



# Programação para Internet

---

## Módulo 7

### Requisições HTTP Assíncronas e a Técnica Ajax - Parte 1

(Mensagens HTTP, Ajax, Conceitos, JavaScript Assíncrono, XMLHttpRequest, JSON)

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Conteúdo protegido por direito autoral, nos termos da Lei nº 9 610/98

A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

# Conteúdo do Módulo

## Parte 1

- Mensagens HTTP
- Introdução à técnica Ajax e conceitos relacionados
- Ajax com o XMLHttpRequest
  - Formas de uso, tratamento de erros, muitos exemplos

## Parte 2

- Ajax com a API Fetch
- JavaScript Promises
- API Fetch com async / await

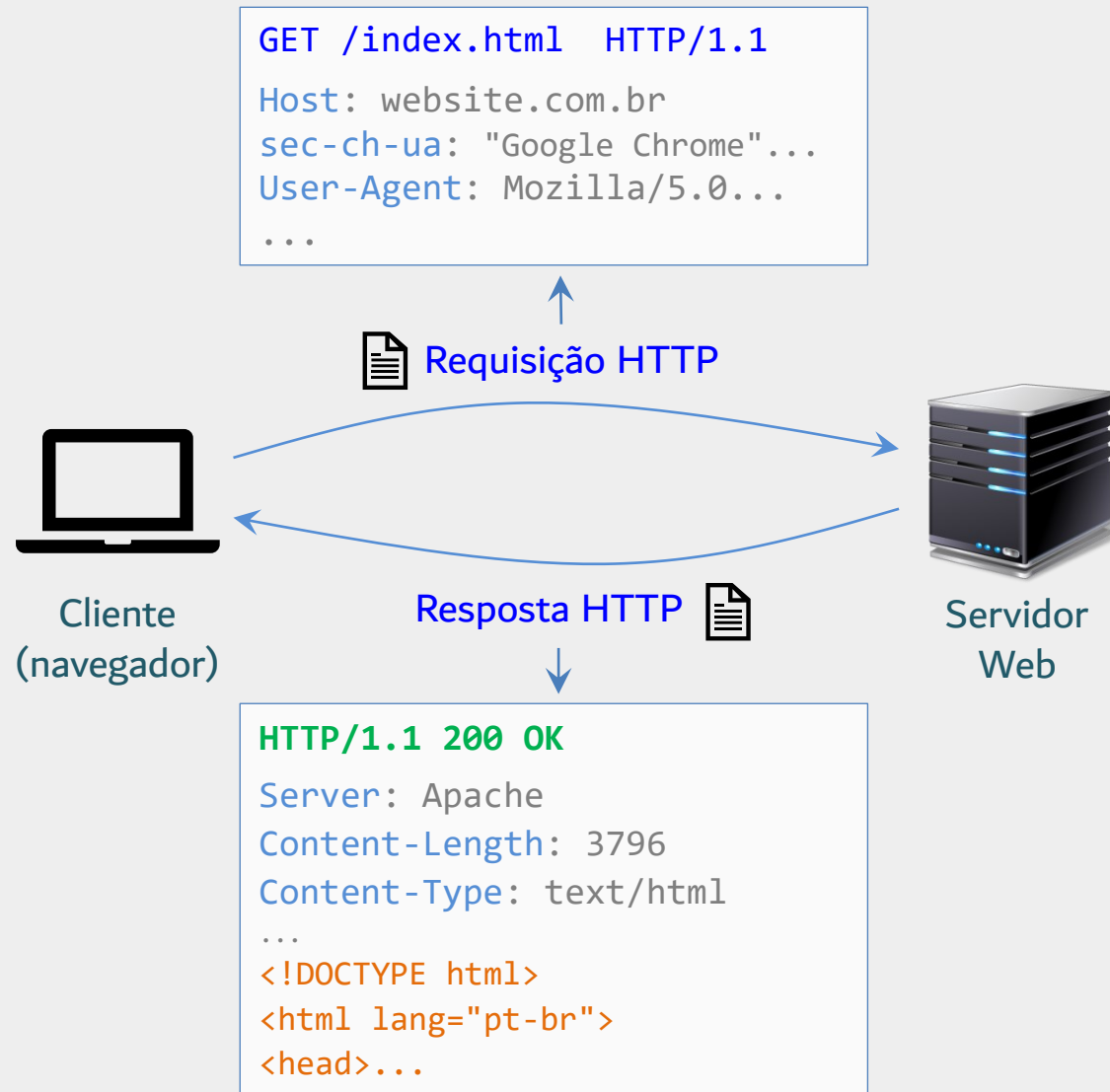
