

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

function App() {
  const [students,
  setStudents] = useState([]);
  const [error,
  setError] = useState(null);

  // Function to
  fetch and parse XML data
  const fetchStudentData
  = async () => {
    try {
      const response =
      await fetch("/data.xml");
      if (!response.ok)
      {
        throw new Error
        (`HTTP
          error!
          Status: ${response.
            status}`);
```

```
      }
      const xmlText
      = await response.text();
      const parser
      = new DOMParser();
      const xmlDoc
      = parser.parseFromString(xmlText, "application/xml");

      // Extract
      student data from XML
      const
      studentNodes =
      xmlDoc.
      getElementsByTagName("student");
      const
      studentArray
      = Array.from
      (studentNodes).
      map((student) => ({
        name:
        student.
        getElementsByTagName
        ("name")[0]
        ?.textContent
        || "N/A",
        age:
        student.
        getElementsByTagName
        ("age")[0]?.textContent
        || "N/A",
      }));

      setStudents(studentArray);
    } catch (err) {
      setError(err.message);
    }
  };

  // useEffect - fetch students when component mounts
  useEffect(() => {
    fetchStudentData();
  }, []);
```

```
  return (
    <div>
      <h1>Student List</h1>
      <button onClick={fetchStudentData}>Load Student
Data</button>

      {error && <p style={{ color: "red" }}>Error: {error}</p>}

      {students.length > 0 ? (
        <ul>
          {students.map((student, index) => (
            <li key={index}>
              <strong>Name:</strong> {student.name},
<strong>Age:</strong> {student.age}
            </li>
          ))}
        </ul>
      ) : (
        <p>No student data available</p>
      )}
    </div>
  );
}

export default App;
```

3. React DOM Rendering (src/index.js):

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.createRoot(document.getElementById('root')).render(<App
/>);
```

How This Works:

1. **AJAX Request:** The fetchStudentData function uses the **Fetch API** to get data from data.json.
2. **Error Handling:** If the fetch fails, an error is displayed.
3. **Data Display:** The students array is mapped to dynamically render the list of students.
4. **React Lifecycle:** useEffect triggers the fetch when the component mounts.

How to Run the React App:

1. **Initialize React App:**

```
npx create-react-app ajax-react-demo
cd ajax-react-demo
```

2. **Replace the Code:**

- Replace src/App.js and src/index.js with the provided code.
- Add data.json to the public folder.

3. **Start the App:**

```
npm start
```

4. **Open in Browser:**

- Visit http://localhost:3000 to see the AJAX demo in action.

Key Points:

- **useEffect:** Ideal for calling AJAX requests when the component loads.
- **Error Handling:** Always handle fetch errors to avoid breaking the UI.
- **State Management:** Use useState to manage fetched data and display it dynamically.

AJAX File Handling:

Project Structure:

```
ajax-demo/
|
```

```
├── index.html      # Main HTML file with UI elements
├── app.js          # JavaScript file with AJAX code
├── data.json       # Sample JSON data file
└── data.xml        # Sample XML data file
```

// index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>AJAX JSON and XML Demo</title>
</head>
<body>
    <h1>AJAX Example: Fetching JSON & XML Data</h1>

    <!-- Buttons to trigger AJAX requests -->
    <button onclick="fetchJSONData()">Load JSON Data</button>
    <button onclick="fetchXMLData()">Load XML Data</button>

    <!-- Container to display the response -->
    <div id="output"></div>

    <!-- Link to external JavaScript file -->
    <script src="app.js"></script>
</body>
</html>
```

// app.js

```javascript
// Function to handle AJAX request and response for JSON
function fetchJSONData() {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', 'data.json', true);
    xhr.onload = function() {
        if (xhr.status === 200) {
            const data = JSON.parse(xhr.responseText);
            let output = '<h2>Student List (JSON)</h2>';
            data.forEach(student => {
                output += `<p>Name: ${student.name}, Age:
${student.age}</p>`;
```

```
            });
            document.getElementById('output').innerHTML = output;
        } else {
            document.getElementById('output').innerHTML =
`<p>Error: ${xhr.status}</p>`;
        }
    };
    xhr.onerror = function() {
        document.getElementById('output').innerHTML = '<p>Failed
to load data. Please try again.</p>';
    };
    xhr.send();
}

// Function to handle AJAX request and response for XML
function fetchXMLData() {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', 'data.xml', true);
    xhr.onload = function() {
        if (xhr.status === 200) {
            const xmlDoc = xhr.responseXML;
            const students =
xmlDoc.getElementsByTagName('student');
            let output = '<h2>Student List (XML)</h2>';
            for (let i = 0; i < students.length; i++) {
                const name =
students[i].getElementsByTagName('name')[0].textContent;
                const age =
students[i].getElementsByTagName('age')[0].textContent;
                output += `<p>Name: ${name}, Age: ${age}</p>`;
            }
            document.getElementById('output').innerHTML = output;
        } else {
            document.getElementById('output').innerHTML =
`<p>Error: ${xhr.status}</p>`;
        }
    };
    xhr.onerror = function() {
        document.getElementById('output').innerHTML = '<p>Failed
to load data. Please try again.</p>';
    };
    xhr.send();
```

```
}
```

// data.json

```
[
    { "name": "John Doe", "age": 22 },
    { "name": "Jane Smith", "age": 24 },
    { "name": "Mike Johnson", "age": 20 }
]
```

// data.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student>
        <name>John Doe</name>
        <age>22</age>
    </student>
    <student>
        <name>Jane Smith</name>
        <age>24</age>
    </student>
    <student>
        <name>Mike Johnson</name>
        <age>20</age>
    </student>
</students>
```

```
<script src="app.js" defer></script>
```