# Programação para Internet

## Módulo 8 – Gestão da Informação

## Requisições HTTP Assíncronas e a Técnica Ajax

(Mensagens HTTP, JSON, Técnica Ajax e conceitos, JavaScript Assíncrono, XMLHttpRequest, JavaScript Promises, API Fetch, Async/await)

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

# Conteúdo do Módulo

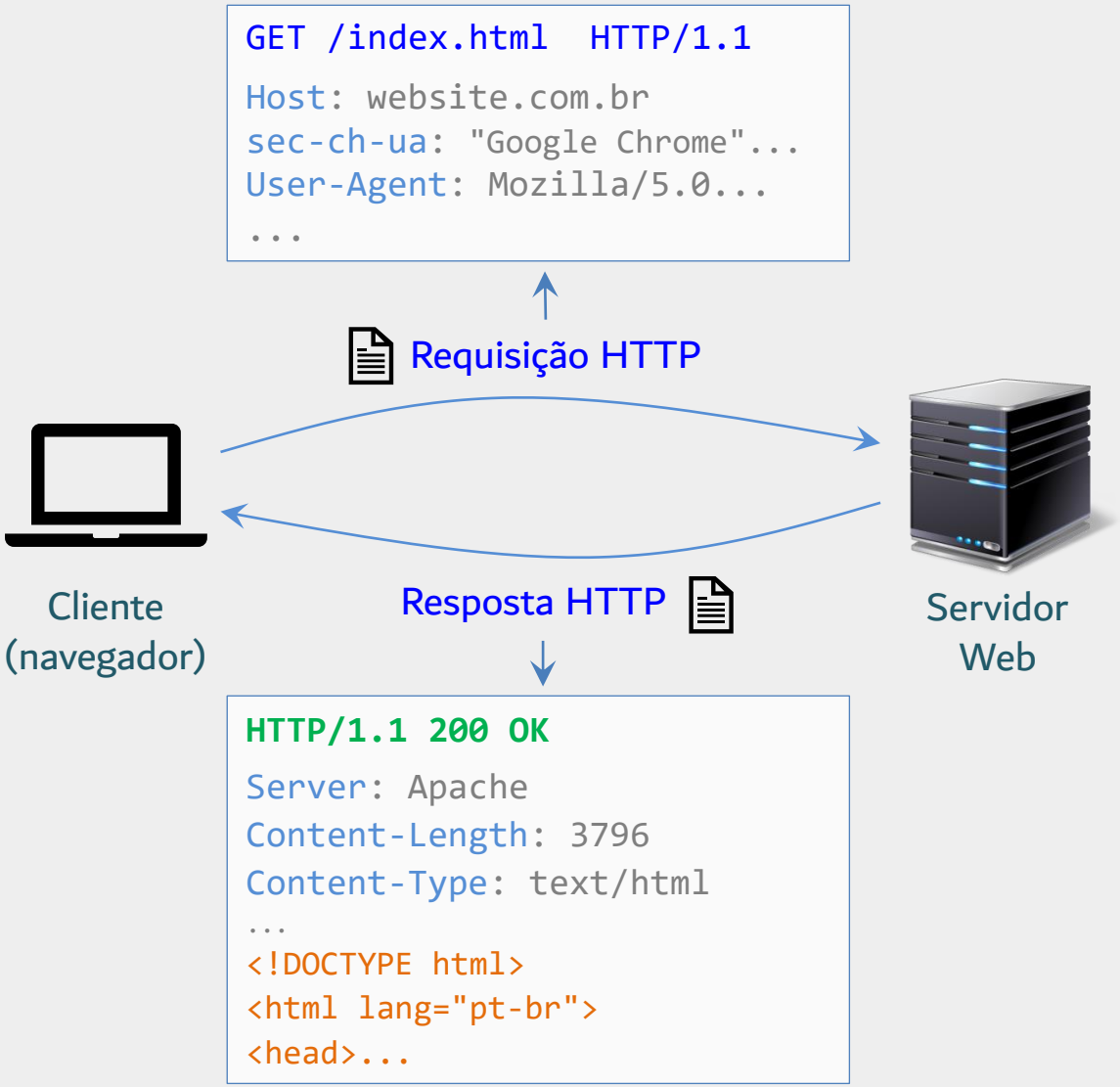
- Parte 1
  - Requisições e Respostas HTTP
  - Recapitulando o formato JSON
- Parte 2
  - Técnica Ajax e conceitos relacionados
  - Requisições assíncronas com o XMLHttpRequest
  - Tratamento de erros: rede x aplicação
  - Requisições com retorno em JSON
  - Submissão assíncrona de formulários
- Parte 3
  - Introdução à requisições assíncronas com à API Fetch