# Introdução ao Protocolo HTTP

---

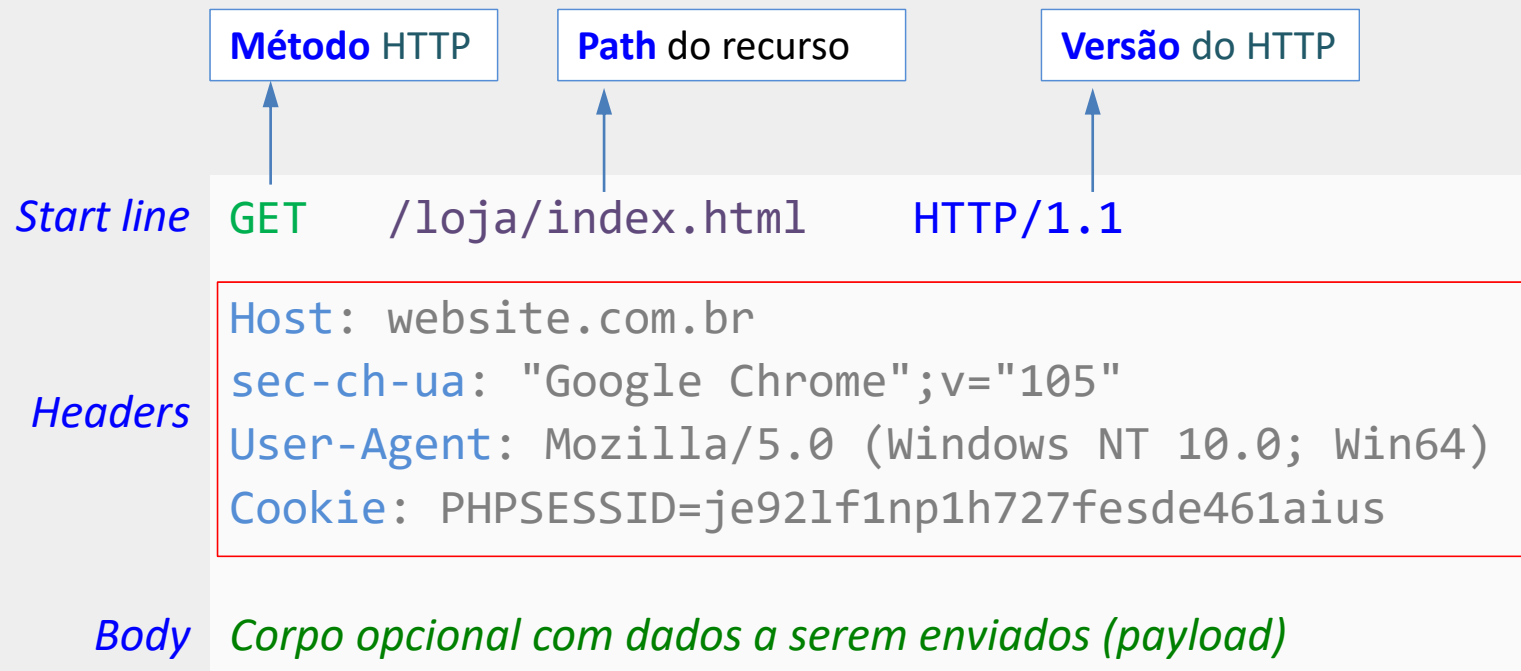
# Mensagens HTTP

- Conceito crucial para entendimento da técnica Ajax
- Visão geral, sem entrar em detalhes do protocolo

# Noção de Mensagens HTTP



# Exemplo Simplificado de Mensagem de Requisição HTTP



Há headers padronizados e personalizados

# Exemplo Simplificado de Mensagem de Requisição HTTP

*Start line* **POST** /loja/cadastro.php HTTP/1.1

*Headers*  
Host: website.com.br  
sec-ch-ua: "Google Chrome";v="105"  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64)  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 49

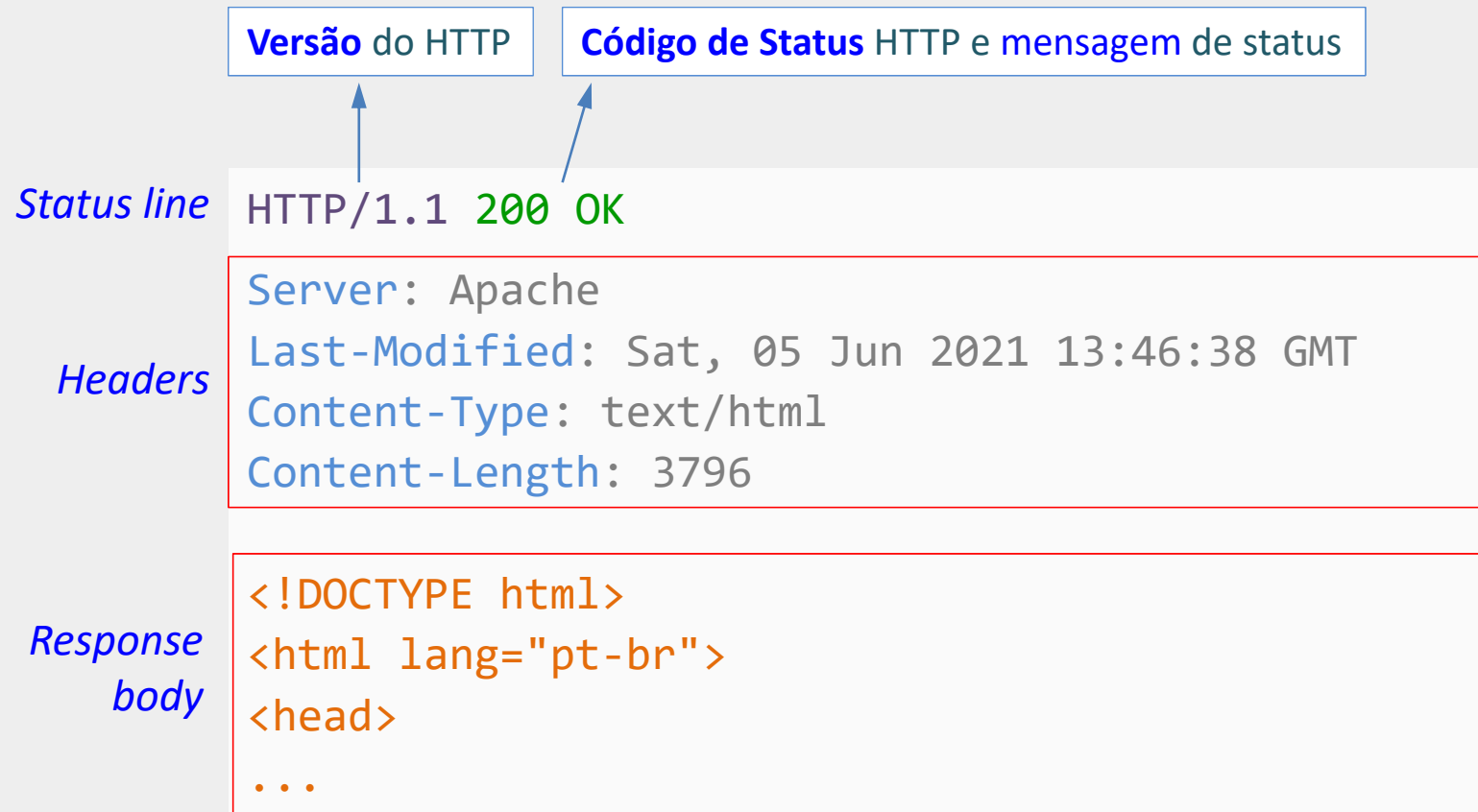
*Body* nome=Fulano&email=fulano%40mail.com&telefone=1234



