APUNTES

COMANDOS ANGULAR

Para crear un nuevo proyecto -> ng n nombre-del-proyecto

Para crear un nuevo componente -> ng g c nombre-del-componente

Para crear una nueva directiva -> ng g d nombre-de-la-directiva

Para crear un nuevo servicio -> ng g s nombre-del-servicio

Para visualizar tu proyecto -> ng serve -o

Para recuperar los node-modules -> npm i

TIPOS DE DATOS EN ANGULAR (TypeScript)

```typescript
// Variable numérica
numero: number = 5;

// Variable de texto
texto: string = 'Hola mundo';

// Variable booleana
esVerdadero: boolean = true;

// Variable de fecha
fecha: Date = new Date();

// Variable de array
lista: number[] = [1, 2, 3, 4, 5];

// Variable de objeto
persona: {nombre:string, edad: number} = { nombre: 'Juan', edad: 25 };
```

FICHEROS MODELOS

Hablamos de **Ficheros Modelo** cuando almacenamos clases de objetos en un fichero .ts para más tarde utilizarlas en cualquier componente de la aplicación.

Normalmente, los ficheros modelos se crean en una carpeta llamada Models.

1. Creación de la clase en un fichero modelo (alumno.model.ts)

```
export class Alumno {
    id: number;
    nombre: string;
    curso: string;

    constructor(id: number, nombre: string, curso: string) {
        this.id = id;
        this.nombre = nombre;
        this.curso = curso;
    }
}
```

2. Importar la clase en el componente donde se desea usar e instanciar un objeto de dicha clase.

```
import { Alumno } from './alumno.model';
```

```
let miAlumno = new Alumno(1, "Juan", "Desarrollo Web")
```

DATA BINDING – INTERPOLATION

Interpolation es una técnica en Angular que permite la inclusión de expresiones dentro de las etiquetas HTML. Permite mostrar información de manera dinámica en la vista. Se representa con doble llaves **{{ }}**

```
<p>Hola, {{ nombre }}</p>
```

DATA BINDING - PROPERTY BINDING

Property Binding nos permite modificar de manera dinámica el valor de los atributos de una etiqueta HTML, la cual debemos poner entre corchetes [ ], el dato que va entre comillas " ", sería el nombre de la variable que se encuentra en el fichero TS

```
<img [src]="imagenUrl">
```

```
imagenUrl: string = 'https://ejemplo.com/imagenes/mi-imagen.jpg';
```

DATA BINDING - EVENT BINDING

Event Binding nos permite enlazar un evento de un elemento HTML con una expresión/método definido en el fichero TS.

1. En el TS tenemos definida la variable texto y el método modificarTexto(), el cual estará asociado a un botón en el HTML, el método es llamado cuando se hace click en el botón

```
texto: string = 'Hola Mundo';

modificarTexto() {
  this.texto = 'Bienvenidos a mi Mundo';
```

2. En el HTML tenemos un botón con un evento (click) asociado al método modificarTexto() y la variable texto interpolada. El valor inicial de la variable es "Hola Mundo", esta cambiará a "Bienvenidos a mi Mundo" una vez se haga click en el botón.

```
<button (click)="modificarTexto()">Modificar Texto</button>
<p>{{texto}}</p>
```

DIRECTIVA DE ATRIBUTO - [(ngModel)]

Para realizar un **Two-Way Binding** o **Binding Bidireccional** usamos la directiva **[(ngModel)]**
Para hacer uso de esta directiva debemos importarla en el app.module.ts

```
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    FormsModule,
    // Otros módulos...
  ],
  // ...
})
```

1. En el fichero TS tenemos la variable miNombre declarada e inicializada con el valor “Victor”

```
miNombre: string = 'Victor';
```

2. En el fichero HTML tenemos dicha variable interpolada, y una etiqueta input con un **Two-Way Binding** a la variable "miNombre" usando **[(ngModel)]**.
Cualquier cambio en el input será reflejado en el componente y viceversa, además se tiene un párrafo que muestra el valor de la variable "miNombre" para poder ver el cambio.

```
<input [(ngModel)]="miNombre">
<p>{{ miNombre }}</p>
```

DIRECTIVA DE ATRIBUTO – [ngClass]

[ngClass] es una directiva de atributo de Angular que permite aplicar una o varias clases CSS a un elemento HTML en función de una expresión booleana, se evalua una condición y en base a esta aplica una clase CSS.