# Requisições e Respostas HTTP
# Introdução

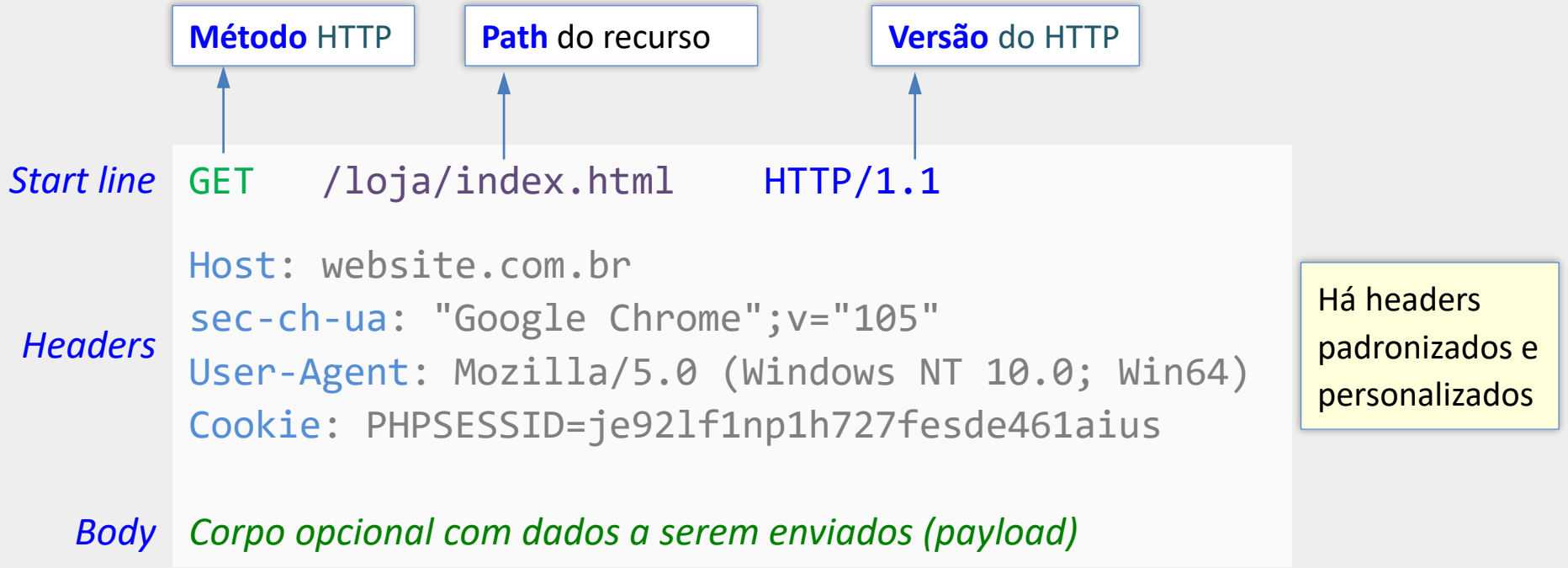
# Requisições e Respostas HTTP

- Conceito crucial para entendimento da técnica Ajax
- A comunicação entre o navegador de internet e o servidor web ocorre por meio de uma série de **requisições** e **respostas** HTTP
- Uma requisição HTTP consiste essencialmente de uma mensagem contendo uma série de parâmetros que o navegador envia ao servidor para acessar recursos como arquivos HTML, arquivos de imagens etc.
- O servidor web, por sua vez, responde à requisição enviando de volta uma resposta HTTP, que contém o recurso solicitado juntamente com metadados

# Requisições e Respostas HTTP



# Exemplo Simplificado de Mensagem de **Requisição HTTP**




Uma mensagem de **requisição HTTP** é organizada em três partes:
1) a primeira linha (**start line** ou **request line**), que contém o método HTTP a ser utilizado e o caminho do recurso no servidor sendo requisitado;
2) as linhas de cabeçalho (**headers**) contendo parâmetros adicionais como a identificação do navegador de internet, dados de cookies etc.;
3) o corpo da requisição (**body**), que é a região da mensagem que pode conter dados da aplicação a serem enviados juntamente com a requisição (como dados de formulários, arquivos etc.)

# Exemplo Simplificado de Mensagem de **Requisição HTTP**

*Start line*

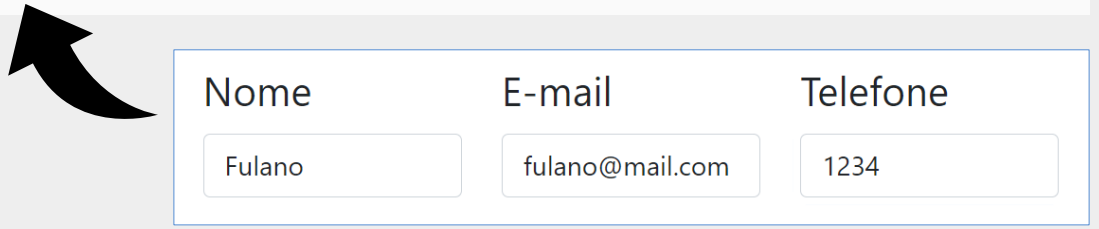
```
POST    /loja/cadastro.php    HTTP/1.1
```