Nome	E-mail	Telefone
<input type="text" value="Fulano"/>	<input type="text" value="fulano@mail.com"/>	<input type="text" value="1234"/>

Exemplo de requisição **POST** com dados de formulário

# Exemplo Simplificado de Mensagem de Resposta HTTP





# Verificando Dados da Requisição no Navegador

No Google Chrome: F12 → Network → F5 → Sel. Arquivo → Headers → Request Headers → View source

The screenshot shows the Google Chrome DevTools interface. The 'Network' tab is selected in the top toolbar. The 'Filter' dropdown is set to 'Fetch/XHR'. The 'Name' column lists two requests: 'Ex1-hello-ajax/' and 'bg2.jpg'. The 'Headers' sub-tab is selected for the first request. The 'Response Headers' section shows the following details:

- Accept-Ranges: bytes
- Connection: Keep-Alive
- Date: Mon, 08 May 2023 11:22:43 GMT
- ETag: "6db-5fae2eede0abf"
- Keep-Alive: timeout=5, max=100
- Last-Modified: Thu, 04 May 2023 19:11:00 GMT
- Server: Apache/2.4.51 (Win64) PHP/8.1.0

The 'Request Headers' section shows the following details:

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7
- Accept-Encoding: gzip, deflate, br
- Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
- Cache-Control: max-age=0
- Connection: keep-alive
- Host: localhost

The background of the browser window shows a webpage titled 'Faculdade de Computação da UFU' with a button that says 'Clique para carregar mais conteúdo com Ajax'.

# Alguns Códigos de Status HTTP

- 200 OK – resposta padrão para sucesso
- 302 Found – recurso encontrado, seguido por redirecionamento
- 304 Not Modified – recurso não modificado. Utilize a versão em cache
- 400 Bad Request – possível erro do cliente ao montar a requisição
- 403 Forbidden – acesso ao recurso não autorizado
- 404 Not Found – recurso não encontrado
- 500 Internal Server Error – erro interno no servidor

# Introdução à Técnica Ajax

---

# Surgimento da Técnica Ajax

- No final da década de 1990 e início dos anos 2000 começou a surgir um novo modelo de aplicações web, que ofereciam maior **interatividade** e **responsividade**
- Um exemplo era o Google Suggest (hoje chamado de Google Autocomplete) e o Google Maps
- Possível devido ao surgimento do XMLHttpRequest

# Surgimento do Termo Ajax em si



## Ajax: A New Approach to Web Applications



February 18, 2005

If anything about current interaction design can be called “glamorous,” it’s creating Web applications. After all, when was the last time you heard someone rave about the interaction design of a product that wasn’t on the Web? (Okay, besides the iPod.) All the cool, innovative new projects are online.

Despite this, Web interaction designers can’t help but feel a little envious of our colleagues who create desktop software. Desktop applications have a richness and responsiveness that has seemed out of reach on the Web. The same simplicity that enabled the Web’s rapid proliferation also creates a gap between the experiences we can provide and the experiences users can get from a desktop application.

*Jesse James Garrett is a founder of Adaptive Path*

That gap is closing. Take a look at Google Suggest. Watch the way the suggested terms update as you type, almost instantly. Now look at Google Maps. Zoom in. Use your cursor to grab the map and scroll around a bit. Again, everything happens almost instantly, with no waiting for pages to reload.

Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what’s possible on the Web.

