Programação para Internet

Módulo 7 – Gestão da Informação

Introdução à Técnica Ajax

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

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();
$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.

Em seguida, o objeto PHP é convertido em uma string JSON utilizando a função json_encode do PHP. Um exemplo de resposta produzida seria:
{ "rua": "Av Floriano", "bairro": "Centro", "cidade": "Uberlândia" }

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 retornando apenas uma string com o nome da rua

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

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

try {
  $stmt = $pdo->prepare($sql);
  $stmt->execute([$cep]);
  $row = $stmt->fetch();
  $rua = $row['rua'];
  echo $rua;
}
catch (Exception $e) {
  exit('Falha inesperada: ' . $e->getMessage());
}
```

Este script de exemplo retorna apenas o nome da rua vinculada ao CEP passado pela URL. Repare que o nome da rua é retornado como uma simples string, sem a necessidade de utilizar o formato JSON. O exemplo não trata todos os erros e pressupõe a existência do CEP informado.

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.

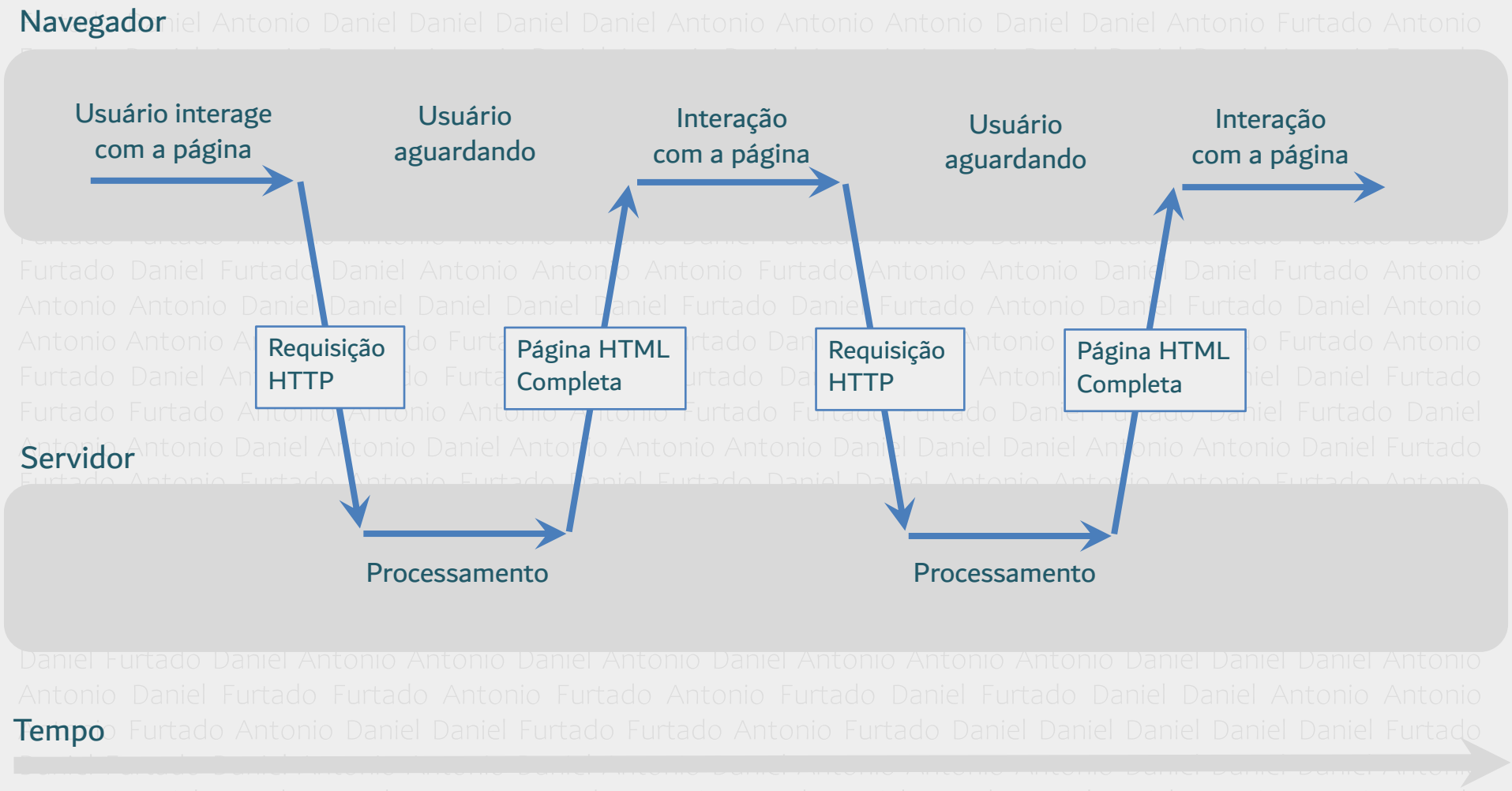
Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what's possible on the Web.