*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
```

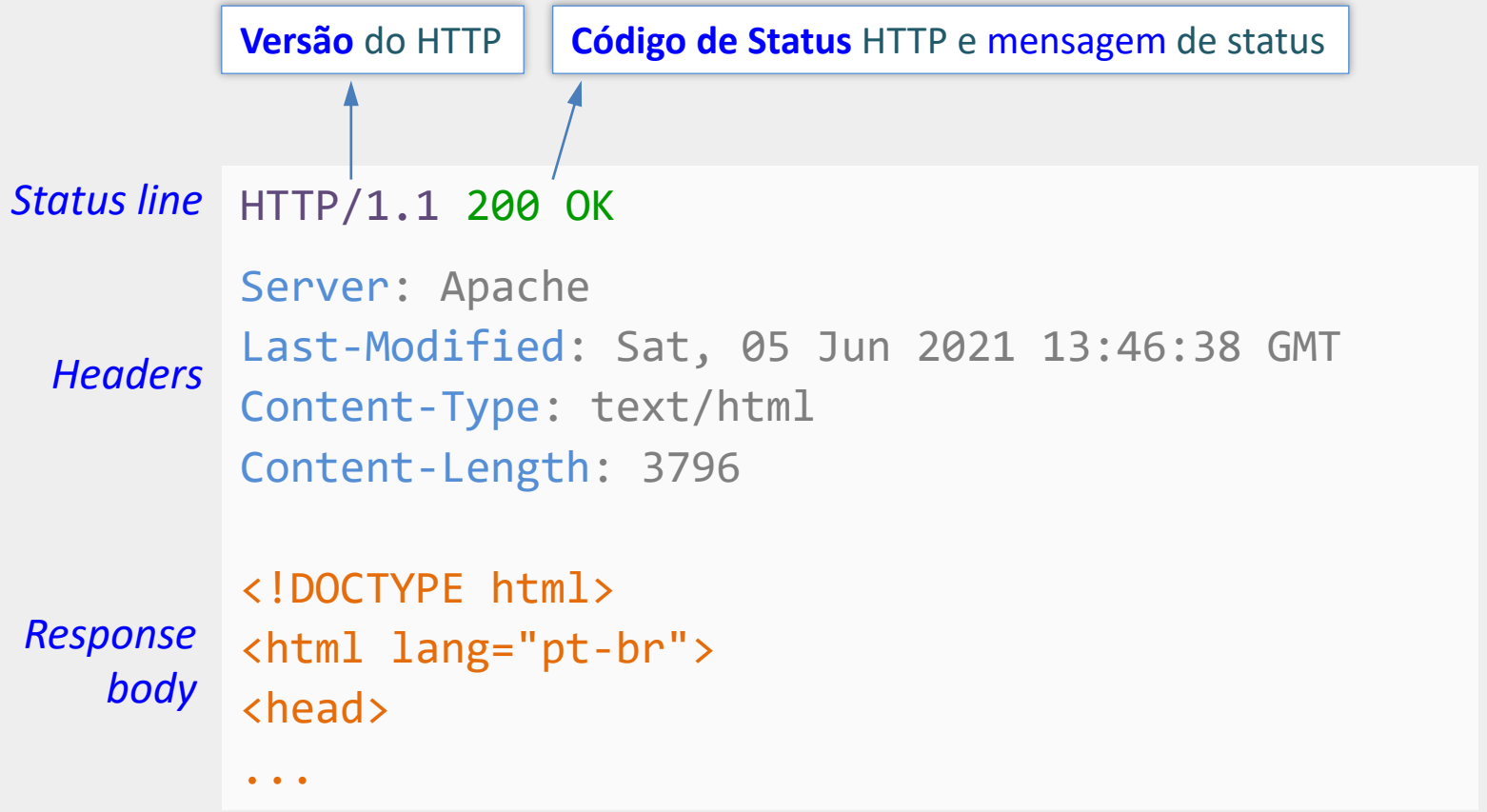
*Body*

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




Quando um formulário HTML é submetido pelo método **POST**, por exemplo, o navegador envia uma requisição HTTP carregando os dados do formulário no **corpo** da mensagem de requisição. Quando o formulário é submetido pelo método **GET**, tais dados são enviados pela própria URL, na primeira linha (**start line**) da mensagem de requisição.

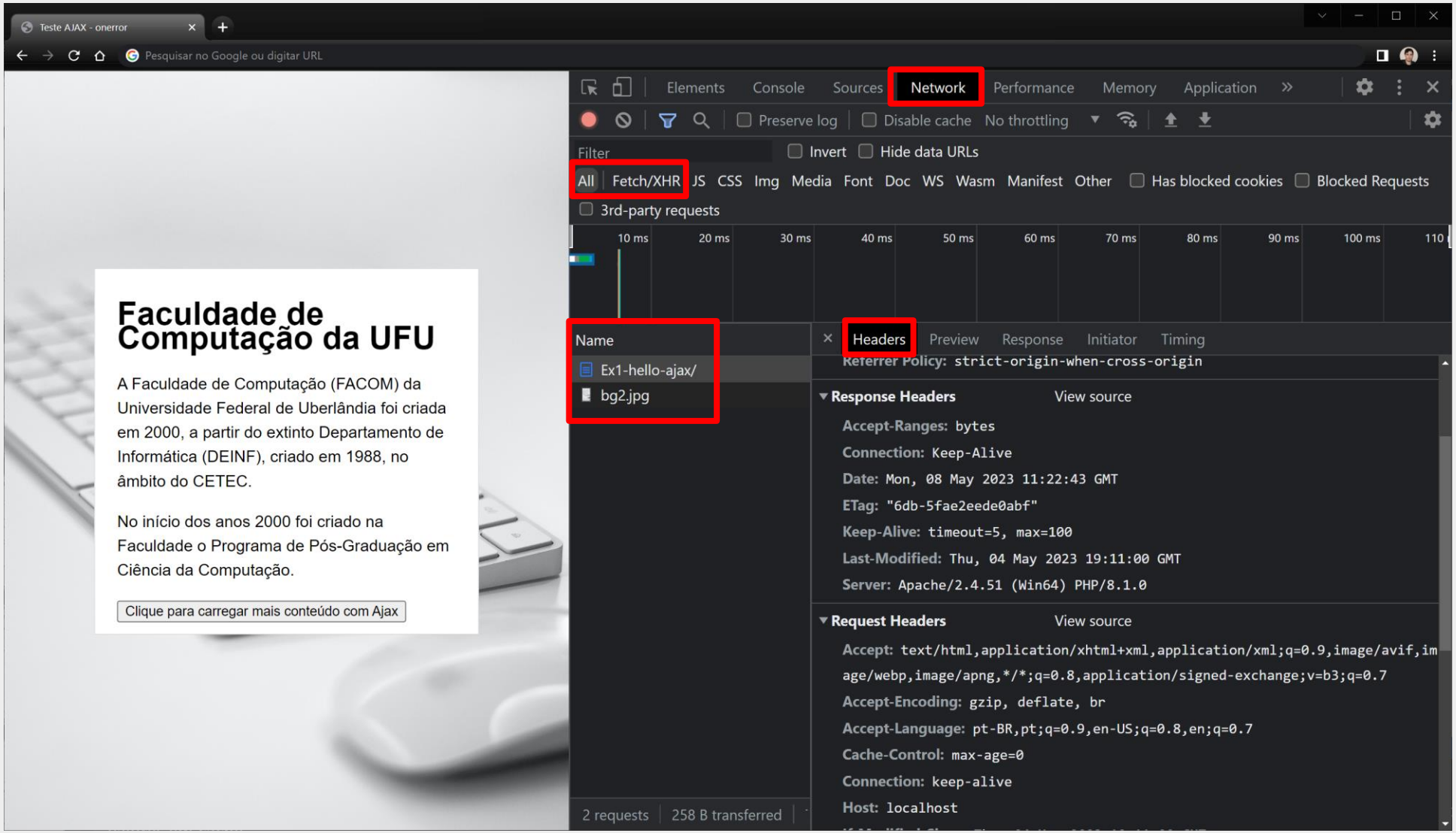
# Exemplo Simplificado de Mensagem de **Resposta HTTP**




Uma mensagem de **resposta HTTP** também é organizada em três partes, de maneira similar à mensagem de requisição. A **linha de status** contém o código de status HTTP, que pode indicar sucesso ou falha na obtenção do recurso solicitado. Os **cabeçalhos** contêm metadados identificando, por exemplo, o tipo de conteúdo sendo retornado, o tamanho desse conteúdo e o servidor HTTP que produziu o conteúdo. O **corpo** normalmente contém o recurso que foi solicitado na requisição como código HTML, string JSON, dados de arquivos de imagem etc.

# Verificando Dados de uma Requisição HTTP no Navegador

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



# 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
- 403 Forbidden – acesso ao recurso não autorizado
- 404 Not Found – recurso não encontrado
- 500 Internal Server Error – erro interno no servidor

# Introdução ao Formato JSON

# Introdução ao Formato JSON

- Acrônimo para **J**ava**S**cript **O**bject **N**otation
- É um formato para representação de dados de forma textual
- Por ser textual, é independente de linguagem
- Muito utilizado para intercâmbio de dados
  - Por exemplo, na comunicação cliente / servidor
- Permite a serialização de dados

# Introdução ao Formato JSON

- Os dados são organizados em pares do tipo "propriedade" : valor, separados por vírgula;
- Os nomes das propriedades devem usar aspas duplas
- Objetos são representados utilizando chaves { }
- Vetores são representados utilizando colchetes [ ]
- Os valores das propriedades podem ser novos objetos, definidos com chaves

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

# Exemplo de Script PHP Retornando Dados em JSON