### Defining Ajax

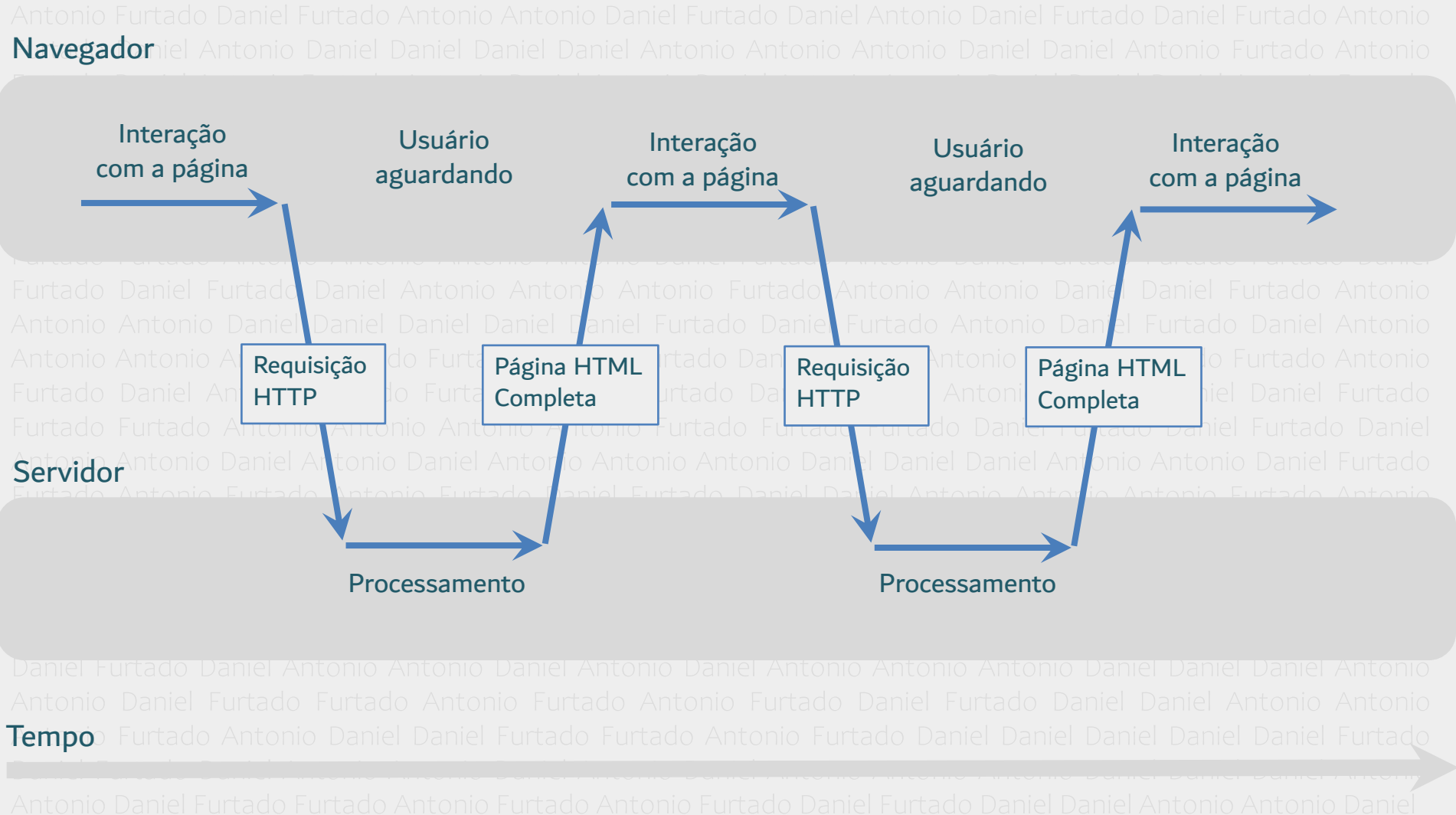
Ajax isn’t a technology. It’s really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

- O termo **Ajax** foi proposto em 2005 por Jesse Garrett
- A técnica em sua essência já era utilizada, mas não havia um nome
- Jesse Garrett destaca que a técnica permite tornar as aplicações web mais parecidas com aplicações desktop, melhorando a responsividade

