

DESARROLLO DE APLICACIONES CON ANGULAR

Angular

1

ANGULAR

■ Angular


- Framework de desarrollo de aplicaciones web SPA.
- Desarrollado en TypeScript.
- Código abierto.
- Desarrollado y mantenido por Google.
- <https://angular.io/>
- Requisito previo: tener instalado node.js
- Instalación:
 - `npm install -g @angular/cli`
- Verificación de versión:
 - `ng version`





ANGULAR

■ Creación de proyecto:

- Recomendable: tener instalado git  git
- `ng new nombre_proyecto`

■ Arranque del servidor:

- `ng serve`
- `ng serve -o` → Abre el navegador
- `ng serve --port 8082` → Arranca en el puerto indicado

angular.json

```
  },  
  "serve": {  
    "builder": "@angular-devkit/build-angular:dev-server",  
    "options": {  
      "port": 8081  
    },  
  },  
  "configurations": {  
    "production": {  
      "browserTarget": "proyecto_demo:build:production"
```



ANGULAR

■ Comandos :

- `ng add` → Instalación de librerías externas
 - Ejemplo: `ng add @angular/localize`
- `ng build` → Compilación y generación de la distribución en la carpeta dist
- `ng generate (g)` → Generación de componentes.
- `ng lint (l)` → Analizador del código existente.
- `ng new (n)` → Generación de proyecto (workspace)
- `ng serve (s)` → Construcción y ejecución del proyecto.
- `ng update` → Actualiza el proyecto.
- `ng test` → Ejecuta los test.



ANGULAR

■ Contenido del proyecto:

- `angular.json` → Configuración del proyecto.
- `karma.conf.js` → Configuración del entorno de pruebas karma.
- `package.json` → Configuración de dependencias.
- `package-lock.json` → Información sobre las versiones de los paquetes instalados.
- `tsconfig.json` → Configuración del compilador de TypeScript.
- `tsconfig.app.json` → Hereda de `tsconfig.json`. Configuración a nivel de aplicación.
- `tsconfig.spec.json` → Hereda de `tsconfig.json`. Configuración referente a los test.
- `src/` → Ficheros fuentes del nivel raíz.



ANGULAR

■ Contenido del proyecto:

- src/ → Ficheros fuentes del nivel raíz.
 - index.html → Página principal con el elemento `<app-root>`
 - main.ts → Arranque de la aplicación.
 - app/:
 - app-routing-module.ts. Fichero de routing.
 - app.component.css. Css del componente principal de la aplicación..
 - app.component.html. Html del componente principal de la aplicación.
 - app.component.spec.ts. Fichero con los test del componente principal de la aplicación.
 - app.component.ts. Fichero TypeScript de la aplicación.
 - app.module.ts. Contiene la estructura de la aplicación.



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:
 - Modifica **app.module.ts** para incluir el componente.
 - `nombre_componente.component.css`. Hoja de estilos del componente.
 - `nombre_componente.component.spec.ts`. Test unitarios del componente.
 - `nombre_componente.component.ts`. Fichero TypeScript del componente.
 - `selector: 'nombre_selector'` → El nombre del selector permite 'incrustar' el componente en la página html como `<nombre_selector></nombre_selector >`.

<https://angular.io/cli/generate#component>



ANGULAR

■ Creación de componentes:

- Fichero ts del componente:
 - Selector
 - **template (código directamente)** ó templateUrl (enlace al fichero).

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

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




ANGULAR

■ Creación de componentes:

- Fichero ts del componente:
 - Selector
 - `template` (código directamente) ó **`templateUrls` (enlace al fichero).**

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

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



ANGULAR

■ Creación de componentes:

- Fichero ts del componente:
 - Selector
 - styles (código directamente) ó **styleUrls (enlace a ficheros).**

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

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



ANGULAR

- Componentes: ciclo de vida.
 - constructor → Inicialización de propiedades. Los **@Inputs** no están disponibles en este método.
 - ngOnChanges → Se ejecuta después del constructor. 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.

<https://angular.io/guide/lifecycle-hooks>



ANGULAR

- Métodos ciclo de vida.
 - ngOnInit.

```
ngOnInit(): void {  
    let objeto = this;  
    setInterval(function() {  
        objeto.contador++;  
    }, 1000);  
}
```

```
ngOnInit(): void {  
    setInterval(() => {  
        this.contador++;  
    }, 1000);  
}
```



ANGULAR

- Data binding.
 - Mecanismos para enlazar los datos del modelo (TS) con los de los 'templates' (HTML).
 - Interpolación.
 - Property binding.
 - Event binding.
 - Two way binding.



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' }}
```



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:

```
<button (click)="onSave()">Save</button>
```

target event name

template statement



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

- 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

■ Decorador @Output:

- Permite comunicar el elemento contenedor (padre) con el elemento contenido (hijo) a través de atributos.
- 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-component1 [nombre]="nombre"  
(eventoPrueba)="metodoTS($event)"></app-component1>
```

- 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

■ 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 correspondiente (nombreAtributo)