JSON, PHP & AJAX

Md. Arman Hossain
Lecturer, CSE, EWU

JSON

JSON is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

- **Lightweight**: Simple and minimal, ideal for data exchange over networks.
- **Readable**: Human-friendly format with a straightforward syntax.
- **Language-Independent**: JSON is supported by most modern programming languages with libraries for parsing and generating JSON data.

Syntax:

Data stored in JSON in key:value manner

{“Key”:”Value”}

Supported Data Types

- Strings (enclosed in double quotes)
- Numbers
- Booleans (true, false)
- Null
- Objects
- Arrays

```json
{
  "name": "Alice",
  "age": 25,
  "isStudent": true,
  "skills": [
    "Python",
    "JavaScript",
    "SQL"
  ],
  "address": {
    "street": "123 Main St",
    "city": "Wonderland",
    "zipcode": 12345
  }
}
```

JSON In JavaScript

JSON to a JavaScript object:

```
const jsonString = '{"name": "Alice", "age": 25}';
const obj = JSON.parse(jsonString); // Converts JSON string to object
console.log(obj.name); // "Alice"
```

JavaScript object to JSON

```
const user = { name: "Bob", age: 30 };
const jsonString = JSON.stringify(user); // Converts object to JSON string
console.log(jsonString); // '{"name":"Bob","age":30}'
```

JSON In PHP

JSON to PHP array or object:

```php
$jsonString = '{"name": "Alice", "age": 25}';
$data = json_decode($jsonString, true); // Converts JSON string to associative array
echo $data['name']; // "Alice"
```

PHP array to JSON

```php
$user = ["name" => "Bob", "age" => 30];
$jsonString = json_encode($user); // Converts array to JSON string
echo $jsonString; // '{"name":"Bob","age":30}'
```

Efficient Data Handling, Introduction to API-Based Web Development

- Normally we use HTML form or urls to request a page from server.
  - In response, server includes data into HTML page and sends back the whole pages including data, HTML, CSS and JS.
- This approach is not efficient when your page structure is fixed but it is only the data you need to get changed.
- To make it efficient, there should be an approach where server will return only the data in a pre-defined format (e.g: JSON, XML).
  - The frontend extracts and replaces necessary data from the response (e.g. JSON) and render it to the user.
- However, in this approach we will use AJAX to initiate a request to the server instead of form and urls to the browser. (why??)

A Simple User Interaction Using PHP, JSON, and AJAX

Steps:

1. **Frontend**: An AJAX request is initiated (e.g., using JavaScript).
2. **Server-Side (PHP):** Handles the request, processes data, and sends a JSON response.
3. **Frontend**: Processes the JSON response and updates the webpage dynamically.

Backend (PHP): Create an API Endpoint

fetch_user.php

```php
<?php
header("Content-Type: application/json"); // Set response type to JSON

// Simulate data (in a real-world scenario, fetch from a database)
$users = [
    1 => ["name" => "Alice", "email" => "alice@example.com"],
    2 => ["name" => "Bob", "email" => "bob@example.com"],
    3 => ["name" => "Charlie", "email" => "charlie@example.com"],
];

// Get the user ID from the AJAX request
$userId = $_GET["id"] ?? null;

// Check if the user exists
if ($userId && isset($users[$userId])) {
    echo json_encode(["status" => "success", "user" => $users[$userId]]);
} else {
    echo json_encode(["status" => "error", "message" => "User not found"]);
}
```

Frontend (HTML and JavaScript) index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PHP JSON AJAX Example</title>
    <script>
      function fetchUser() {
        const userId = document.getElementById("userId").value;
        // Create an AJAX request
        const xhr = new XMLHttpRequest();
        xhr.open("GET", `fetch_user.php?id=${userId}`, true);
        xhr.onload = function() {
          if (xhr.status === 200) {
            const response = JSON.parse(xhr.responseText);
            if (response.status === "success") {
              // Display user details
              document.getElementById("result").innerHTML =
                    `<p><strong>Name:</strong> ${response.user.name}
                    </p>
                    <p><strong>Email:</strong> ${response.user.email}
                    </p>`;
            } else {
```

Frontend (HTML and JavaScript)
index.html (cont.)

```
24                  document.getElementById("result").innerHTML =
25                      `<p style="color: red;">${response.message}</p>`;
26                  }
27              }
28          };
29          xhr.send(); // Send the request
30      }
31    </script>
32  </head>
33  <body>
34    <h1>Fetch User Details</h1>
35    <label for="userId">Enter User ID:</label>
36    <input type="number" id="userId" placeholder="Enter ID (1-3)" />
37    <button onclick="fetchUser()">Fetch User</button>
38    <div id="result" style="margin-top: 20px;"></div>
39  </body>
40 </html>
```

AJAX

1. **AJAX** (Asynchronous JavaScript and XML) is a set of web development techniques that allows web applications to send and retrieve data from a server asynchronously, without reloading the webpage.
2. It primarily uses JSON as its data format instead of XML in modern web development.
3. **Asynchronous Communication**: Allows data to be sent to and from a server in the background without disturbing the current state of the webpage.
4. **No Page Reloads**: The page doesn't refresh, leading to a more seamless user experience.
5. **Partial Updates**: Only specific parts of a webpage are updated, reducing server load and improving speed.

How it works

AJAX operates by exchanging data between the browser and server. The key steps involved are:

1. A user event (e.g., clicking a button) triggers an AJAX request.
2. JavaScript uses the XMLHttpRequest or a modern fetch() method to send the request to the server.
3. The server processes the request and responds with data (often in JSON format).
4. JavaScript processes the server response
5. And updates the webpage dynamically.

How it works

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "fetch_data.php", true); // Specify request method and URL
xhr.onload = function () {
    if (xhr.status === 200) {
        console.log(xhr.responseText); // Display response from server
    }
};
xhr.send(); // Send the request
```

Equivalent to fetch

```
function fetchData() {
    fetch("fetch_data.php")
        .then(response => response.json()) // Parse JSON response
        .then(data => {
            console.log(data.message); // Output: "Hello, world!"
        })
        .catch(error => console.error("Error:", error));
}
```

xmlHTTPRequest (post)

```
const xhr = new XMLHttpRequest();
xhr.open("POST", "fetch_data.php", true); // Set the method and URL

// Set the request headers
xhr.setRequestHeader("Content-Type", "application/json");

// Set up the response handling
xhr.onload = function () {
  if (xhr.status === 200) {
    const response = JSON.parse(xhr.responseText); // Parse the JSON response
    console.log(response); // Log the response
  } else {
    console.error("Error! HTTP status:", xhr.status);
  }
};

// Handle network errors
xhr.onerror = function () {
  console.error("Request failed");
};

// Send the JSON data in the request body
const data = JSON.stringify({ name: "John", age: 30 });
xhr.send(data);
```

Fetch for Post Request

```
fetch("fetch_data.php", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ name: "John", age: 30 }), // JSON data
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error("Error:", error));
```

XmlHttpRequest vs fetch

- Instead of dealing with event-based callbacks like in XMLHttpRequest, the Fetch API returns a Promise, allowing developers to chain subsequent operations using .then() and .catch() methods.
- This results in cleaner, more readable code and better error handling capabilities.

Thank you