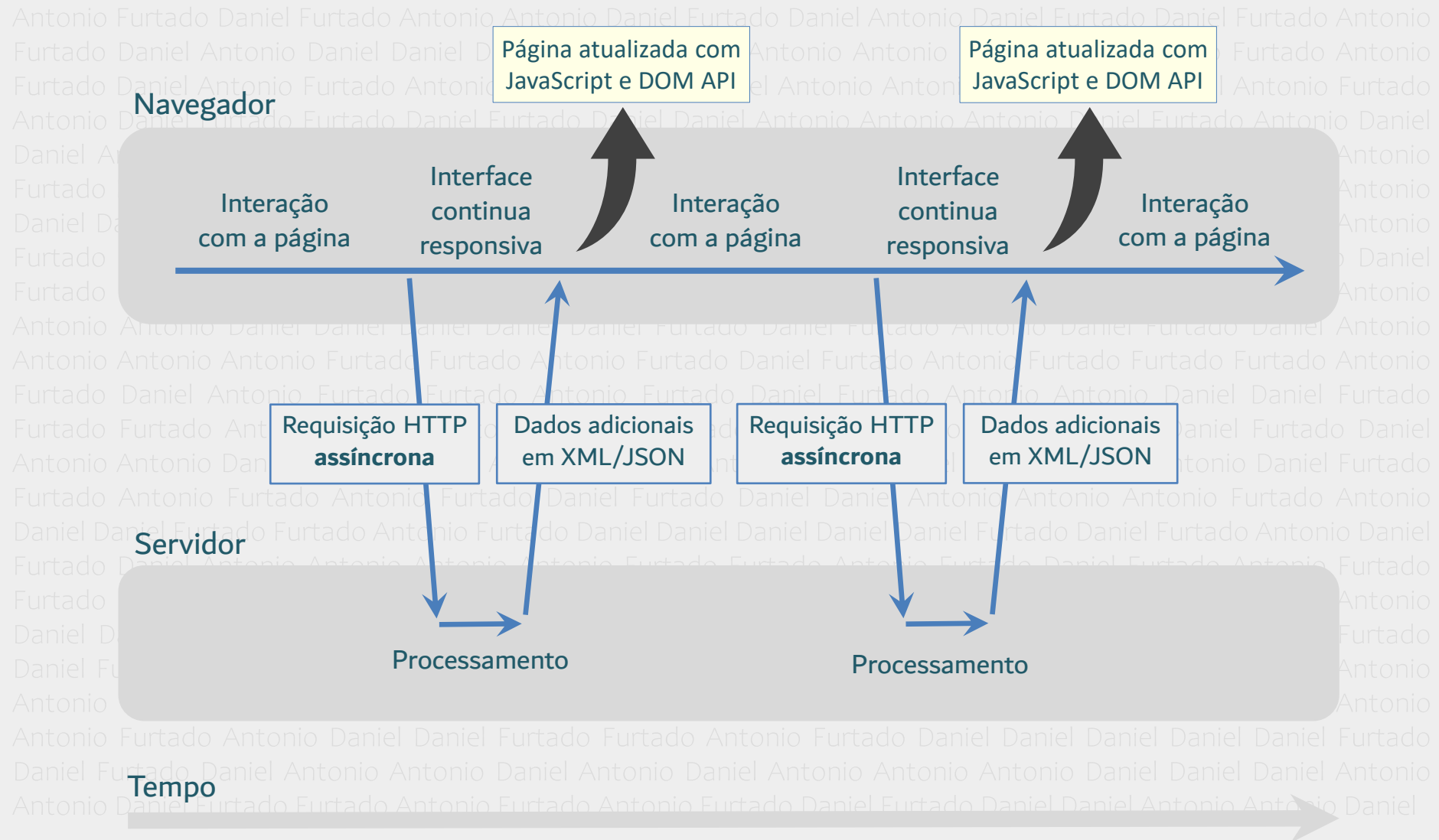
# Técnica Ajax

- Técnica para realizar **atualizações incrementais** na página
  - Permite atualizar página já carregada no navegador
  - Por meio de busca rápida por conteúdo adicional no servidor
  - E inserção na página dinamicamente (atualizando árvore DOM)
  - Dispensa a necessidade de carregar uma nova página inteiramente
- Utiliza requisições HTTP **assíncronas** (executadas em 2º plano)
  - Sem interromper a navegação do usuário
  - Sem congelar a interface
- **Objetivo:** aplicação mais ágil, eficiente e melhor experiência do usuário