```php
<?php
require 'conexaoMysql.php';
$pdo = mysqlConnect();

$sql = <<<SQL
  SELECT nome, telefone
  FROM aluno
  LIMIT 2
SQL;

try {
  $stmt = $pdo->query($sql);
  $arrayAlunos = $stmt->fetchAll(PDO::FETCH_OBJ);
  header('Content-Type: application/json; charset=utf-8'); // dados de cabeçalho da resposta HTTP
  echo json_encode($arrayAlunos);  // converte o array de alunos em uma string JSON
}
catch (Exception $e) {
  exit('Falha inesperada: ' . $e->getMessage());
}
```

Este script de exemplo retorna os 2 primeiros registros da tabela **aluno** como um *array* de objetos no formato **JSON**.

Repare que a função header do PHP foi utilizada para definição adequada do cabeçalho **Content-Type** da resposta HTTP para o valor **application/json**, uma vez que será retornado um conteúdo no formato JSON.

O *array* de objetos é convertido em uma string JSON correspondente utilizando a função **json_encode**. Exemplo de saída produzida:
[ {"nome": "Fulano", "telefone": "123"}, {"nome": "Ciclano", "telefone": "456"} ]

# Exemplo de Script PHP Retornando Dados em JSON

```php
<?php
require 'conexaoMysql.php';
$pdo = mysqlConnect();
$cep = $_GET['cep'] ?? '';

$sql = <<<SQL
  SELECT rua, bairro, cidade
  FROM BaseEnderecos
  WHERE cep = ?
SQL;

