

PROGRAMAÇÃO FRONT END

João Choma Neto

joao.choma@unicesumar.edu.br

<https://github.com/JoaoChoma/frontend>

Unicesumar – Maringá



MODULARIZAÇÃO

MODULARIZAÇÃO

Na aula de hoje, discutiremos o conceito de modularização no desenvolvimento frontend e como aplicá-lo em projetos web. A modularização é uma prática essencial para manter o código organizado, reutilizável e fácil de manter.

O QUE É MODULARIZAÇÃO

- **Modularização** é o processo de dividir um sistema em partes independentes e autônomas, chamadas de módulos.
- Cada módulo tem uma responsabilidade específica e pode ser desenvolvido, testado e mantido separadamente dos outros módulos.

O QUE É MODULARIZAÇÃO

- Em desenvolvimento frontend, a modularização envolve dividir o código JavaScript, HTML e CSS em módulos distintos, cada um responsável por uma funcionalidade ou conjunto de funcionalidades específicas.

POR QUE MODULARIZAR?

- **Reutilização de código:** A modularização permite reutilizar módulos em diferentes partes do aplicativo, economizando tempo e esforço de desenvolvimento.

POR QUE MODULARIZAR?

- **Facilidade de manutenção:** Módulos independentes são mais fáceis de entender, testar e modificar, facilitando a manutenção do código ao longo do tempo.

POR QUE MODULARIZAR?

- **Escalabilidade:** Projetos modulares são mais fáceis de escalar à medida que o aplicativo cresce, pois novos recursos podem ser adicionados como novos módulos sem afetar o código existente.

COMO MODULARIZAR NO FRONTEND?

- **Divisão em Componentes:** No frontend, os componentes são a unidade básica de modularização. Cada componente é responsável por uma parte específica da interface do usuário e encapsula seu próprio comportamento e estilo.

COMO MODULARIZAR NO FRONTEND?

- **Arquivos Separados:** Separe cada componente em arquivos JavaScript, HTML e CSS separados. Isso ajuda a manter o código organizado e facilita a localização e edição de código relacionado.

COMO MODULARIZAR NO FRONTEND?

- **Módulos JavaScript:** Use módulos JavaScript para organizar e encapsular funcionalidades relacionadas. Isso pode ser feito usando a sintaxe import e export do ECMAScript.

OBJETOS

O QUE SÃO OBJETOS?

- **Objetos** em JavaScript são estruturas de dados que permitem armazenar e organizar informações relacionadas em pares de chave-valor.
- Cada valor é acessado por meio de uma chave exclusiva, tornando os objetos uma estrutura de dados eficiente para representar e manipular dados complexos.

ESTRUTURA DE UM OBJETO

- Um **objeto** em JavaScript é definido usando a sintaxe de chaves {} e pode conter zero ou mais pares de chave-valor.
- Cada par consiste em uma chave única seguida por dois pontos : e o valor correspondente.
- Os pares de chave-valor são separados por vírgulas.

OBJETO JS

```
const pessoa = {  
  nome: 'João',  
  idade: 30,  
  cidade: 'São Paulo'  
};
```

ACESSANDO PROPRIEDADES DE UM OBJETO

- As propriedades de um objeto podem ser acessadas usando a notação de ponto . ou a notação de colchetes [].

```
console.log(pessoa.nome); // Saída: João
```

```
console.log(pessoa['idade']); // Saída: 30
```


ACESSANDO PROPRIEDADES DE UM OBJETO

- Novas propriedades podem ser adicionadas a um objeto atribuindo um valor a uma chave que ainda não existe.

```
pessoa.profissao = 'Desenvolvedor';
```

```
pessoa.idade = 31;
```

ACESSANDO PROPRIEDADES DE UM OBJETO

- Propriedades de um objeto podem ser removidas usando o operador delete.