# Aplicação Web Convencional



# Aplicação Web com Ajax





# Exemplo

CEP

38408-100

Endereço

Av João Naves de Ávila

Bairro

Santa Mônica

Cidade

Uberlândia

Estado

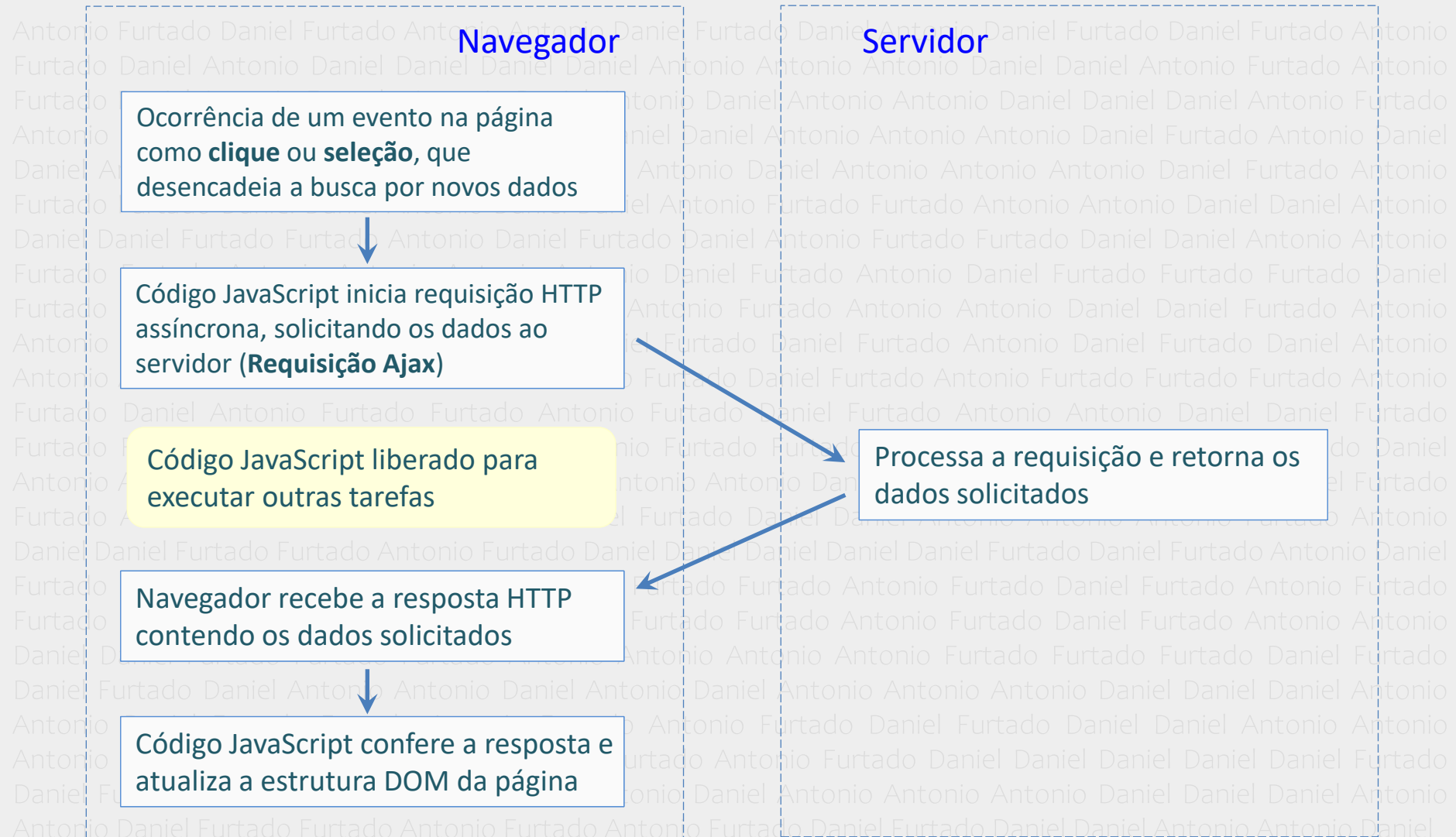
Minas Gerais

- Atualização quase instantânea
- Comunicação com servidor eficiente
- Troca de dados essenciais



Código JavaScript insere os dados no formulário (atualizando árvore DOM)

# Ideia Geral da Técnica Ajax



# Informações Adicionais sobre Ajax

- Ajax não é uma linguagem de programação
- Ajax é uma técnica que combina várias tecnologias como:
  - HTML, CSS
  - XML / JSON
  - JavaScript
  - Árvore DOM
  - XMLHttpRequest / Fetch

# Informações Adicionais sobre Ajax

- Principal formato (na época) para troca assíncrona de dados
- Outros formatos são mais comuns atualmente (JSON)

Ajax = Asynchronous JavaScript and XML

- Requisições HTTP em segundo plano (outra thread)
- Sem congelamentos da interface do usuário

# Outros Exemplos de Aplicações

- Websites do tipo **SPA** (Single-Page Application)
  - Aplicação de Página Única
  - Conteúdo principal carregado uma única vez
  - Conteúdo adicional carregado dinamicamente com Ajax
    - TAGs HTML podem ser geradas no navegador
    - Conteúdo pode vir em JSON
    - Esforço de renderização no lado cliente
- Buscas instantâneas
- Rolagem infinita

# Como realizar Requisições Ajax com JavaScript?

## Nativo

- XMLHttpRequest
- API Fetch

## Bibliotecas

- jQuery
- Axios

# Ajax com o XMLHttpRequest


---

# Objeto XMLHttpRequest (XHR)

- Projetado para buscar XML via requisições HTTP
- Também suporta outros formatos (ex. JSON)
- Amplamente suportado pelos navegadores
- É uma API da Web - não faz parte da JavaScript em si
- Baseado em funções de callback
- Algumas facilidades: timeout, andamento da requisição
- Código mais longo, mas conceitualmente mais simples
  - Não utiliza Promises
  - Fácil aprendizado
- Dificuldades: callback hell



# Passos para Iniciar Requisição Ajax com o XHR

1. Criar objeto XMLHttpRequest (XHR)
  2. Indicar a URL da requisição - método open
  3. Indicar função para tratar resposta - propriedade onload
  4. Enviar a requisição - método send
- 

# Exemplo Simples de Requisição Ajax

```
let xhr = new XMLHttpRequest();  
xhr.open("GET", "filmes.txt", true);  
xhr.onload = function () {  
    console.log(xhr.responseText);  
};  
xhr.send();
```

*Propriedade **onload**  
Permite executar  
uma ação quando a  
requisição finalizar  
e a resposta estiver  
pronta*

*Prop. **responseText**  
Contém a resposta  
textual retornada  
pelo servidor*

**OBS:** Exemplo simples, sem tratamento de erros

# Método open

```
xhr.open("GET", "busca.php?id=1", true);
```

**Método** da requisição

**Recurso** sendo requisitado  
(URL completa ou caminho relativo)

*Neste exemplo um  
parâmetro é enviado pela  
própria URL*

true para **assíncrona**  
false para **síncrona**

*Caso não informado, o  
padrão é **assíncrona***

O método **open** permite configurar a requisição, antes de enviá-la, indicando o método HTTP, a URL do recurso sendo requisitado, e a forma de tratamento da requisição (síncrona ou assíncrona). Há dois outros parâmetros opcionais, user e password, que podem ser usados para fins de autenticação

# Requisição Assíncrona x Síncrona com o XHR

## Requisição Assíncrona

- O código JS prossegue enquanto requisição é gerenciada pelo navegador
- É possível executar outras operações enquanto a requisição é tratada
- O andamento da requisição pode ser monitorado com eventos

## Requisição Síncrona

- Considerada **obsoleta** (mdn web docs)
- O código JavaScript fica “bloqueado”, aguardando resposta do servidor
- Não é recomendada, pois pode prejudicar a responsividade
- Se for utilizar, que seja fora da *thread* principal, com [Web Workers](#)
- Alguns recursos não estão disponíveis

# Tratando Eventuais Erros de Rede

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "filmes.txt", true);

xhr.onload = function () {
    console.log(xhr.responseText);
};

xhr.onerror = function () {
    console.log("Erro a nível de rede");
};

xhr.send();
```

*Propriedade **onerror**  
Permite tratar erros  
de rede que tenham  
impedido a finalização  
da requisição*

# Observações sobre **onerror**