try {
  $stmt = $pdo->prepare($sql);
  $stmt->execute([$cep]);
  $endereco = $stmt->fetch(PDO::FETCH_OBJ);
  header('Content-Type: application/json; charset=utf-8');
  echo json_encode($endereco);
}
catch (Exception $e) {
  exit('Falha inesperada: ' . $e->getMessage());
}
```

Script simplificado que recebe um CEP pela URL, busca por ele em tabela do banco de dados e retorna os dados do endereço no formato JSON.

Observe que o registro é recuperado como um objeto PHP, pois foi utilizada a opção *PDO::FETCH_OBJ* na chamada do método fetch.

Exemplo de resposta produzida:
{ "rua": "Av Floriano", "bairro": "Centro", "cidade": "Uberlândia" }

# 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
- Dois exemplos eram o Google Suggest (hoje chamado de Google Autocomplete) e o Google Maps
- Tais aplicações realizavam atualizações na página em segundo plano, de maneira quase instantânea, à medida que o usuário interagia com os elementos da interface

# O que é a Técnica Ajax

- Ajax é uma técnica para realizar atualizações incrementais na página web
  - Permite atualizar uma 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)
  - Não interrompe a navegação do usuário
  - Não "congela" a interface
- **Objetivo**: aplicação mais ágil, eficiente e melhor experiência do usuário

# Surgimento do Termo Ajax em si

adaptive path

**Ajax: A New Approach to Web Applications**

*Jesse James Garrett is a founder of Adaptive Path*

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.

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.

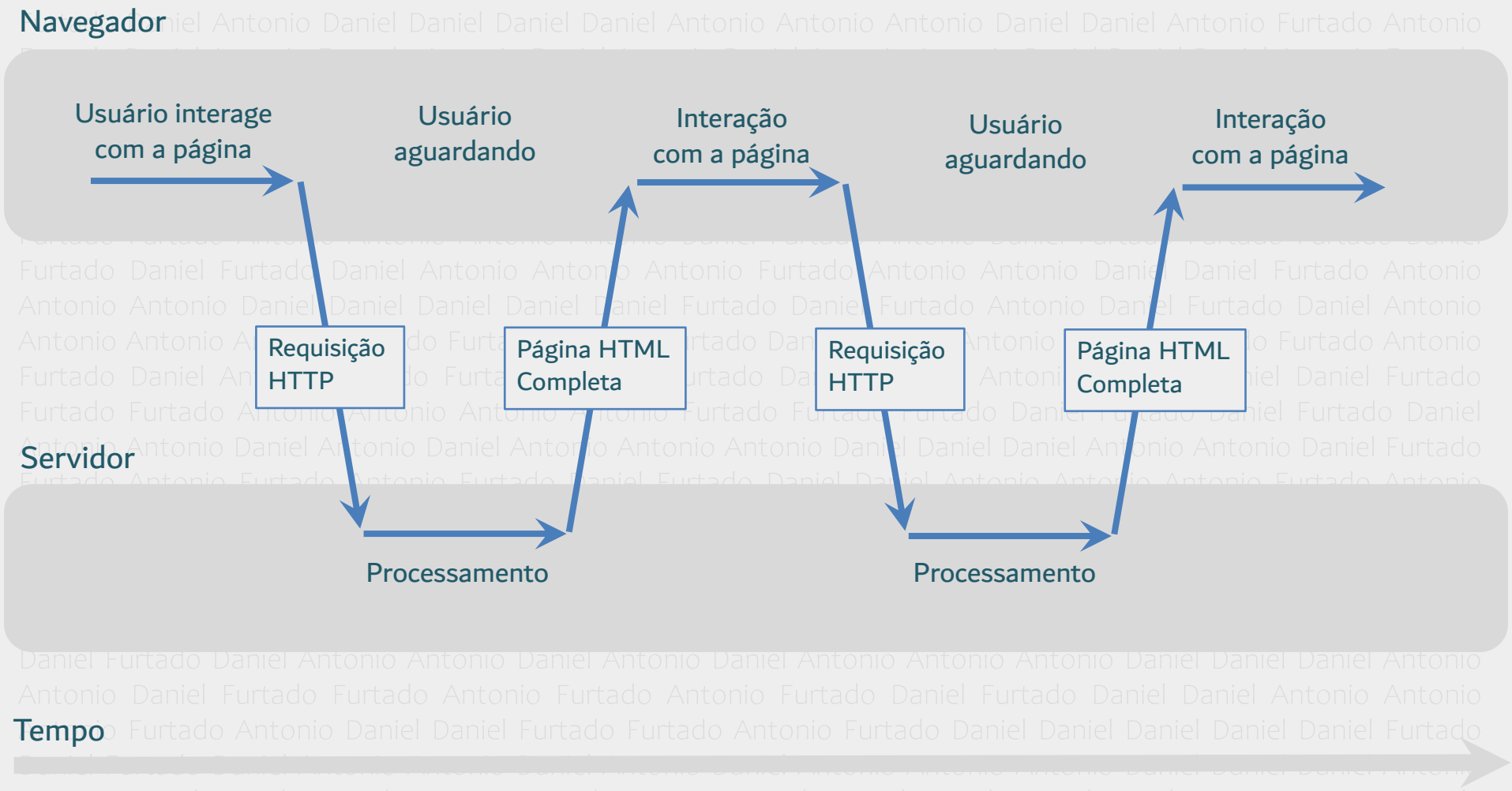
Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what's possible on the Web.

**Defining Ajax**

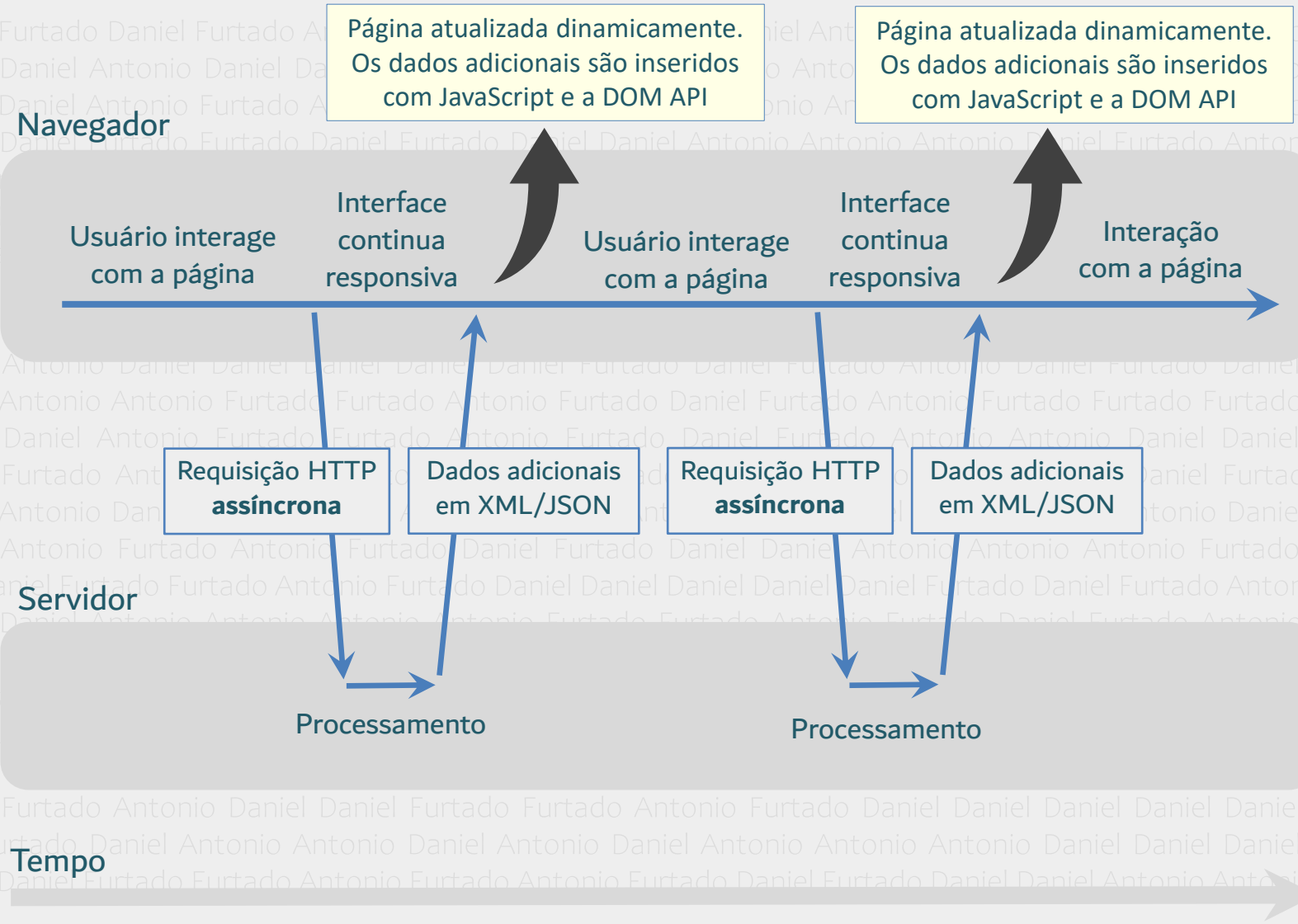
Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

- O termo Ajax foi proposto em 2005 pelo canadense 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

# Aplicação Web Convencional (Sem Ajax)



# Aplicação Web com Ajax



# Exemplo

CEP

38408-100

Endereço

Av João Naves de Ávila

Bairro

Santa Mônica

Cidade

Uberlândia

Estado

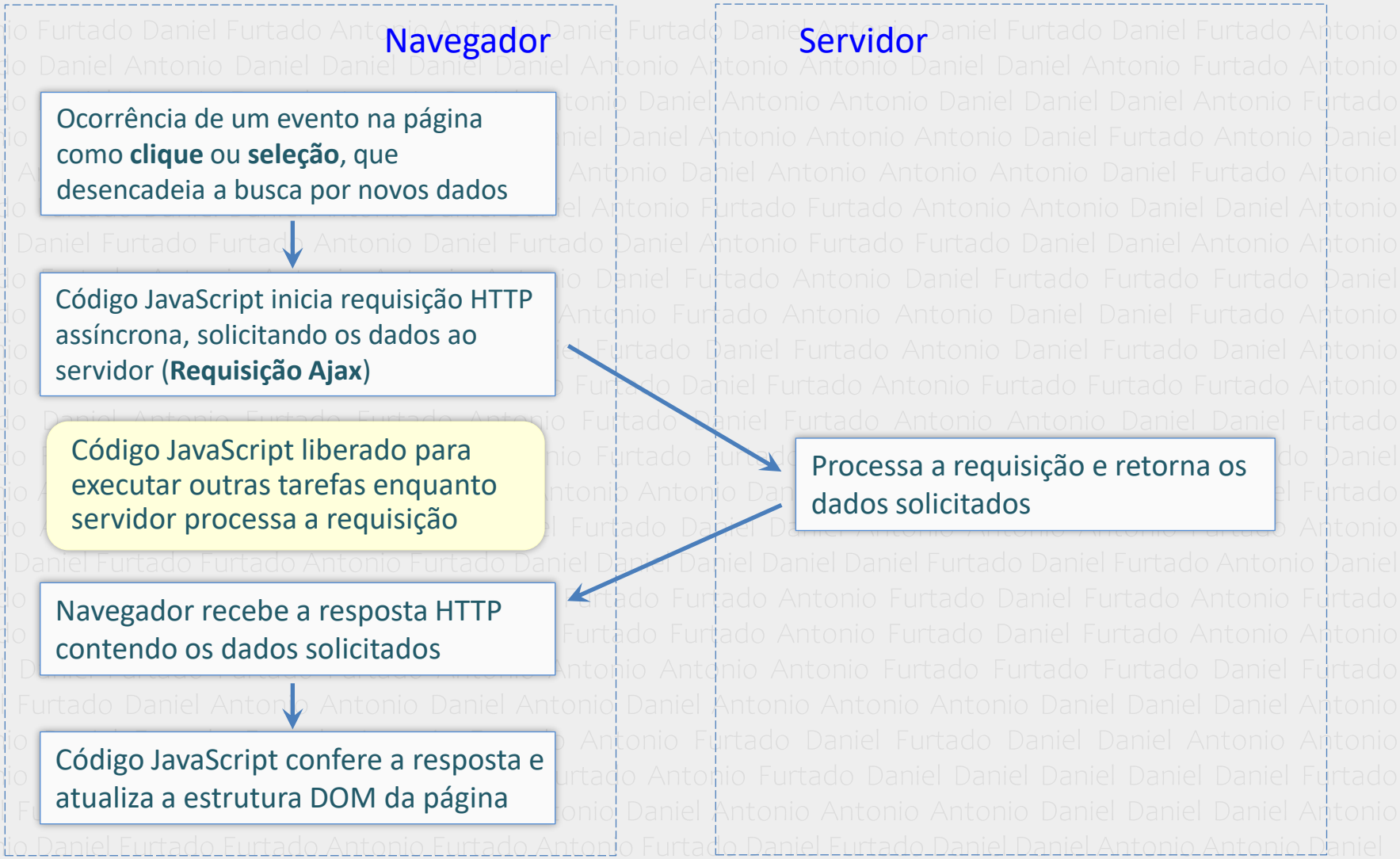
Minas Gerais

Preenchimento automático do formulário assim que o usuário digita o CEP

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


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

# Sequência de Eventos Envolvidos na Técnica Ajax



# Informações Adicionais sobre Ajax

- Ajax não é linguagem de programação, API ou biblioteca
- 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 = **A**synchronous **Ja**vaScript and **X**ML