1. Declaramos e inicializamos la variable a evaluar en el fichero TS

```
nota: number = 7;
```

2. En el fichero HTML

```
<p [ngClass]="{aprobado: nota >= 5, suspenso: nota < 4}">{{ nota }}</p>
```

3. En el fichero CSS tenemos las clases .aprobado y .suspenso

```
.aprobado {
    color: green;
}
.suspenso {
    color: red;
}
```

De esta forma estamos agregando una clase CSS a la etiqueta que contiene la directiva **[ngClass]** en base al valor de la variable “nota”.

DIRECTIVA DE ATRIBUTO – [ngStyle]

[ngStyle] nos permite modificar de manera dinámica los estilos de un elemento HTML en base a una expresión (normalmente un método).

1. Creamos el método/función y una variable en el fichero TS.

```
nota: number = 5;

setColor(){
  if (this.nota < 5) {
    return "red";
  }else{
    return "green";
  }
}
```

2. En el HTML del componente

```
<p [ngStyle]="{'color': setColor()}">{{nota}}</p>
```

Con esto se está llamando al método **setColor()** y se está retornando el color en base a la condición de la nota.

DIRECTIVA DE ESTRUCTURA - *ngFor

La directiva ***ngFor** permite iterar sobre un array y/o objeto para poder mostrar sus atributos/valor/datos en etiquetas HTML.

Ejemplo 1:

La variable “lista” que es un Array de Strings, está declarada e inicializada en el fichero TS del componente.

```
lista = ['Dato 1', 'Dato 2', 'Dato 3', 'Dato 4'];
```

En este ejemplo se está utilizando *ngFor para iterar sobre una variable(Array) llamada "lista" y mostrando cada elemento dentro de una etiqueta <li>.

```
<ul>
  <li *ngFor="let item of lista">{{ item }}</li>
</ul>
```

Ejemplo 2:

La variable “lista” que es un Array de Objetos, está declarada e inicializada en el fichero TS del componente.

```
lista = [
  {nombre: 'Juan', edad: 25},
  {nombre: 'Maria', edad: 30},
  {nombre: 'Pedro', edad: 35},
];
```

En este ejemplo se está utilizando la directiva ***ngFor** para iterar sobre el array "lista" y mostrar cada propiedad de cada objeto en una etiqueta <p> (nombre y la edad de cada persona).

```
<div *ngFor="let persona of lista">
  <p>{{ persona.nombre }} - {{ persona.edad }} años</p>
</div>
```

DIRECTIVA DE ESTRUCTURA - *ngIf

La directiva ***ngIf** nos permite evaluar una condición en el fichero HTML, para hacer que un elemento se muestre o no.

Ejemplo:

La variable “lista” que es un Array de Objetos, esta declarada e inicializada en el fichero TS del componente.

```
lista = [
  {nombre: 'Juan', edad: 25},
  {nombre: 'Maria', edad: 30},
  {nombre: 'Pedro', edad: 35},
];
```

En este ejemplo se está utilizando la directiva ***ngFor** para iterar sobre el Array de Objetos "lista", y para cada objeto se está utilizando la directiva ***ngIf** para comprobar si la edad es mayor a 30 años, si es así, se muestra el nombre y la edad de cada persona, si no se oculta.

```
<div *ngFor="let persona of lista">
  <div *ngIf="persona.edad > 30">
    <p>{{ persona.nombre }} - {{ persona.edad }} años</p>
  </div>
</div>
```

DIRECTIVA DE ESTRUCTURA - *ngSwitch

La directiva ***ngSwitch** permite seleccionar un conjunto de elementos HTML para mostrar en función de un valor específico. Es similar al uso de un **switch** en programación, se especifica un valor para comparar y se seleccionan los elementos a mostrar dependiendo del valor. Se pueden especificar varios casos con ***ngSwitchCase** y un caso default con ***ngSwitchDefault**.