```
xhr.onerror = function () {  
    console.log("Erro a nível de rede");  
};
```

Cobre apenas erros a nível de rede, como:

- Falha na conexão com a internet
- Servidor não encontrado ou demorando para responder
- Alguns erros relacionados a permissões de acesso (CORS)

Não disparado em situações como:

- Servidor responde com código de status de erro (500, 404, etc.)
- Servidor responde com dados inesperados
  - Ex.: mensagens de erros/warnings do back-end

# Verificando o código de status HTTP retornado

```
xhr.onload = function () {  
    if (xhr.status == 200)  
        console.log(xhr.responseText);  
    else  
        console.error("Falha: " + xhr.status + xhr.responseText);  
};  
  
xhr.onerror = function () {  
    console.log("Erro de rede");  
};
```

`xhr.status` permite verificar o código de status HTTP retornado pelo servidor  
`200` é o código de status padrão indicando sucesso/ok.

# Verificando o código de status HTTP retornado

```
xhr.addEventListener("load", function () {  
    if (xhr.status == 200)  
        console.log(xhr.responseText);  
    else  
        console.error("Falha: " + xhr.status + xhr.responseText);  
});  
  
xhr.addEventListener("error", function () {  
    console.log("Erro de rede");  
});
```

Código equivalente ao anterior, porém utilizando o `addEventListener`



# Outras Propriedades de Evento do XHR

- **onloadstart** – início do carregamento da resposta
- **onloadend** – término do carregamento da resposta
- **onprogress** – permite monitorar o carregamento
- **onreadystatechange** – permite monitorar o andamento da requisição
- **ontimeout** – tempo máximo para encerrar requisição excedido

```
let xhr = new XMLHttpRequest();  
xhr.timeout = 5000;  
xhr.ontimeout = function () {  
    ...  
};
```

# Propriedade `xhr.responseText`

- Permite especificar o tipo da resposta
- A resposta em si pode ser resgatada em `xhr.response`
- Alguns valores possíveis
  - `'text'` ou string vazia: valor padrão. Neste caso a resposta textual pode ser acessada em `xhr.response` ou `xhr.responseText`;
  - `'json'`: a string JSON retornada será analisada e convertida em objeto JavaScript, o qual estará disponível em `xhr.response`
  - `'blob'`: `xhr.response` será um objeto Blob contendo dados binários
  - `'document'`: `xhr.response` será um objeto Document ou XMLDocument

OBS 1: A propriedade `responseType` não pode ser alterada quando a requisição for síncrona fora de web workers.

OBS 2: Ao definir `responseType` para um determinado valor, o desenvolvedor deve certificar-se de que o servidor está realmente enviando uma resposta compatível com esse formato. Se o servidor retornar dados incompatíveis com o `responseType` definido, o valor de `xhr.response` será `null`.

# Requisição Ajax Retornando Imagem

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "imagemMuitoGrande.jpg");
xhr.responseType = "blob";

xhr.onload = function () {
    // recupera os dados binários da imagem
    const blob = xhr.response;

    // insere a imagem dinamicamente na página
    const img = document.createElement("img");
    img.src = window.URL.createObjectURL(blob);
    document.body.appendChild(img);
};

xhr.send();
```

Exemplo de uso das propriedades **responseType** e **response** para carregar imagem dinamicamente com Ajax

# Inserindo Imagem Dinamicamente sem Ajax

```
...  
const img = document.createElement("img");  
img.src = "images/imagemMuitoGrande.jpg";  
document.body.appendChild(img);  
...
```

Diferente do exemplo anterior, este código é executado de forma síncrona e irá "bloquear" a execução do JavaScript até a imagem ser carregada

# Requisição Ajax Retornando HTML como Objeto Document

Arquivo `conteudoAdicional.html`

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="UTF-8">
  <title>Uma página de exemplo</title>
</head>

<body>

  <h1>Programação para Internet</h1>
  <p>Olá, mundo!</p>

</body>

</html>
```

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "conteudoAdicional.html");
xhr.responseType = "document";

xhr.onload = function () {
  const doc = xhr.response;
  alert(doc.querySelector("h1").textContent);
};

xhr.send();
```

Mostrará o título "Programação para Internet"

# Requisição Ajax com Retorno em JSON

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "endereco.php?cep=38400-100");
xhr.responseType = 'json';

xhr.onload = function () {
    if (xhr.response === null) {
        console.log("Resposta não obtida");
        return;
    }

    const endereco = xhr.response;
    form.bairro.value = endereco.bairro;
    form.cidade.value = endereco.cidade;
};

xhr.send();
```

Ao definir **responseType** com o valor **'json'**, a string JSON retornada será automaticamente convertida em um objeto JS, que pode ser acessado pela propriedade **response**

**Mas atenção:** caso haja um erro na conversão da string JSON para o objeto JavaScript, não será possível identificar o erro em detalhes (via JS)

# Requisição Ajax com Retorno em JSON – Conversão Manual

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "endereco.php?cep=38400-100");
xhr.onload = function () {
    try {
        // JSON.parse converte string JSON em objeto JS
        var endereco = JSON.parse(xhr.responseText);
    }
    catch (e) {
        console.error("JSON inválido: " + xhr.responseText);
        return;
    }

    // insere os dados do endereço no formulário
    form.bairro.value = endereco.bairro;
    form.cidade.value = endereco.cidade;
};
xhr.send();
```

# Recapitulando o Formato JSON

- Formato para representação de dados de forma textual
- Independente de linguagem
- Muito utilizado para intercâmbio de dados
  - Por exemplo, na comunicação cliente / servidor
- Permite a serialização de dados
- Acrônimo para **JavaScript Object Notation**