- 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
    - Os elementos HTML podem ser gerados no próprio no navegador
    - O conteúdo em si pode vir em JSON
    - Traz o esforço de renderização para o lado cliente
- Buscas instantâneas oferecidas por lojas virtuais
- Rolagem infinita da página
  - Redes sociais, listagem de produtos etc.

# Como realizar Requisições Ajax com JavaScript?

Nativo

- XMLHttpRequest
- API Fetch

Bibliotecas

- jQuery
- Axios

# Ajax com o XMLHttpRequest

# Objeto XMLHttpRequest (XHR)

- Projetado para buscar conteúdo em XML via requisições HTTP assíncronas
- Mas também suporta outros formatos como JSON, texto, HTML etc.
- Amplamente suportado pelos navegadores
- É uma API da Web - não faz parte da JavaScript em si
- Código mais longo, mas conceitualmente mais simples
  - Não utiliza Promises (o XHR se baseia em funções de callback)
  - Fácil aprendizado
- Dificuldades
  - Encadeamento de várias requisições de difícil manutenção (callback hell)

# Principais Passos para Iniciar Requisição Ajax com o XHR

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

# Exemplo Simples de Requisição Ajax sem Tratamento de Erros

O método **open** configura a requisição HTTP:
- O 1º parâmetro indica o método HTTP a ser utilizado (GET, POST, PUT etc.)
- O 2º parâmetro é o endereço no servidor do recurso sendo solicitado (arquivo, script etc.). Pode ser um caminho relativo ou URL completa.
- O 3º parâmetro indica se a requisição deve ser realizada de forma assíncrona (true) ou síncrona (false). Se omitido, será assíncrona (padrão).

Criação do objeto XMLHttpRequest para iniciar requisição Ajax

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

Neste exemplo a propriedade **responseText** é utilizada para acessar a resposta textual enviada pelo servidor (conteúdo do arquivo **dados.txt**).

A propriedade **onload** permite indicar uma função de callback que será chamada automaticamente quando a resposta enviada pelo servidor terminar de ser carregada pelo navegador. É a função que utilizará os dados requisitados. Deve ser indicada antes do envio da requisição.

Envia a requisição HTTP. No caso de requisição assíncrona, o código JavaScript prosseguirá normalmente enquanto a requisição será tratada em segundo plano.

# Requisição Assíncrona x Síncrona com o XHR

## Requisição Assíncrona

- O código JavaScript prossegue enquanto requisição é gerenciada pelo navegador em segundo plano (outra thread)
- 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 (fora de Web Workers)

# Exemplo de Requisição Ajax Buscando Conteúdo Adicional