`delete pessoa.cidade;`

ACESSANDO PROPRIEDADES DE UM OBJETO

- Além de armazenar dados, os objetos em JavaScript também podem conter métodos, que são funções definidas como propriedades.

```
const pessoa = {  
  nome: 'Maria',  
  idade: 25,  
  saudacao: function() {  
    return 'Olá, meu nome é ' + this.nome + '!';  
  }  
};
```

```
console.log(pessoa.saudacao()); // Saída: Olá, meu nome é Maria!
```

```
let item = "Maçã";
```

```
let valor = 2.50;
```

```
let objeto = { item, valor };
```

```
console.log(objeto); // Output: { item: "Maçã", valor: 2.50 }
```

this

- O `this` é uma palavra-chave especial em JavaScript que é usada para se referir ao objeto atual em que um determinado código está sendo executado.
- O valor de `this` é determinado pelo contexto de execução no qual o código está sendo executado e pode variar dependendo de como e onde uma função é chamada.

this

- Quando uma função é chamada como um método de um objeto, o valor de this é o próprio objeto no qual o método é chamado.
- O this é usado para acessar outras propriedades e métodos do objeto dentro do método.
- Quando uma função de evento é acionada em resposta a uma interação do usuário, como um clique de botão, o valor de this geralmente se refere ao elemento DOM que disparou o evento.

CRUD

CRUD é um acrônimo para as quatro operações básicas de um sistema de gerenciamento de banco de dados ou aplicação: Create (Criação), Read (Leitura), Update (Atualização) e Delete (Exclusão)

Essas operações representam as principais ações que podem ser realizadas em qualquer tipo de sistema que envolva manipulação de dados.

CRUD

- 1.Create (Criação):** A operação de criação envolve a adição de novos dados ao sistema.
- 2.Read (Leitura):** A operação de leitura envolve a recuperação de dados existentes do sistema.
- 3.Update (Atualização):** A operação de atualização envolve a modificação dos dados existentes no sistema.
- 4.Delete (Exclusão):** A operação de exclusão envolve a remoção de dados do sistema.

CALLBACK

Um callback é uma função que é passada como argumento para outra função e é executada dentro dessa função. Em outras palavras, você está dizendo à função principal: "Quando terminar de fazer o que precisa fazer, execute esta função adicional."

Callbacks são uma parte fundamental do JavaScript, especialmente em operações assíncronas, como requisições de rede, leitura de arquivos e interações de usuário.

```
function saudacao(nome, callback) {  
    console.log("Olá, " + nome + "!");  
  
    // Chamando o callback depois de um segundo  
    setTimeout(function() {  
        callback();  
    }, 1000);  
}  
  
function finalizacao() {  
    console.log("Espero que você tenha um ótimo dia!");  
}  
  
// Chamando a função saudacao com um callback  
saudacao("João", finalizacao);
```

BOOTSTRAP

<https://getbootstrap.com/>



Include via CDN

When you only need to include Bootstrap's compiled CSS or JS, you can use [jsDelivr](#). See it in action with our simple [quick start](#), or [browse the examples](#) to jumpstart your next project. You can also choose to include Popper and our JS [separately](#).

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist" 
```

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dis" 
```



JSON

JSON

- JSON significa JavaScript Object Notation.
- É um formato leve e fácil de ler para troca de dados.
- Baseado na sintaxe de objetos JavaScript.
- Independente de linguagem.

JSON

- Consiste em pares chave-valor.
- Os valores podem ser strings, números, objetos, arrays, booleanos ou nulos.
- Exemplo:

```
{ "nome": "Maria", "idade": 25, "cidade": "Rio de Janeiro", "ativos":  
["caminhada", "leitura", "cozinhar"], "casada": false, "endereco": null }
```

- Os objetos são delimitados por chaves {} e os pares chave-valor são separados por vírgulas.

Por que JSON é Importante?

- Ampla adoção na web devido à sua simplicidade.
- Facilita a troca de dados entre servidores e clientes.
- Usado em APIs, armazenamento de dados, etc.

Como Usar JSON em JavaScript

- `JSON.parse()`: Converte uma string JSON em um objeto JavaScript.
- `JSON.stringify()`: Converte um objeto JavaScript em uma string JSON.

PROGRAMAÇÃO FRONT END

João Choma Neto

joao.choma@unicesumar.edu.br

<https://github.com/JoaoChoma/frontend>

Unicesumar – Maringá

