PRIMEROS PASOS

Angular

ANGULAR

Componentes: creación y estructura

https://angular.dev/guide/components

ANGULAR

- Creación de componentes:
  - **ng generate component** *nombre_componente*
  - **ng generate component** *nombre_componente* ***--skip-tests*** *→No crea ficheros de test (spec.ts)*
  - Crea, dentro de app, una carpeta con el nombre del componente y el siguiente contenido:
    - **nombre_componente.ts**. Fichero TypeScript del componente.
    - **nombre_componente.html**. Código HTML del componente.
    - **nombre_componente.css**. Hoja de estilos del componente.
    - **nombre_componente. spec.ts**. Test unitarios del componente.

https://angular.dev/cli/generate#component

ANGULAR

- Creación de componentes:
  - Fichero ts del componente:
    - **selector** → nombre del componente para incrustar en la aplicación.
    - **imports** → recoge las dependencias o módulos que necesita el componente.
    - **template** o **templateUrls** → código HTML del componente.
    - **styles** o **styleUrl** → estilo CSS del componente.

ANGULAR

- Creación de componentes:
  - Fichero ts del componente:

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

@Component({
  selector: 'app-componente1',
  templateUrl:'./html/template1.html',
  styleUrls: ['./componente1.component.css']
})
```

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

@Component({
  selector: 'app-componente1',
  template: `
    <p>
      COMPONENTE 1
    </p>
  `,
  styleUrls: ['./componente1.component.css']
})
```

```
@Component({
  selector: 'app-tarjeta',
  standalone: true,
  template: `
    <div class="tarjeta">
      <h2>{{ titulo }}</h2>
      <p>{{ descripcion }}</p>
      <button (click)="like()">👍 Me gusta ({{ likes }})</button>
    </div>
  `,
  styles: [`
    .tarjeta {
      border: 2px solid #1976d2;
      border-radius: 10px;
      padding: 1rem;
      max-width: 300px;
      background-color: #f5faff;
      box-shadow: 0 2px 6px rgba(0,0,0,0.1);
    } `]
})
```

ANGULAR

Componentes: ciclo de vida

https://angular.dev/guide/components/lifecycle

ANGULAR

Phase	Method	Summary
Creation	`constructor`	Standard JavaScript class constructor. Runs when Angular instantiates the component.
Change Detection	`ngOnInit`	Runs once after Angular has initialized all the component's inputs.
	`ngOnChanges`	Runs every time the component's inputs have changed.
	`ngDoCheck`	Runs every time this component is checked for changes.
	`ngAfterContentInit`	Runs once after the component's *content* has been initialized.
	`ngAfterContentChecked`	Runs every time this component content has been checked for changes.
	`ngAfterViewInit`	Runs once after the component's *view* has been initialized.
	`ngAfterViewChecked`	Runs every time the component's view has been checked for changes.

ANGULAR

Rendering	`afterNextRender`	Runs once the next time that **all** components have been rendered to the DOM.
	`afterEveryRender`	Runs every time **all** components have been rendered to the DOM.
Destruction	`ngOnDestroy`	Runs once before the component is destroyed.

ANGULAR

- Componentes: ciclo de vida.
  - constructor→Inicialización de propiedades. Los **@Inputs** no están disponibles en este método.
  - ngOnChanges→Se ejecuta después del constructor y antes de ngOnInit. Se ejecuta cada vez que se modifica los @Inputs. Los **@Inputs** comienzan a estar disponible en este método. Admite la forma ngOnChanges(changes: SimpleChanges) que contiene los valores previos y actuales de un @Input.
  - ngOnInit→Acciones de arranque del componente. Viene como consecuencia de implementar OnInit. Los **@Inputs** están disponible en este método.
  - ngAfterViewInit()→Después de que la vista está construida (tras ngOnInit). Todo el HTML está creado y está disponible.
  - ngOnDestroy. Se ejecuta antes de que Angular destruya el componente. En este método se eliminan Observables y manejadores de eventos.

