Decorador @ViewChild en Angular

En esta lección, aprenderemos a utilizar el decorador `@ViewChild` en Angular. Veremos qué es, su sintaxis básica, cómo utilizarlo para acceder a elementos y componentes hijos en el DOM de Angular, y un ejemplo práctico que te permitirá comprender su utilidad en una aplicación Angular.

1. ¿Qué es el Decorador `@ViewChild`?

El decorador `@ViewChild` en Angular se utiliza para obtener una referencia directa a un elemento del DOM o a un componente hijo dentro del mismo componente. Esto es útil cuando necesitas interactuar con un elemento o componente desde el código TypeScript, como acceder a métodos, propiedades, o manipular el DOM directamente.

- **Acceso a Elementos del DOM:** Puedes usar `@ViewChild` para obtener una referencia a un elemento HTML como un `div`, `input`, o cualquier otro.

- **Acceso a Componentes Hijos:** También puedes utilizar `@ViewChild` para interactuar con un componente hijo, accediendo a sus métodos o propiedades desde el componente padre.

2. Sintaxis Básica de `@ViewChild`

La sintaxis básica de `@ViewChild` es la siguiente:

```
@ViewChild(selector: string | Type<any>, {static: boolean})
```

- **`selector`**: Puede ser una referencia a un elemento del DOM o un componente. El selector puede ser una clase, un tipo, o una referencia local (`#`).
- **`static`**: Indica si la referencia debe resolverse antes de que Angular procese el cambio de detección (generalmente `false`, a menos que necesites acceso antes de que el DOM esté completamente renderizado).

3. Ejemplo Práctico de `@ViewChild`

Vamos a crear un ejemplo donde accedemos a un `input` y a un componente hijo usando `@ViewChild`.

Paso 1: Acceso a un Elemento del DOM

En este ejemplo, queremos acceder a un campo de entrada (`input`) y cambiar su valor desde el mismo componente:

Creamos el componente ViewChild:

ng g c ViewChild --skip-tests

Código del Componente (`viewchild.component.html`)

```
<div class="container">
    <h3 class="text-primary">Acceso a Elemento del DOM</h3>
    <input #referenciaInput type="text" class="form-control w-50 mx-auto"
placeholder="Escribe algo">
    <button class="btn btn-success mt-2" (click)="cambiarTexto()">Cambiar Texto</button>
</div>
```

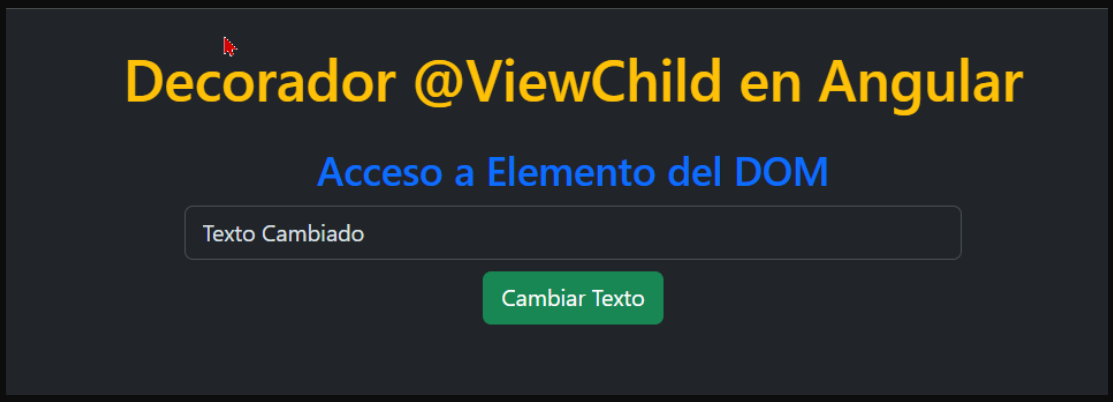
Código del Componente (`viewchild.component.ts`)