Defining Ajax

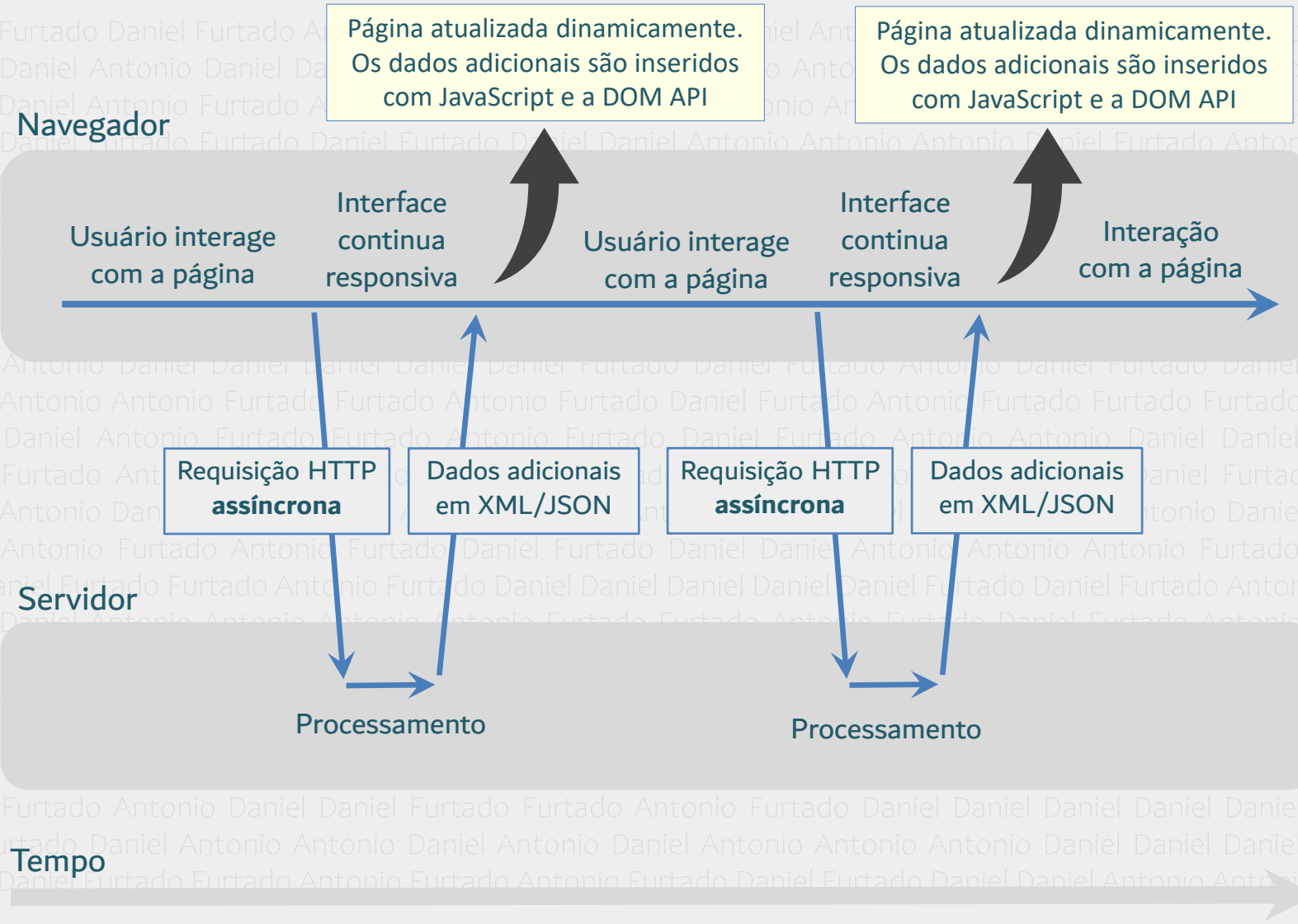
Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

- 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

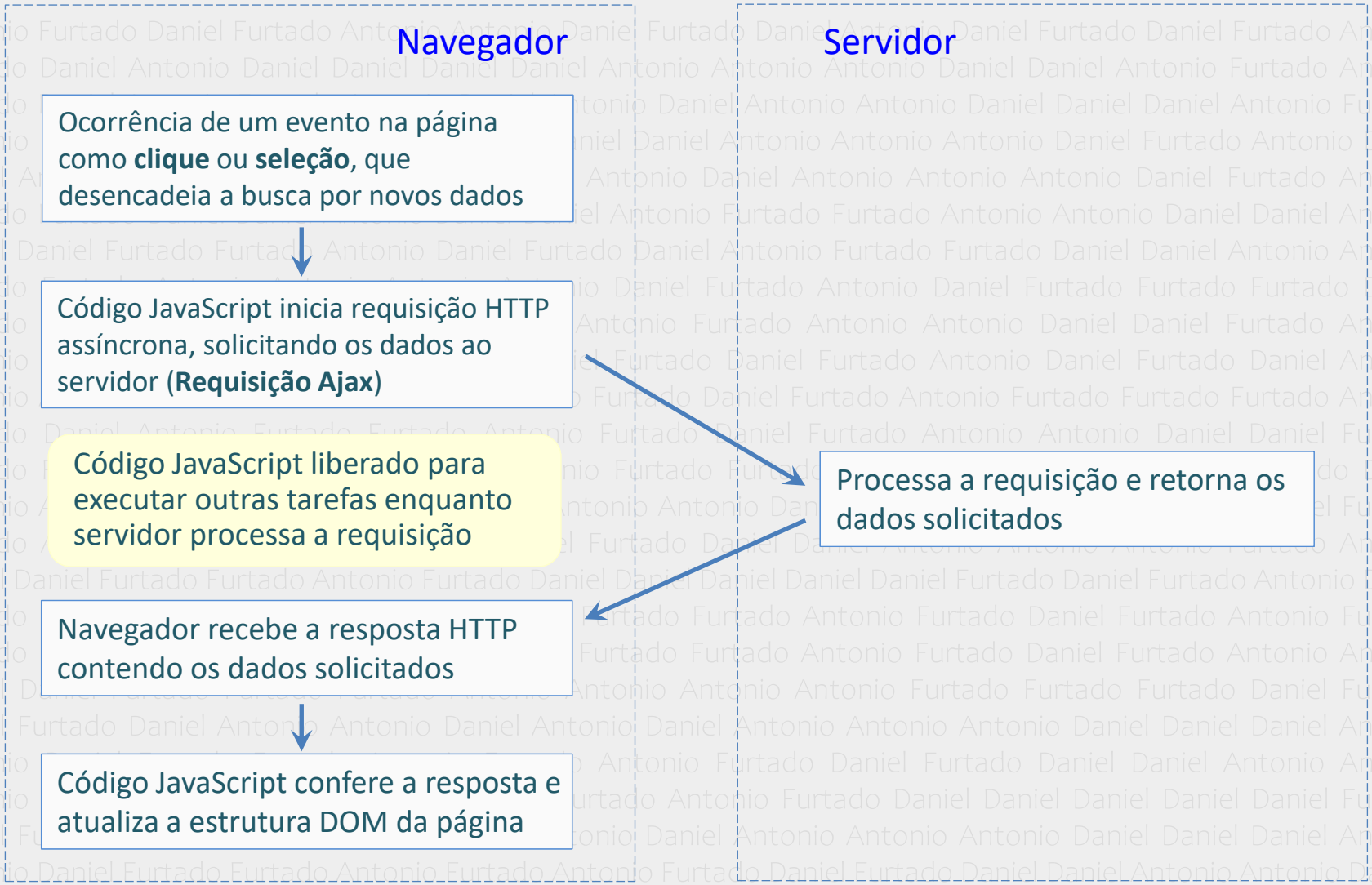
Minas Gerais

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



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.

Ajax com o XMLHttpRequest

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).

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.

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**).

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

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>
```

