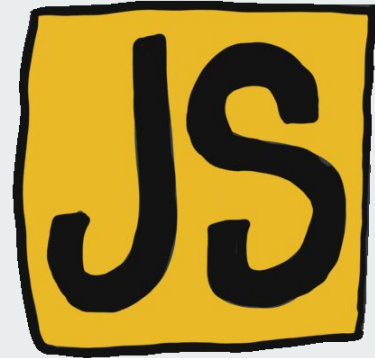




PROGRAMAÇÃO ASSÍNCRONA E APIs REST





Conteúdo

- Funções Callback
- Promises
- Async/Await
- APIs REST
- Chamadas a APIs em JavaScript



O que é programação assíncrona?

- Até agora programamos ações que aconteciam **uma após a outra** em uma **sequência definida no código**
- A **programação assíncrona** permite executar ações ao tempo todo sem uma sequência preestabelecida
- **Um exemplo:** usuário está no checkout, manda salvar seu endereço na conta, mas pode prosseguir para a finalização sem recarregar a página, pois adicionar endereço ocorre de forma assíncrona



Funções Callbacks

- Um dos recursos JavaScript mais utilizados em programação assíncrona são as **Callbacks Functions**
- Com elas podemos executar ações após algum acontecimento no código
- Uma **Callback** é uma função passada como argumento para outra função
- Essa técnica permite que **uma função chame outra função**



Controle de Sequência

Chamando duas funções

```
function myDisplayer(some) {  
  document.getElementById("demo").innerHTML = some;  
}  
  
function myCalculator(num1, num2) {  
  let sum = num1 + num2;  
  return sum;  
}  
  
let result = myCalculator(5, 5);  
myDisplayer(result);
```

Chamando explicitamente uma função a partir de outra

```
function myDisplayer(some) {  
  document.getElementById("demo").innerHTML = some;  
}  
  
function myCalculator(num1, num2) {  
  let sum = num1 + num2;  
  myDisplayer(sum);  
}  
  
myCalculator(5, 5);
```



Usando Callback Function

```
function myDisplayer(some) {  
    document.getElementById("demo").innerHTML = some;  
}
```

```
function myCalculator(num1, num2, myCallback) {  
    let sum = num1 + num2;  
    myCallback(sum);  
}
```

```
myCalculator(5, 5, myDisplayer);
```



Funções Callbacks

- No mundo real, as callbacks são mais frequentemente usadas com **funções assíncronas**
- Um exemplo típico é o JavaScript **setTimeout()**



```
1 console.log("Ainda não chamou a callback");
2
3 setTimeout(function(){
4     console.log("Chamou a callback");
5 }, 3000);
6
7 console.log("Ainda não chamou a callback");
```



Funções Callbacks

- Ao usar a função JavaScript `setTimeout()`, você pode especificar uma função **callback** a ser executada no tempo limite:



```
1 console.log("Ainda não chamou a callback");
2
3 setTimeout(function(){
4     console.log("Chamou a callback");
5 }, 3000);
6
7 console.log("Ainda não chamou a callback");
```



Funções Callbacks

- Ao usar a função JavaScript `setInterval()`, você pode especificar uma função **callback** a ser executada a cada intervalo de tempo definido:



```
1  setInterval(imprimirData, 1000);
2
3  function imprimirData() {
4      let d = new Date();
5      console.log(d.getHours() + ":" +
6                  d.getMinutes() + ":" +
7                  d.getSeconds());
8  }
```



Alternativas às Funções Callbacks

- Com a programação assíncrona, os programas JavaScript podem iniciar tarefas de longa duração e continuar executando outras tarefas em paralelo
- Programas assíncronos **são difíceis de escrever e de depurar**
- Por esse motivo, a maioria dos métodos assíncronos modernos do JavaScript **não usa callbacks**
- Em vez disso, no JavaScript, a programação assíncrona é resolvida com o uso de **Promises** (promessas)



Promises

- **Promises** são ações assíncronas que podem produzir um valor em algum momento
 - *“Eu prometo um resultado”*