# Recapitulando o Formato JSON

```
const strJSON = '{  
  "Disciplina" : "Programação para Internet",  
  "Carga Horária" : 60,  
  "Avaliações" : [ 30, 30, 40 ],  
  "Professor" : "Furtado"  
';
```

- Dados organizados em pares ( "*propriedade*" : *valor* )
- Nomes das propriedades devem usar aspas duplas
- Pares separados por vírgula
- **Objetos** são colocados entre **chaves**
- **Vetores** são colocados entre **colchetes**
- Valores das propriedades podem ser novos objetos

# Exemplo de Código PHP Produzindo JSON como Resposta

```
<?php
class Endereco {
    public $rua;
    public $bairro;
    public $cidade;

    function __construct($rua, $bairro, $cidade) {
        $this->rua = $rua;
        $this->bairro = $bairro;
        $this->cidade = $cidade;
    }
}

$cep = $_GET['cep'] ?? '';

if ($cep == '38400-100')
    $endereco = new Endereco('Av Floriano', 'Centro', 'Uberlândia');
else if ($cep == '38400-200')
    $endereco = new Endereco('Av Tiradentes', 'Fundinho', 'Uberl');
else
    $endereco = new Endereco('', '', '');

header('Content-type: application/json');
echo json_encode($endereco); // converte objeto PHP em string JSON
```

*Exemplo de uso da função `json_encode` do PHP para converter um objeto da linguagem em uma string **JSON** correspondente, que é enviada como resposta ao cliente.*

# Recuperando Dados de Cabeçalho da Resposta

## `xhr.getResponseHeader`

- retorna uma informação de cabeçalho específica da resposta HTTP

## `xhr.getAllResponseHeader`

- retorna string com todas as informações de cabeçalho da resposta HTTP

```
...  
xhr.onload = function () {  
    let contType = xhr.getResponseHeader("Content-Type");  
    if (contType !== "application/json")  
        return;  
};  
...
```

# Exemplo de Requisição Enviando JSON

```
// objeto JavaScript a ser enviado
let objetoJS = {
  modelo : "Fusca",
  ano : "1970"
};

let xhr = new XMLHttpRequest();
xhr.open("POST", "cadastra.php");

xhr.onload = function () { ... }

// define cabeçalho HTTP 'Content-Type' para envio de JSON
xhr.setRequestHeader("Content-Type", "application/json");

// JSON.stringify converte um objeto JS para uma string JSON
xhr.send(JSON.stringify(objetoJS));
```

*setRequestHeader* deve ser chamada **depois** do método *open* e **antes** do método *send*

# Submetendo Formulários com o XHR

---

# Submetendo Formulários com o XHR

- Há duas formas de submeter um formulário com o XHR
  1. Utilizando JavaScript puro (não será apresentada)
  2. Utilizando a API `FormData`

# Submetendo Formulário com FormData

```
// cria-se um objeto FormData utilizando o objeto do formulário
let meuForm = document.querySelector("meuForm");
let formData = new FormData(meuForm);

let xhr = new XMLHttpRequest();
xhr.open("POST", "cadastra.php");
xhr.send(formData); // envia-se o objeto utilizando o método send
```

# Submetendo Formulário com FormData

Acrescentando campos com o método **append**

```
let meuForm = document.querySelector("form");  
let formData = new FormData(meuForm);  
  
formData.append("id", "123456");  
  
let xhr = new XMLHttpRequest();  
xhr.open("POST", "cadastra.php");  
xhr.send(formData);
```



# Enviando Dados por POST com FormData

```
let formData = new FormData();  
formData.append("modelo", "Fusca");  
formData.append("ano", "1970");  
  
let xhr = new XMLHttpRequest();  
xhr.open("POST", "cadastra.php");  
xhr.send(formData);
```

1. Cria-se um objeto **FormData**
2. Adiciona-se dados (**nome**, **valor**) com o método **append**
3. Utiliza-se **POST** em **xhr.open** e envia com **xhr.send**

# Enviando Dados por POST sem o FormData

## Conteúdo Complementar

```
let xhr = new XMLHttpRequest();  
xhr.open("POST", "cadastra.php");  
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
xhr.send("modelo=Fusca&ano=1970");
```

1. Utiliza-se o método **POST** em `xhr.open`
2. Utiliza-se o `setRequestHeader` para alterar o cabeçalho da requisição
3. Envia-se os dados pelo método `send` na forma de uma string/URL

*Codificação adicional pode ser necessária dependendo dos caracteres da string de dados.  
Funções adicionais como **encodeURIComponent** podem ser necessárias.*

# Propriedade onreadystatechange

```
let xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {
    if (this.readyState === this.DONE) {
        console.log(this.responseText);
    }
};

xhr.onerror = function () {
    console.log("Erro de rede");
};

xhr.open("GET", "busca.php");
xhr.send();
```

A propriedade **onreadystatechange** permite monitor o andamento da requisição. O valor de **readyState** varia de 0 (início) até 4 (término).

# Referências

- [github.com/public-apis/public-apis](https://github.com/public-apis/public-apis)
- [rapidapi.com/collection/list-of-free-apis](https://rapidapi.com/collection/list-of-free-apis)
- <https://xhr.spec.whatwg.org>
- <https://www.ecma-international.org/ecma-262/>
- <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>
- [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
- <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous/Concepts>
- Jasse J. Garrett. **Ajax: A New Approach to Web Applications**, Adaptive Path, 2005.
- David Flanagan. **JavaScript: The Definitive Guide**. 7ª ed., 2020.
- Jon Duckett. **JavaScript and JQuery: Interactive Front-End Web Development**.