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.

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

OBS: 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.

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 da 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

```
async function getJSON(URL) {
   const response = await fetch(URL);
   return 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)

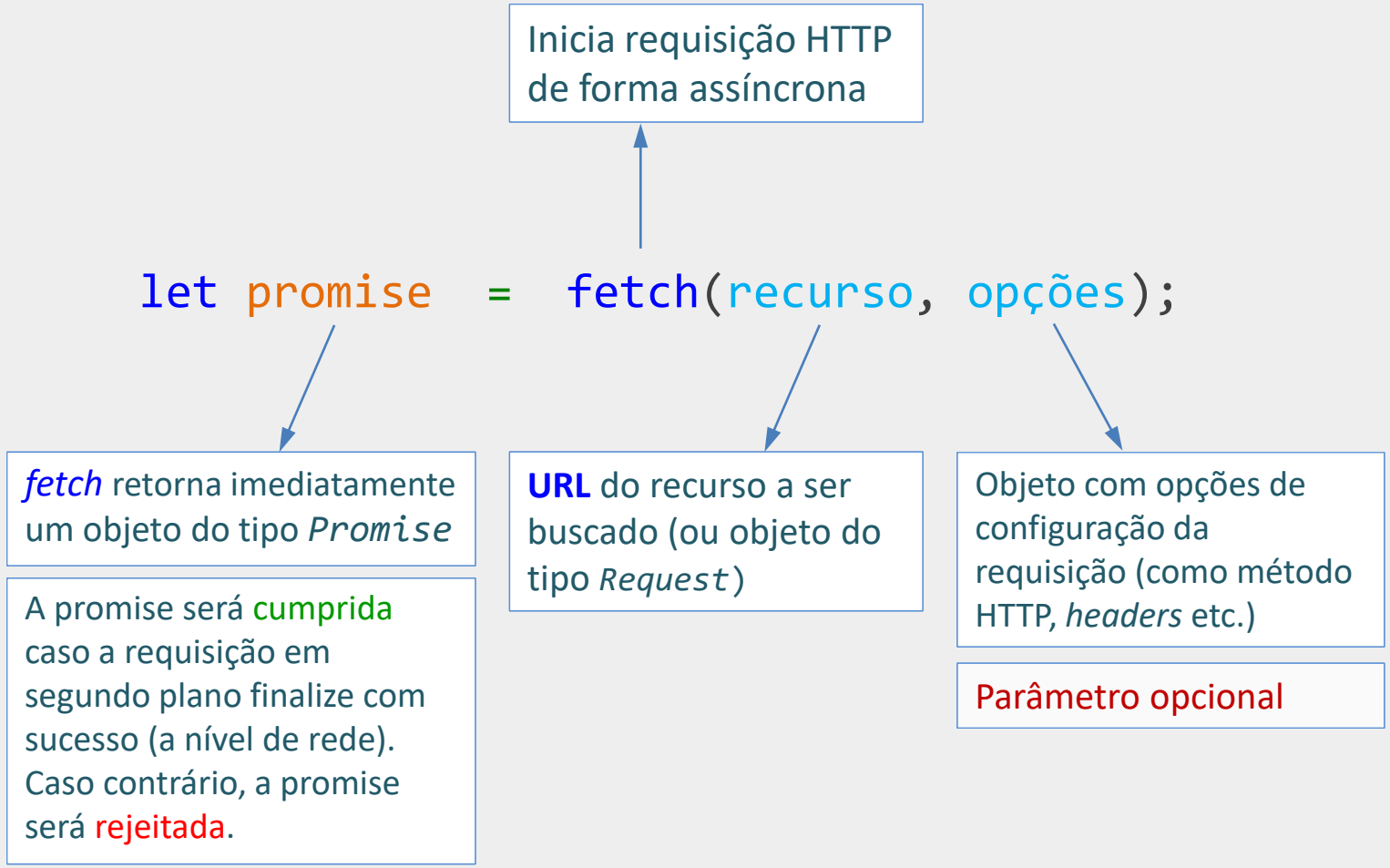
Introdução à Promises

- 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 devem ser indicadas funções de callback utilizando o método then() da promise

Introdução à Promises - Método fetch



Método then

- then é um método do objeto promise
- then permite resgatar o resultado ou o erro produzido pela tarefa:
  - Duas funções de callback podem ser passadas na chamada do método then;
  - A primeira delas é a callback de sucesso, que será chamada para processar o resultado da tarefa caso ela finalize com sucesso;
  - A segunda é a callback de erro, que será chamada para tratar o erro, caso a tarefa finalize com falha;

Exemplo – Passando callbacks para o método then