Promises: sintaxe

- O objeto das promises é **Promise**
- **resolve** é o método que resolve uma Promise
- Além do resolve, há o método **reject**, que ocorre quando a promessa não satisfaz nosso programa
- **then** é o que faz ela poder ser executada em um ponto futuro

```
let myPromise = new Promise(function(myResolve, myReject) {  
  // "Producing Code" (May take some time)  
  
  myResolve(); // when successful  
  myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise)  
myPromise.then(  
  function(value) { /* code if successful */ },  
  function(error) { /* code if some error */ }  
);
```



Falha nas Promises

- Uma Promise pode falhar, podemos reter esse erro com um método chamado **catch**

```
let myPromise = new Promise(function(myResolve, myReject) {  
  // "Producing Code" (May take some time)  
  
  myResolve(); // when successful  
  myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise)  
myPromise.then(  
  function(value) { /* code if successful */ },  
  function(error) { /* code if some error */ }  
)  
.catch(  
  function(failure) { /* code if some failure */ }  
);
```



Async / Await

- Podemos criar **funções assíncronas** com a palavra reservada **async**
- Elas retornam uma **Promise**
- Nas **async functions** podemos determinar um ponto de espera para uma promise ser resolvida com a palavra reservada **await**, a fim de apresentar os resultados

```
function somaComDelay(a, b) {  
  return new Promise(resolve => {  
    setTimeout(() => {  
      resolve(a + b);  
    }, 2000);  
  })  
}
```

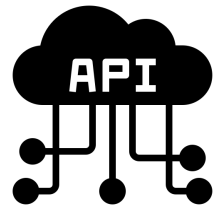
```
async function soma(a, b, c, d) {  
  let x = somaComDelay(a, b);  
  let y = somaComDelay(c, d);  
  
  return await x + await y;  
}
```

```
soma(8, 5, 3, 2).then(resultado => console.log(resultado));
```



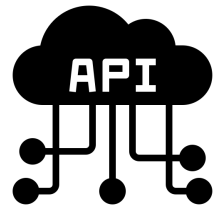
Exercício

- Crie uma função assíncrona que use um callback para simular uma chamada de API que demora 2 segundos para retornar uma mensagem "Dados carregados".



APIs REST

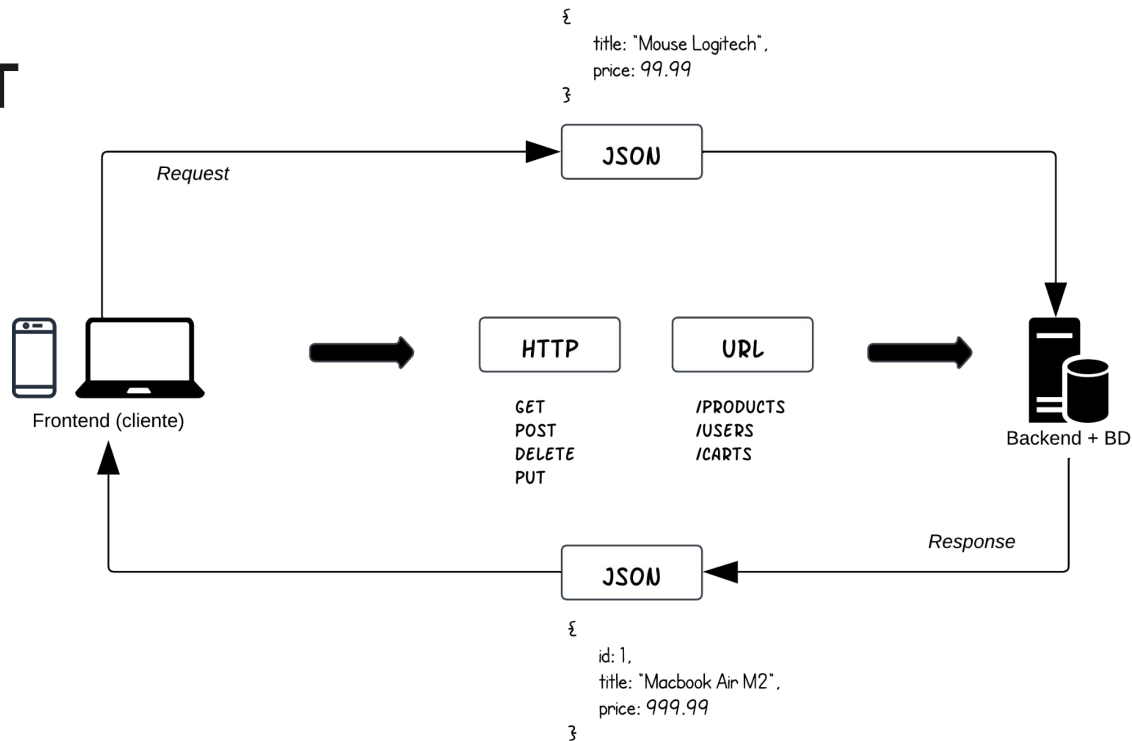
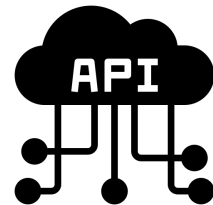
- O que é uma API?
 - APIs (Application Programming Interfaces) permitem que **diferentes sistemas se comuniquem**.
 - No desenvolvimento web, as APIs são usadas para **acessar dados externos** ou **executar operações em um servidor**.
 - Exemplos:
 - Obter produtos do servidor de uma loja virtual para apresentá-los no frontend da aplicação;
 - Processar uma operação de pagamento através de um serviço de pagamento externo;
 - Calcular o valor do frete e o prazo de entrega de um pedido a partir do endereço do cliente, através de uma solicitação à API de alguma empresa de entregas.