```html
<main>
  <h1>Faculdade de Computação da UFU</h1>
  <p>A Faculdade de Computação (FACOM) da Universidade...</p>
  <p>No início dos anos 2000 foi criado na Faculdade...</p>
  <button>Clique para carregar mais conteúdo com Ajax</button>
</main>

<script>
  function loadExtraContent() {
    let xhr = new XMLHttpRequest();
    xhr.open("GET", "conteudoAdicional.html", true);
    xhr.onload = function () {
      const main = document.querySelector("main");
      main.insertAdjacentHTML("beforeend", xhr.responseText);
    };
    xhr.send();
  }
  const button = document.querySelector("button");
  button.addEventListener("click", loadExtraContent);
</script>
```

Arquivo **conteudoAdicional.html**

```html
<h2>Cursos de Graduação</h2>
<p>A FACOM oferece os cursos de Bacharelado...</p>

<h2>Cursos de Pós-Graduação</h2>
<p>A oferece também os cursos de
  Mestrado Acadêmico e Doutorado em
  Ciência da Computação.</p>

<address>
  Campus Santa Mônica, Bloco 1A<br>
  Av. João Naves de Ávila, 2121, Santa Mônica<br>
  Uberlândia, MG<br>
  CEP 38400-902
</address>
```

# 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.

# XMLHttpRequest com JSON
# Tratamento Adequado da Resposta
# com Definição de xhr.responseType

# Propriedade xhr.responseType

- A propriedade xhr.responseType possibilita especificar o tipo da resposta esperada do servidor, permitindo que os dados recebidos sejam tratados de maneira adequada pelo JavaScript
- Por exemplo, ao definir xhr.responseType = 'json', os dados retornados pelo servidor no formato json serão automaticamente convertidos em um objeto JavaScript correspondente, que estará disponível em xhr.response
- Outros valores comuns para xhr.responseType:
  - 'text' ou string vazia: valor padrão. Neste caso a resposta textual pode ser acessada em xhr.response ou xhr.responseText;
  - '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 com Retorno em JSON

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "endereco.php?cep=38400-100");
xhr.responseType = 'json';
xhr.onload = function () {
  if (xhr.status != 200 || xhr.response === null) {
    console.log("Resposta não obtida");
    return;
  }
  const endereco = xhr.response;
  let form = document.querySelector("#meuForm");
  form.bairro.value = endereco.bairro;
  form.cidade.value = endereco.cidade;
};
xhr.onerror = function () {
  console.error("Requisição não finalizada");
  return;
};
xhr.send();
```

Ao definir xhr.**responseType** com o valor 'json', a string JSON retornada pelo servidor será automaticamente convertida em um objeto JavaScript correspondente, que poderá ser resgatado pela propriedade xhr.**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 JavaScript. Porém o desenvolvedor pode utilizar o ambiente de desenvolvimento do navegador para verificar o **corpo** da resposta HTTP com eventual mensagem de erro.

# 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();
```

Este exemplo ilustra uma requisição Ajax para buscar conteúdo em JSON sem utilizar xhr.responseType = 'JSON'. Observe que neste caso a conversão da string JSON para um objeto JavaScript precisa ser feita manualmente utilizando a função JSON.parse.

# 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("#meuFormulario");
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
```

# Parte 2
# Requisições Assíncronas com a API Fetch

# API Fetch

- Outra forma de realizar requisições Ajax
- Mais nova que o XMLHttpRequest
- Maior facilidade para encadear tarefas assíncronas (evitando callback hell)
- Maior clareza e simplicidade quando utilizada com async / await
- Utiliza o conceito de promise do JavaScript

# Callback Hell

```
let xhr1 = new XMLHttpRequest();
xhr1.onload = function () {
   let xhr2 = new XMLHttpRequest();
   xhr2.onload = function () {
      let xhr3 = new XMLHttpRequest();
      xhr3.onload = function () {
         let xhr4 = new XMLHttpRequest();
         xhr4.onload = function () {
            console.log(xhr4.response);
         };
      };
   };
};
```

Este exemplo ilustra um possível encadeamento de requisições Ajax utilizando o XMLHttpRequest. Repare que há diversas chamadas em cascata de funções de callback (callback hell), tornando o código complexo e de difícil manutenção. O conceito de **promise** em conjunto com a API **Fetch** permite evitar esta situação.

# Evitando Callback Hell

```
1.  let xhr1 = new XMLHttpRequest();
2.  xhr1.open("GET", "URL1");
3.  xhr1.responseType = 'json';
4.  xhr1.onload = function () {
5.      const data1 = xhr1.response;
6.      let xhr2 = new XMLHttpRequest();
7.      xhr2.open("GET", "URL2");
8.      xhr2.responseType = 'json';
9.      xhr2.onload = function () {
10.         const data2 = xhr2.response;
11.         let xhr3 = new XMLHttpRequest();
12.         xhr3.open("GET", "URL3");
13.         xhr3.responseType = 'json';
14.         xhr3.onload = function () {
15.             const data3 = xhr3.response;
16.             console.log(data3);
17.         }
18.         xhr3.onerror = function () {
19.             console.error("Erro de rede XHR3");
20.         };
21.         xhr3.send();
22.     }
23.     xhr2.onerror = function () {
24.         console.error("Erro de rede XHR2");
25.     };
26.     xhr2.send();
27. }
28. xhr1.onerror = function () {
29.     console.error("Erro de rede XHR1");
30. };
31. xhr1.send();
```

Encadeando Requisições com o XHR

```
function getJSON(URL) {
   return fetch(URL)
      .then(response => response.json());
}

async function getData() {
   try {
      let data1 = await getJSON('URL1');
      let data2 = await getJSON('URL2');
      let data3 = await getJSON('URL3');
      console.log(data3);
   }
   catch (error) {
      console.error(error);
   }
}
```

Código equivalente com Fetch e async/await
(Será apresentado em detalhes ao longo deste material)