ANGULAR

Data binding

ANGULAR

- Data binding.
  - Mecanismos para enlazar los datos del modelo (TS) con los de los 'templates' (HTML).
    - Interpolación.
      - https://angular.dev/guide/templates/binding
    - Property binding.
      - https://angular.dev/guide/templates/binding
    - Event binding.
      - https://angular.dev/guide/templates/event-listeners
    - Two way binding.
      - https://angular.dev/guide/templates/two-way-binding

ANGULAR

Data binding: interpolación

ANGULAR

- Interpolación:
  - Referencia desde los templates a los valores de las clases.
  - Los atributos y las llamadas a los métodos se rodean con {{ }}.
  - Fichero ts:

```
export class Componente1Component implements OnInit {
  nombre:string="Fernando";
  constructor() { }
  getValor():number {
    return 5;
  }
  ngOnInit(): void {
  }
}
```

  - Fichero html:

```
<h1>Componente 1 {{nombre}}</h1>
<h2>{{getValor()}}</h2>
```

ANGULAR

- Interpolación:
  - Admiten expresiones ternarias:
    - {{ expresión_booleana ? valor_if_true : valor_is_false }}.
  - Fichero ts:

```
{{ factura.pagada ? 'PAGADA' : 'PENDIENTE' }}
```

  - "Equivalente" en JavaScript:

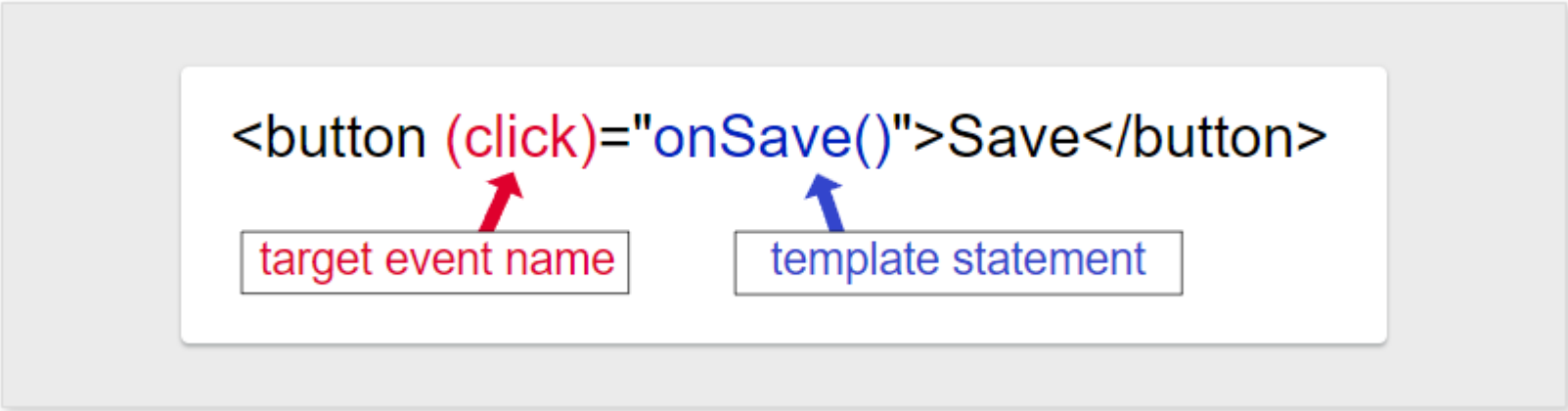
```
if (factura.pagada){
 console.log("Pagada");
} else {
  console.log("Pendiente");
}
```

ANGULAR

Data binding: property binding

ANGULAR

- Property binding:
  - **Utilización de atributos de las clases ts como valores de atributos html:**
    - `<p id="{{nombre_atributo}}">Texto del párrafo</p>`
    - `<p [id]="nombre_atributo">`
- Eventos:



ANGULAR

Data binding: events

ANGULAR

- Event binding:
  - Html:

```
<button (click)="saludar()">Saludar</button>
```

  - TS:

```
saludar():void {
    this.nombre+="*";
}
```

ANGULAR

- Eventos con captura:
  - Html:

```
<button (click)="saludar($event)">Saludar</button>
```