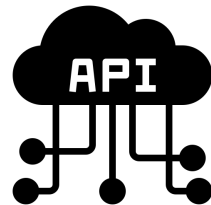


APIs REST

- O que é uma REST?
 - *Representational State Transfer* (REST) é um **estilo de arquitetura** para projetar APIs de forma simples, escalável e eficiente, no ambiente Web.
 - O objetivo principal é facilitar a comunicação entre cliente e servidor através de **regras e padrões baseados no protocolo HTTP**.
 - **Por que usar?**
 - *Simples de implementar e usar.*
 - *Compatível com qualquer linguagem que suporte HTTP.*
 - *Ideal para integração entre sistemas e desenvolvimento de aplicativos web e móveis.*

APIs REST





APIs REST

- Características Principais

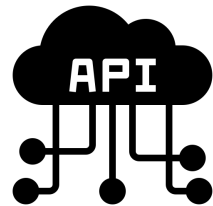
- *Stateless* (Sem estado):

- Cada requisição ao servidor deve conter todas as informações necessárias.
 - O servidor não armazena informações de estado entre as requisições.

- Baseada em Recursos:

- Cada recurso (ex.: produtos, usuários) é representado por uma URL única.
 - Exemplo:

- `/products`
 - `/products/1`



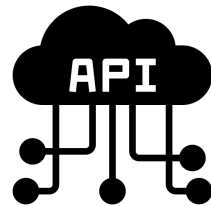
APIs REST

- URL

- *URL - Universal Resource Locator*
- A URL é usada para localizar/acessar os recursos pelo nome:
 - `GET: host:port/produtos`
 - `GET: host:port/produtos?page=4`
 - `GET: host:port/produtos/25`
 - `GET: host:port/produtos/25/categorias`

- Padrão para URLs

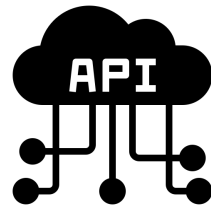
- A URL deve indicar o recurso desejado e não a ação a ser executada.
- A ação deve ser expressa pelo método HTTP.
- **ERRADO:**
 - `GET: host:port/inserirtProduto`
 - `GET: host:port/listartProduto`
- **CERTO:**
 - `POST: host:port/produtos`
 - `GET: host:port/products`



APIs REST

- **Características Principais**

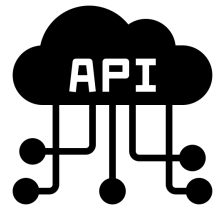
- Uso de métodos HTTP na requisição (leia mais [aqui](#))
 - **GET, POST, PUT, DELETE** são os mais comuns e definem a ação a ser executada.
 - Ação do método deve ser previsível e consistente.
- Uso de códigos de status HTTP na resposta
 - Códigos numéricos retornados pelo servidor em resposta a uma requisição.
 - Eles indicam o resultado da operação e ajudam o cliente (navegador ou aplicação) a entender se a solicitação foi bem-sucedida, teve algum problema ou precisa de ajustes.



