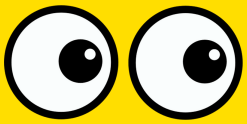
SENHORA DEVA

Componente web com Vanillão JavaScript

@senhora.deva

Componente web com Vanilla JavaScript

Você sabia que podemos criar componentes sem precisar de um framework? Ora, com JavaScript "Vanillão" podemos fazer isso de uma maneira bem simples!



Antes precisamos entender o que são os componentes da Web.

De maneira resumida, **os componentes Web** são conjuntos que especificam e adicionam funcionalidades em nossas aplicações é o que observa o **Mdn web docs**. Por exemplo, eles permitem a reutilização e o encapsulamento de elementos HTML.

Componente web com Vanilla JavaScript

Os componentes Web tem como base três tecnologias para criar elementos personalizados. Eles abrangem:

- **Elementos personalizados**: essas APIs permitem definir e criar elementos personalizados para fornecer a interface de usuário.
- **Shadow DOM**: APIs que permitem o encapsulamento. Elementos específicos são separados do seu DOM principal, evitando problemas como colisão de documentos.
- **Modelos HTML**: elementos que permitem declarar fragmentos de modelos de marcação que não são exibidos na página. Esses elementos são utilizados como um modelo para serem reutilizados em vários lugares.

Componente web com Vanilla JavaScript

Para colocar em prática o conceito, vamos criar um componente chamado "app-title", bora lá !

Componente web com Vanilla JavaScript

Vamos começar criando os arquivos index.html e Title.js.
No primeiro arquivo crie a "casca" do HTML.
Ah, e insira o caminho do nosso segundo arquivo.

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Componente Title JavaScript Vanilla</title>
</head>
<body>
<script src="./Title.js"></script>
</body>
</html>
```

Componente web com Vanilla JavaScript

No Title.js vamos criar o modelo, nós iremos ter um elemento com conteúdo HTML **<template>**

```
const template = document.createElement('template');
```

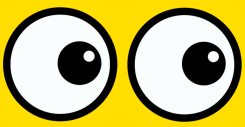
Agora vamos carregar nosso modelo com o conteúdo. Ele será armazenado em uma **template** variável e vinculado em propriedades **innerHTML**.
A innerHTML define o conteúdo HTML no elemento. Logo, podemos adicionar o HTML que queremos exibir na tela, como , <p>, e mais elementos HTML.

Componente web com Vanilla JavaScript

```
template.innerHTML = `
<style>
  .container {
    font-family: sans-serif;
    color: #800080;
    width: 250px;
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 10vh;
  }

</style>
<section class="container">
  <h1></h1>
</section>
`;
```

Em seguida o **elemento HTML** - iremos criar a classe Title estende a classe HTMLElement.



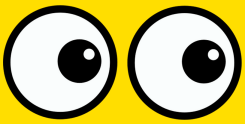
O HTMLElement representa todos os elementos HTML.

```
class Title extends HTMLElement{}
```


SENHORA DEVA

Componente web com Vanilla JavaScript

Encapsular o elemento HTML - temos que criar o construtor e chamar o **super( )**



super( ) - método da classe base para herdar os recursos de uma classe.

Implementamos um shadow DOM **this.attachShadow({mode: 'open'})** esse carinha é que vai encapsular nosso componente web.

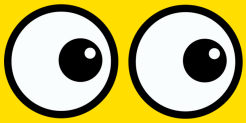
```
class Title extends HTMLElement{
 constructor(){
     super();
     this.attachShadow({ mode: 'open'});
     this.shadowRoot.appendChild(template.content.cloneNode(true));
     this.shadowRoot.querySelector('h1').innerText = this.getAttribute('name');

 }
```

Componente web com Vanilla JavaScript

De acordo com a documentação, os Web Components nos permite utilizar o shadow DOM, um recurso embutido no navegador. Assim, se os elementos filho forem adicionados a um shadow DOM de um componente, eles serão agrupados dentro de uma raiz sombra.

Componente web com Vanilla JavaScript



A API do componente obtém o name dos atributos a serem exibidos na página.

```
this.shadowRoot.querySelector('h1').innerText = this.getAttribute('name');
```

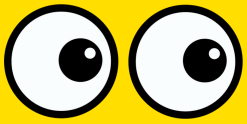
Métodos de ciclo de vida - é importante que você saiba que, existem quatro métodos de retorno de chamada definidos dentro dos elementos personalizados. Eles permitem que o código seja executado quando ocorrerem eventos em um elemento.
Bora ver o que estamos utilizando aqui?

- **connectedCallback**: é chamado quando o componente é inserido no DOM de um documento HTML.

Componente web com Vanilla JavaScript

```
 this.shadowRoot.que connectedCallback(){
  this.h1 = this.getAttribute("name")
  this.render();
}

render(){
  this.h1;
}
```



Temos também disconnectedCallback, adoptedCallback e attributeChangedCallbac. Sugerimos que pesquisem sobre, pois aprender a fazer isso é uma skill importante.

Criando elementos personalizados - o componente foi criado, mas falta os elementos personalizados.

```
window.customElements.define('app-title', Title);
```

@senhora.deva

SENHORA DEVA

Componente web com Vanilla JavaScript

Ufa! Conseguimos, agora falta testar nosso componente, para isso, basta chama-lo lá no index.html.

```
<main>
      <app-title name="Senhora Deva, uma senhora de programa"> </app-title>
</main>
```

Componente web com Vanilla JavaScript

```
const template = document.createElement('template');
template.innerHTML = `
<style>
  .container {
    font-family: sans-serif;
    color: #800080;
    width: 250px;
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 10vh;
  }

</style>
<section class="container">
  <h1></h1>
</section>
`;

class Title extends HTMLElement{
 constructor(){
     super();
     this.attachShadow({ mode: 'open'});

this.shadowRoot.appendChild(template.content.cloneNode(true));
     this.shadowRoot.querySelector('h1').innerText =
this.getAttribute('name');

 }

 connectedCallback(){
   this.h1 = this.getAttribute("name")
   this.render();
 }

 render(){
   this.h1;
 }
}

window.customElements.define('app-title', Title);
```

Materiais de pesquisa

https://developer.mozilla.org/ptBR/docs/Web/Web_Components

https://www.zup.com.br/blog/web-components

https://javascript.info/web-components

Componente web com Vanilla JavaScript

Código completo no

@senhora.deva

SENHORA DEVA