  - TS:

```
saludar(evento:any):void {
    this.nombre+="*";
}
```

ANGULAR

- Eventos con captura:
  - Html:

```
<select (change)="cambiar($event)">
    <option value="Uno">1</option>
    <option value="Dos">2</option>
    <option value="Tres">3</option>
</select>
```

  - TS:

```
cambiar(evento:any):void {
  console.log(evento.target.value);
}
```

ANGULAR

- Otros eventos:
  - (input)→Detecta la introducción de un carácter en un elemento de tipo <input>.
  - (focus)→El elemento HTML obtiene el foco.
  - (blur)→El elemento HTML pierde el foco.

ANGULAR

- Eventos de ratón:
  - (mousedown)
  - (mouseenter)
  - (mouseleave)
  - (mousemove)
  - (mouseout)
  - (mouseover)
  - (mouseup)
  - (mousewheel)

ANGULAR

Data binding: Decorador Input

https://angular.dev/api/core/Input

ANGULAR

- Decorador @Input:
  - Permite comunicar el elemento contenedor (padre) con el elemento contenido (hijo) a través de atributos.
  - En el elemento contenido se debe importar **Input**.
  - En el elemento contenido se debe "decorar" el atributo con el decorador @Input():
    - `@Input() nombre:string="Fernando";`
    - Opcionalmente se puede asignar un alias @Input("elnombre")
  - En el elemento contenedor se debe agregar al componente una propiedad con el nombre del atributo (o el alias).
    - `<app-componente1 nombre="Fulano"></app-componente1>`
  - Los valores pasados como parámetros no están disponibles en el constructor, pero sí en ngOnInit.

ANGULAR

- Decorador @Input:
- Permite la aplicación de “property binding”.
  - <app-componente1 nombre="{{nombre}}"></app-componente1>
  - <app-componente1 [nombre]="nombre"></app-componente1>

ANGULAR

Data binding: Decorador Output

https://angular.dev/api/core/output

ANGULAR

- Decorador @Output:
  - Permite comunicar al elemento contenido (hijo) con el elemento contendor (padre).
  - Son emisores de eventos.
  - En el elemento contenido se debe importar **Output** y **EventEmitter**.
  - En el elemento contenido se debe declarar el emisor e inicializarlo en el constructor:

```
@Output() eventoPrueba: EventEmitter<string>;

constructor() {
   this.eventoPrueba = new EventEmitter<string>();
}
```

  - En el lugar adecuado, se realiza la emisión del evento. El contenido debe coincidir con el tipo indicado en la declaración del evento:

```
this.eventoPrueba.emit("Un string");
```

ANGULAR

- Decorador @Output:
  - En el HTML contenedor, se debe asociar el evento a la ejecución de un método enviando como parámetro $event para que llegue la información:

```
<app-componente1 [nombre]="nombre"
(eventoPrueba)="metodoTS($event)"></app-componente1>
```

  - Este TS contenedor deberá estar programado el método.

```
export class AppComponent {
  title = 'proyecto_demo';
  nombre = "Mengano";

  metodoTS(info:any):void{
    console.log("Output ejecutado:” + info);
  }
}
```

ANGULAR

Data binding: Two Ways Binding

https://angular.dev/api/core/output

ANGULAR

- Two way binding
  - Enlaza un atributo con un elemento HTML en los dos sentidos. Si se modifica en la página se modifica en el modelo y viceversa.
  - En **app.module.ts** realizar el siguiente import:
    - `import { FormsModule } from '@angular/forms';`
  - En **app.module.ts** incluir el módulo importado en el array de imports:

```
imports: [
  BrowserModule,
  FormsModule
]
```

  - En el fichero HTML incluir las referencias:

```
<input type="text" [(ngModel)]="nombreAtributo">
```

  - En el fichero TS incluir el atributo correpondiente (nombreAtributo)