```
let promise = fetch(URL);

promise.then(
  function (result) {
    console.log(result);
  },
  function (error) {
    console.log(error);
  }
);
```

Indique uma função de *callback* a ser chamada quando a promise finalizar com sucesso ***(fulfils)***

Indique uma função de *callback* a ser chamada quando a promise finalizar com erro ***(rejects)***

Opcional

As funções de callback recebem por parâmetro o resultado obtido pela operação assíncrona (ou erro produzido em caso de falha).

Callbacks definidas como funções anônimas

Garantias: no momento da chamada do método **then** é possível que a promise já tenha sido finalizada (*fulfilled* ou *rejected*). Ainda assim, a função de callback indicada será chamada (de sucesso ou de falha, respectivamente).

async/await

- Parte da ECMAScript 2017 (não suportado por navegadores mais antigos como o Internet Explorer)
- Possibilita que funções assíncronas (que retornam promises) 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

Vantagens de Utilizar async/await

- Maior clareza e simplicidade do código
- Dispensa os aninhamentos das chamadas .then/.catch
- Melhor tratamento de erros com try/catch
- Mais fácil de depurar

Função Assíncrona e Expressões async/wait

```
async function funcaoExemplo() {
  try {
    const response = await fetch("endereco.php?cep=38400");
    const endereco = await response.json();
    console.log(endereco);
  } catch(e) { console.error(e) }
}
```

- await pode ser usada na chamada de funções que retornam promises
- Permite que funções assíncronas sejam chamadas no “estilo” das síncronas
- Suspende a execução da **função assíncrona** (funcaoExemplo) até que a promise retornada seja cumprida ou rejeitada (sem bloquear a thread principal)
- O valor resolvido da promise será o valor de retorno da expressão await

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);
  }
}
```

Neste exemplo, o bloco catch mostrará um erro nas seguintes situações:

1) Se a requisição não finalizar devido a um erro de rede (a promise retornada pelo fetch é rejeitada)

2) Se a requisição retornar um código de status fora da faixa 200-299

3) Se houver um erro na leitura ou conversão dos dados em JSON

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 bodyText = await response.text();
    const endereco = JSON.parse(bodyText);
    console.log(endereco);
  }
  catch (e) {
    console.log(bodyText ?? "");
    console.error(e);
  }
}
```

Este exemplo lê o corpo da resposta HTTP de forma textual e converte a string JSON em objeto JavaScript utilizando a função JSON.parse. Caso o servidor não retorne uma string JSON válida (mas uma mensagem de erro de script, por exemplo), o conteúdo textual retornado (mensagem de erro) poderá ser visto no console do navegador. No exemplo do slide anterior, o desenvolvedor precisaria utilizar o ambiente de desenvolvimento do navegador para visualizar o conteúdo da resposta HTTP.

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/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**.