1

Pedro H. Maciel /Camilly V. Schmidt

Orientador: Emerson Rocha | Helder Pestana

Sorocaba

2025

O QUE É AJAX?

AJAX (Asynchronous JavaScript and XML) é uma técnica de desenvolvimento web que permite atualizar partes de uma página da web sem precisar recarregar a página inteira. Isso torna as interações na web mais rápidas e dinâmicas, melhorando a experiência do usuário

Explicando, basicamente:

- **Assíncrono**: Significa que as requisições feitas ao servidor podem ser executadas em segundo plano, sem interromper a interação do usuário com a página.

- **JavaScript**: O código JavaScript é usado para enviar e receber dados do servidor.

- **XML**: Originalmente, o AJAX utilizava XML para o formato de troca de dados, mas hoje em dia, é mais comum usar o formato JSON (JavaScript Object Notation), que é mais leve e fácil de trabalhar.

Como funciona o AJAX?

1. O JavaScript faz uma **requisição assíncrona** ao servidor, geralmente usando o método XMLHttpRequest ou a API fetch.

2. O servidor processa essa requisição e envia uma resposta.

3. O JavaScript recebe a resposta e atualiza apenas a parte relevante da página, sem precisar recarregar a página inteira.

Exemplos comuns de uso:

01. JQuery - $.AJAX ( )

```
$("#carregar").click(function( ){
  $.ajax({
    url: "https://jsonplaceholder.typicode.com/posts/1",
    type: "GET",
    success: function(resposta){
      $("#resultado").html("<h3>" + resposta.title + "</h3><p>" + resposta.body + "</p>");
    },
    error: function(){
      alert("Erro na requisição!");
    }
  });
});
```

- `$("#carregar").click(...)` → quando o botão com `id="carregar"` for clicado, executa a função.
- `$.ajax({...})` → faz a chamada AJAX usando jQuery
- `url` → endereço do recurso (neste caso, uma API fake de posts)
- `type: "GET"` → método HTTP usado (poderia ser POST, PUT, DELETE etc.)
- `success:` → o que fazer se der certo (inserimos o título e o corpo no `#resultado`)
- `error:` → o que fazer se der erro (aqui apenas um `alert`)

02. XMLHttpRequest (XHR)

```
<button onclick="carregar()">Carregar com XHR</button>
<div id="saida"></div>
```

```
<script>
  function carregar(){
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "https://jsonplaceholder.typicode.com/posts/1", true);

    xhr.onload = function(){
      if(xhr.status === 200){
        var resposta = JSON.parse(xhr.responseText);
        document.getElementById("saida").innerHTML =
"<h3>"+resposta.title+"</h3><p>"+resposta.body+"</p>";
      }
    };

    xhr.onerror = function(){
      alert("Erro na requisição!");
    };

    xhr.send();
  }
</script>
```

- `new XMLHttpRequest()` → cria o objeto responsável pela requisição
- `xhr.open("GET", url, true)` → define o método (`GET`), o endereço e `true` significa assíncrono
- `xhr.onload` → função chamada quando a resposta chega
- `xhr.status === 200` → garante que deu certo
- `xhr.responseText` → resposta bruta (texto)
- `JSON.parse(...)` → transforma em objeto JS
- `xhr.onerror` → o que fazer se der erro
- `xhr.send()` → envia a requisição

03. Fetch ( )

```
function buscar(){
  fetch("https://jsonplaceholder.typicode.com/posts/1")
```

```
    .then(response => response.json())
    .then(data => {
      document.getElementById("resposta").innerHTML =
        `<h3>${data.title}</h3><p>${data.body}</p>`;
    })
    .catch(() => alert("Erro ao carregar!"));
}
```

- `fetch(url)` → inicia a requisição
- `.then(response => response.json())` → transforma a resposta em JSON
- `.then(data => {...})` → usa os dados (inserindo no HTML)
- `.catch(...)` → trata erros
- Uso de **template string** (`` `${variavel}` ``) → deixa o código mais limpo

Em resumo, o método 01 é mais antigo, mas útil se ja usa JQuery; o métod 02 é raiz, dá mais trabalho e o método 03 é moderno, exuto, padrão atual