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

@Component({
  selector: 'app-viewchild',
  standalone: true,
  imports: [],
  templateUrl: './viewchild.component.html',
  styleUrl: './viewchild.component.css'
})
export class ViewchildComponent {
  @ViewChild('referenciaInput') inputElemento!: ElementRef;

  cambiarTexto() {
    this.inputElemento.nativeElement.value = 'Texto Cambiado';
  }

}
```

Explicación:

- **`@ViewChild('referenciaInput') inputElemento!: ElementRef;`:**
  - Usamos `@ViewChild` para obtener una referencia al elemento `input` en el DOM.

  - `ElementRef` es un tipo de Angular que proporciona una referencia directa al elemento en el DOM.
- **Método `cambiarTexto()`:**
  - Al hacer clic en el botón, cambiamos el valor del campo de entrada mediante la referencia al `input` que obtuvimos con `@ViewChild`.



Paso 2: Acceso a un Componente Hijo

Ahora, vamos a acceder a un componente hijo desde el componente padre utilizando `@ViewChild`. Desde el componente hijo únicamente definimos los elementos que necesitamos, pero no tienen ninguna sintaxis en particular. El uso de @ViewChild será desde el componente padre que veremos más adelante.

Código del Componente Hijo (`hijo.component.html`)

```
<div class="alert alert-info w-50 mx-auto">
    <p>{{ mensaje }}</p>
</div>
```

Código del Componente Hijo (`hijo.component.ts`)

```
import { Component, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'app-hijo',
  standalone: true,
  imports: [],
  templateUrl: './hijo.component.html',
  styleUrl: './hijo.component.css'
})
export class HijoComponent {
  mensaje: string = 'Mensaje desde el componente hijo';

  cambiarMensaje(nuevoMensaje: string) {
    this.mensaje = nuevoMensaje;
```

```
  }
}
```

Código del Componente Padre (`padre.component.html`)

```
<div class="container">
    <h3 class="text-primary">Componente Padre</h3>
    <app-hijo></app-hijo>
    <button class="btn btn-primary mt-3"
            (click)="cambiarMensajeHijo()">Cambiar Mensaje del Hijo</button>
</div>
```

Código del Componente Padre (`padre.component.html`)

```
import { Component, ViewChild } from '@angular/core';
import { HijoComponent } from "./hijo/hijo.component";

@Component({
  selector: 'app-padre',
  standalone: true,
  imports: [HijoComponent],
  templateUrl: './padre.component.html',
  styleUrl: './padre.component.css'
})
export class PadreComponent {
  @ViewChild(HijoComponent) componenteHijo!: HijoComponent;

  cambiarMensajeHijo() {
    this.componenteHijo.cambiarMensaje('Mensaje actualizado desde el Componente Padre');
  }
}
```

Explicación:

- **`@ViewChild(HijoComponent) componenteHijo!: HijoComponent;`:**
  - Usamos `@ViewChild` para obtener una referencia al componente hijo `HijoComponent`. Esto nos permite acceder a sus métodos y propiedades desde el componente padre.
- **Método `cambiarMensajeHijo()`:**
  - Al hacer clic en el botón, llamamos al método `cambiarMensaje` del componente hijo y cambiamos el mensaje que se muestra en el componente hijo.

4. Detalles Importantes sobre `@ViewChild`

- **`static: false` (por defecto)**: Cuando se establece en `false`, la referencia a un elemento o componente se resuelve después de que Angular ha renderizado la vista. Esto es útil cuando necesitas interactuar con el DOM después de que el ciclo de vida de la vista haya comenzado.
- **`static: true`**: Si necesitas que la referencia se resuelva antes de que la vista sea renderizada, establece `static: true`. Generalmente se usa cuando necesitas interactuar con el DOM o los componentes hijos durante la inicialización.

Conclusión

El decorador `@ViewChild` es una poderosa herramienta en Angular que te permite interactuar directamente con elementos del DOM y componentes hijos. Te proporciona control directo sobre el DOM y te permite comunicarte con componentes sin necesidad de emitir eventos.

Saludos!

Ing. Ubaldo Acosta

Fundador de GlobalMentoring.com.mx