APIs REST

Tabela de resumo de Status Codes mais comuns (leia mais [aqui](#))

CÓDIGO	CATEGORIA	DESCRIÇÃO
200	Sucesso	Requisição bem-sucedida.
201	Sucesso	Recurso criado com sucesso.
204	Sucesso	Requisição bem-sucedida sem conteúdo.
400	Erro do Cliente	Requisição inválida.
401	Erro do Cliente	Não autenticado, autenticação requerida.
403	Erro do Cliente	Proibido, acesso negado.
404	Erro do Cliente	Recurso não encontrado.
500	Erro do Servidor	Erro genérico no servidor.
503	Erro do Servidor	Serviço temporariamente indisponível.



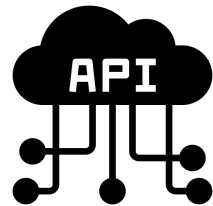
APIs REST

- Dados e Formatos nas APIs REST

- Formato de dados mais usado é o **JSON**:

- Estrutura leve e fácil de interpretar por humanos e máquinas.
 - **JavaScript Object Notation (JSON)**: formato de dados baseado na notação de objetos em JavaScript
 - Exemplo:

```
{  
  "id": 1,  
  "title": "Produto Exemplo",  
  "price": 29.99  
}
```

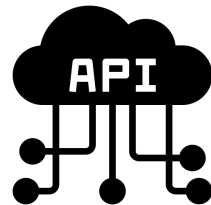


APIs REST

- Sintaxe Básica - JSON

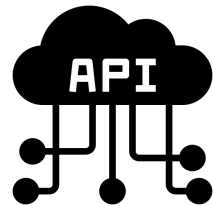
- Dados formatados como pares “chave-valor”
- O nome das chaves fica entre aspas, separado do valor por dois pontos
- Objetos são definidos entre chaves {} e arrays entre colchetes []

```
{  
  "id": 1,  
  "title": "Mouse",  
  "price": 29.99,  
  "categories": ["Informática", "Acessórios"]  
}
```



APIs REST

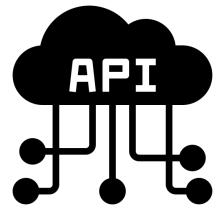
- **Dados e Formatos nas APIs REST**
 - Outros formatos suportados, por exemplo, são: XML, YAML, CSV, etc.



Chamadas a APIs

- Como fazemos chamadas a APIs no JavaScript?
 - Utilizamos métodos como `fetch` ou bibliotecas como **Axios** para enviar requisições e receber respostas.
 - O `fetch` é uma função nativa do JavaScript para fazer requisições HTTP. Ele retorna uma Promise que resolve em um objeto Response contendo os dados da resposta.

```
1 fetch('https://fakestoreapi.com/products')  
2   .then(response => response.json())  
3   .then(data => console.log(data))  
4   .catch(error => console.error('Erro:', error));
```



Chamadas a APIs

- Exemplo prático - Listando Produtos

- [Fake Store API](#)

```
1  async function displayProducts() {
2    try {
3      const response = await fetch('https://fakestoreapi.com/products');
4      const products = await response.json();
5
6      const container = document.getElementById('product-list');
7      products.forEach(product => {
8        const item = document.createElement('div');
9        item.innerHTML = `

### ${product.title}</h3><p>${product.price}</p>`; 10 container.appendChild(item); 11 }); 12 } catch (error) { 13 console.error('Erro:', error); 14 } 15 } 16 17 displayProducts();


```