XMLHttpRequest (legacy approach)

🗃️ `XMLHttpRequest` in JavaScript — Legacy AJAX Method Explained

📌 What is `XMLHttpRequest`?

`XMLHttpRequest` (often abbreviated as **XHR**) is a built-in JavaScript object that allows you to send **HTTP requests** to a server **without reloading the page**. It was the standard way to make AJAX requests **before the Fetch API**.

Though considered **legacy**, it's still used in some older applications or environments where backward compatibility is needed.

✅ Basic Syntax

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.example.com/data", true);

xhr.onload = function () {
  if (xhr.status >= 200 && xhr.status < 300) {
    console.log("Response:", xhr.responseText);
  } else {
    console.error("Request failed:", xhr.status);
  }
};

xhr.onerror = function () {
  console.error("Network Error");
};

xhr.send();
```

⚙️ Key Methods

Method	Description
`open(method, url, async)`	Initializes the request.
`send([body])`	Sends the request.
`setRequestHeader()`	Sets custom HTTP headers.
`abort()`	Cancels the request.

🧾 Making a POST Request

```
const xhr = new XMLHttpRequest();
xhr.open("POST", "https://api.example.com/data", true);
xhr.setRequestHeader("Content-Type", "application/json");

xhr.onload = function () {
  if (xhr.status === 201) {
    console.log("Created:", xhr.responseText);
  }
};

xhr.send(JSON.stringify({ name: "John", age: 30 }));
```

🔍 Reading Response Data

```
xhr.responseText   // Raw text response
xhr.responseXML    // Parsed XML (if response is XML)
```

📡 Monitoring Progress

```
xhr.onprogress = function (event) {
  if (event.lengthComputable) {
```

```
    console.log(`Received ${event.loaded} of ${event.total} bytes`);
  }
};
```

⚠️ Error Handling

XHR does not reject on HTTP errors (e.g., 404 or 500). You must check `xhr.status` manually in the `onload` callback.

```
xhr.onload = function () {
  if (xhr.status >= 400) {
    console.error("HTTP error:", xhr.status);
  }
};
```

⏱️ Timeout Handling

```
xhr.timeout = 5000; // 5 seconds
xhr.ontimeout = () => console.log("Request timed out!");
```

🧠 Comparison with Fetch

Feature	`XMLHttpRequest`	`Fetch API`
Modern/Legacy	Legacy	Modern
Promises	❌ No	✅ Yes
Readable Code	❌ Verbose, callback-based	✅ Cleaner, async/await
Stream API support	❌ No	✅ Yes
JSON auto-parsing	❌ Manual	✅ With `.json()`

✅ Use Cases (Today)

- Supporting old browsers (e.g., IE11)

- Embedded systems or environments that don't support Fetch
- Learning how AJAX worked before Promises

🧪 Example: Reusable XHR Function

```
function sendRequest(method, url, data = null) {
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest();
    xhr.open(method, url);
    xhr.setRequestHeader("Content-Type", "application/json");

    xhr.onload = () => {
      if (xhr.status >= 200 && xhr.status < 300) {
        resolve(xhr.responseText);
      } else {
        reject(`HTTP Error: ${xhr.status}`);
      }
    };

    xhr.onerror = () => reject("Network Error");
    xhr.send(data ? JSON.stringify(data) : null);
  });
}
```

📌 Summary

- `XMLHttpRequest` allows asynchronous HTTP communication with servers.
- It's verbose and callback-based, lacking Promise support.
- Still useful in legacy systems or when Fetch is not available.
- Best replaced with the Fetch API in modern applications.

Let me know if you'd like a comparison demo between `XMLHttpRequest` and `fetch()` using the same API call.