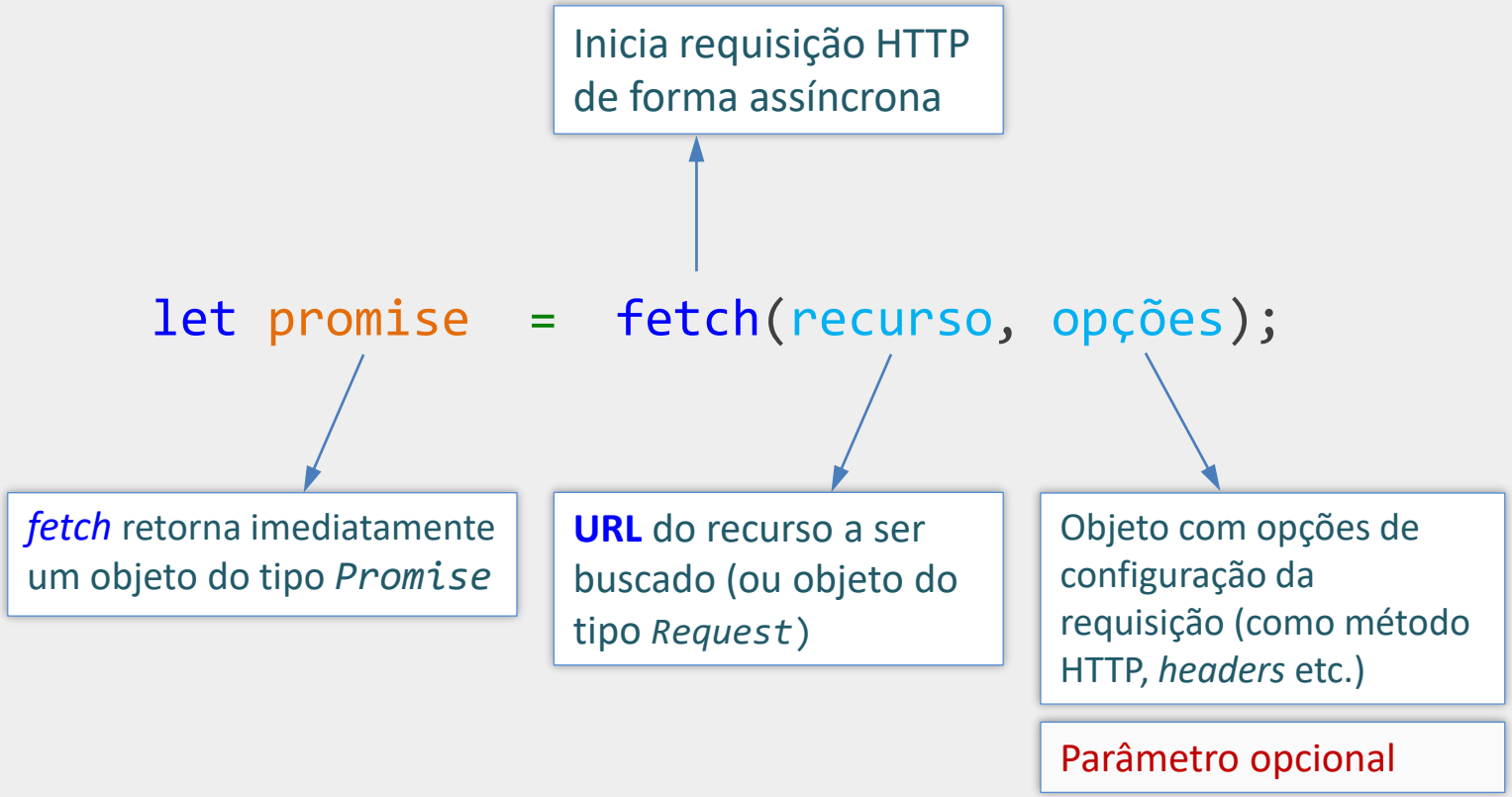
# Introdução à Promises

- Promises em JavaScript simplificam o uso de **métodos assíncronos**
- Métodos assíncronos são executados em **segundo plano** (em outra thread)
  - Portanto, não retornam um valor final imediatamente
  - Mas retornam imediatamente um objeto do tipo promise, representando uma "promessa" de fornecer o valor final no futuro
- Em outras palavras, uma promise é um objeto que representa uma tarefa assíncrona a ser finalizada no futuro

# Introdução à Promises

- Se a tarefa assíncrona finalizar com **sucesso**, a promise será **cumprida** e produzirá um **valor** (resultado)
- Se a tarefa assíncrona finalizar com **falha**, a promise será **rejeitada** e produzirá um **motivo** (erro)
- Para processar o resultado ou tratar o erro, podem ser indicadas funções de callback utilizando o método then() da promise ou utilizando os recursos async / await da JavaScript

# Introdução à Promises - Método fetch



# async/await

- Como um recurso relativamente novo da JavaScript, os termos async / await possibilitam que funções assíncronas sejam chamadas com sintaxe similar às síncronas
- Utiliza-se o termo async para definir novas funções assíncronas, e o termo await, dentro dessas funções, para chamar outras funções assíncronas

# Tratamento de Erros com async/await – Exemplo

```
async function buscaEndereço(cep) {
  try {
    const response = await fetch("endereco.php?cep=" + cep);
    if (! response.ok)
      throw new Error("Falha inesperada: " + response.status);

    const endereco = await response.json();
    console.log(endereco);
  }
  catch (e) {
    console.error(e);
  }
}
```

O termo await antes de fetch faz com que a função buscaEndereco seja suspensa até que a requisição assíncrona seja finalizada (e retorne os dados do endereço). Quando isso ocorrer, a execução de buscaEndereço é retomada e a próxima linha é executada para verificar se o servidor eventualmente retornou um código de status que não seja de sucesso, lançando uma exceção.
Caso contrário o código prossegue a o método response.json() é chamado para fazer a leitura dos dados no formato JSON retornados pelo servidor e convertê-los em objeto JavaScript correspondente. Esse método também é executado de forma assíncrona e por isso é necessário o await. Quando a leitura e conversão terminar, em segundo plano, a execução é retomada e o endereço é mostrado no console do navegador.

# Encadeamento de Requisições com fetch e async/await

```
async function buscaClimaLocal() {
  try {
    // busca a latitude e longitude local
    const response1 = await fetch('https://ipapi.co/json/');
    if (! response1.ok) throw new Error(response1.statusText);
    local = await response1.json();

    // busca informações do clima local passando a latitude e a longitude como parâmetro
    const response2 = await fetch(`https://api.open-meteo.com/v1/forecast?latitude=${local.latitude}&lon
    if (! response2.ok) throw new Error(response2.statusText);
    clima = await response2.json();

    // apresenta as informações do clima
    document.getElementById("temp").textContent = clima.current_weather.temperature + 'º';
    document.getElementById("wind").textContent = clima.current_weather.windspeed + ' km/h';
  }
  catch (error) {
    console.log(error);
    alert('Não foi possível obter a temperatura local');
  }
}
```

# Referências


- 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/JavaScript/Guide/Using_promises
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous/Concepts
- https://github.com/public-apis/public-apis
- https://rapidapi.com/collection/list-of-free-apis
- 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**.