📌 Como Funciona o `ElementRef` em Componentes Angular

O `ElementRef` no Angular é usado para acessar e manipular diretamente elementos do DOM dentro de um componente. Isso permite modificar atributos, estilos e até adicionar eventos sem precisar de diretivas específicas.

📌 O Que é `ElementRef`?

O `ElementRef` é uma classe do Angular fornecida pelo módulo `@angular/core`. Ele encapsula um **elemento nativo do DOM** e permite manipulá-lo diretamente.

⚠️ **Atenção!** Embora o `ElementRef` seja poderoso, seu uso direto **não é recomendado** em muitos casos, pois pode comprometer a segurança do Angular (ataques XSS). O uso de **ViewChild** e **Renderer2** é geralmente preferido.

📌 Como Usar `ElementRef` no Angular

📌 Exemplo Simples

Vamos criar um exemplo básico onde mudamos a cor de fundo de um `div` assim que o componente for carregado.

1️⃣ HTML (`app.component.html`)

```
<div id="minha-div">Olá, ElementRef!</div>
```

2️⃣ TypeScript (`app.component.ts`)

```
import { Component, ElementRef, OnInit } from '@angular/core';

@Component({
```

```
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  constructor(private readonly elRef: ElementRef) {}

  ngOnInit() {
    // Pegando o elemento <div> pelo ID e modificando seu estilo
    const divEl = this.elRef.nativeElement.querySelector('#minha-div');
    divEl.style.backgroundColor = 'blue';
    divEl.style.color = 'white';
    divEl.style.padding = '10px';
    divEl.style.borderRadius = '5px';
  }
}
```

🔹 Aqui, `this.elRef.nativeElement.querySelector('#minha-div')` busca o elemento pelo ID e altera suas propriedades **diretamente**.

📌 Usando `ElementRef` com `@ViewChild`

Se tivermos uma **referência local** no HTML (`#minhaDiv`), podemos capturar o elemento usando `@ViewChild`.

📌 Exemplo com `ViewChild`

1️⃣ HTML (`app.component.html`)

```
<div #minhaDiv>Sou um elemento!</div>
```

2️⃣ TypeScript (`app.component.ts`)

```
import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';

@Component({
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  @ViewChild('minhaDiv') divEl!: ElementRef;

  ngOnInit() {
    setTimeout(() => { // Necessário para esperar a view ser carregada
      this.divEl.nativeElement.style.backgroundColor = 'red';
      this.divEl.nativeElement.style.color = 'white';
      this.divEl.nativeElement.textContent = 'Texto alterado!';
    });
  }
}
```

🔹 O `@ViewChild('minhaDiv')` captura a `<div>` e permite manipulá-la diretamente.
🔹 O `setTimeout()` é necessário porque `@ViewChild` só é definido **após a renderização do HTML**.

📌 Adicionando e Removendo Classes CSS

Podemos usar o `classList` para adicionar/remover classes dinamicamente.

📌 Exemplo

1️⃣ HTML

```
<div #meuElemento class="estilo-base">Elemento estilizado!</div>
```

2️⃣ CSS

```
.estilo-base {
  padding: 10px;
  font-size: 16px;
}

.estilo-dinamico {
  background-color: green;
```

```
  color: white;
  font-weight: bold;
  padding: 20px;
}
```

3️⃣ **TypeScript**

```
import { Component, ElementRef, ViewChild, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  @ViewChild('meuElemento') meuElemento!: ElementRef;

  ngOnInit() {
    setTimeout(() => {
      this.meuElemento.nativeElement.classList.add('estilo-dinamico'); //
Adiciona classe CSS
    }, 2000);

    setTimeout(() => {
      this.meuElemento.nativeElement.classList.remove('estilo-dinamico'); //
Remove classe CSS
    }, 5000);
  }
}
```

- 🔹 Após **2 segundos**, a classe `'estilo-dinamico'` é adicionada.
- 🔹 Após **5 segundos**, a classe `'estilo-dinamico'` é removida.

📌 Manipulando Atributos do Elemento

Podemos modificar **atributos HTML** como `disabled`, `src`, `alt`, `title`, etc.

📌 Exemplo com `disabled`

1️⃣ HTML

```
<button #meuBotao>Botão Desativado</button>
```

2️⃣ TypeScript

```
import { Component, ElementRef, ViewChild, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  @ViewChild('meuBotao') botao!: ElementRef;

  ngOnInit() {
    setTimeout(() => {
      this.botao.nativeElement.setAttribute('disabled', 'true');
      this.botao.nativeElement.textContent = 'Agora estou desativado!';
    }, 3000);
  }
}
```

- 🔹 O botão é **desativado** após 3 segundos, alterando seu `disabled`.

📌 Adicionando Eventos Dinamicamente

Podemos adicionar eventos diretamente ao elemento DOM.

📌 Exemplo com `click`

1️⃣ HTML

```
<button #botaoClick>Me Clique</button>
```

2️⃣ TypeScript

```
import { Component, ElementRef, ViewChild, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  @ViewChild('botaoClick') botao!: ElementRef;

  ngOnInit() {
    this.botao.nativeElement.addEventListener('click', () => {
      alert('Botão foi clicado!');
    });
  }
}
```

- 🔹 Ao clicar no botão, aparece um **alerta**.

📌 Criando Novos Elementos Dinamicamente

Podemos criar elementos HTML e adicioná-los à página.

📌 Exemplo

1️⃣ HTML

```
<div #container></div>
```

2️⃣ TypeScript

```
import { Component, ElementRef, ViewChild, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
```

```
  @ViewChild('container') container!: ElementRef;

  ngOnInit() {
    setTimeout(() => {
      const novoParagrafo = document.createElement('p');
      novoParagrafo.textContent = 'Este parágrafo foi criado dinamicamente!';
      novoParagrafo.style.color = 'red';

      this.container.nativeElement.appendChild(novoParagrafo);
    }, 2000);
  }
}
```

- Após 2 segundos, um **novo `<p>` é criado dinamicamente** dentro do `div`.

📌 Conclusão

O `ElementRef` permite acessar elementos do DOM e manipulá-los diretamente. Aqui estão algumas operações que podemos fazer:

📌 Ação	📌 Código
Alterar estilo	`this.el.nativeElement.style.color = 'blue';`
Alterar conteúdo	`this.el.nativeElement.textContent = 'Novo texto';`
Adicionar classe	`this.el.nativeElement.classList.add('nova-classe');`
Remover classe	`this.el.nativeElement.classList.remove('classe-antiga');`
Adicionar atributo	`this.el.nativeElement.setAttribute('disabled', 'true');`
Adicionar evento	`this.el.nativeElement.addEventListener('click', callback);`
Criar elemento	`document.createElement('div');`

O uso do `ElementRef` deve ser feito com **cuidado**, pois manipular o DOM diretamente pode **quebrar a segurança** do Angular. O recomendado é usar **Renderer2** para essas manipulações sempre que possível. 🚀