El elemento HTML donde se coloque la directiva ngSwitch, será el contenedor donde se coloquen los elementos a mostrar dependiendo del valor especificado.

Ejemplo:

La variable “lista” que es un Array de Objetos, esta declarada e inicializada en el fichero TS del componente.

```
lista = [
  {nombre: 'Juan', edad: 25},
  {nombre: 'Maria', edad: 30},
  {nombre: 'Pedro', edad: 35},
];
```

En este ejemplo se está utilizando la directiva ***ngFor** para iterar sobre el array "lista" y para cada objeto se esta utilizando la directiva ***ngSwitch** para comparar el valor de la edad con los casos 25, 30 y 35. Si la edad coincide con alguno de esos valores se muestra el nombre y la edad de cada persona, si no se usa el caso default para mostrar un mensaje.

```
<div *ngFor="let persona of lista">
  <div [ngSwitch]="persona.edad">
    <p *ngSwitchCase="25">{{ persona.nombre }} - {{ persona.edad }} años</p>
    <p *ngSwitchCase="30">{{ persona.nombre }} - {{ persona.edad }} años</p>
    <p *ngSwitchCase="35">{{ persona.nombre }} - {{ persona.edad }} años</p>
    <p *ngSwitchDefault>{{ persona.nombre }} - edad no es 25, 30 o 35</p>
  </div>
</div>
```

DECORADOR @Input – PADRE -> HIJO

El decorador **@Input** en Angular se utiliza para pasar datos desde un componente padre a un componente hijo.

Por ejemplo, si un componente padre tiene una propiedad (Variable) llamada "mensajePadre" y desea pasar ese valor a un componente hijo, puede hacerlo de la siguiente manera:

1. En el fichero HTML del componente padre “enviamos” el valor de la propiedad/variable incluyéndolo en el selector del componente hijo.

   ```
   <app-hijo [mensaje]="mensajePadre"></app-hijo>
   ```

   [mensaje] hace referencia al nuevo nombre del dato que estamos enviando.
   “mensajePadre” es el nombre de la variable/propiedad que contiene el el dato a enviar.

2. En el fichero TS del componente hijo “recibimos” el dato.

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

   @Input() mensaje: string;
   ```

   Importamos **{ Input }**
   Creamos un **@Input() mensaje: string;**

   El nombre de la variable asociada al decorador @Input ha de ser el mismo que usamos en el selector a la hora de enviar los datos `[mensaje]="mensajePadre"`:

   Es importante hacer coincidir el tipo de dato que recibe con el que se envía.

DECORADOR @Output – HIJO -> PADRE

El decorador **@Output** en Angular se utiliza para emitir eventos desde un componente hijo a un componente padre.

Por ejemplo, tenemos un Array de Objetos en la variable **"students"** declarada e inicializada en el fichero TS del componente hijo y queremos enviarlo al componente padre.

EN EL COMPONENTE HIJO:

1. Importar los módulos necesarios en el fichero TS.

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

2. Tener la variable declarada e inicializada en el fichero TS.

```
let students = [
    { name: 'John Smith', age: 25, grade: 85 },
    { name: 'Jane Doe', age: 22, grade: 90 }
];
```

3. Declarar una propiedad/variable decorada con **@Output** para emitir el evento.

```
@Output() studentsChanged = new EventEmitter<any[]>();
```

4. Crear una función que emita el evento cuando sea necesario, en este caso cuando se quiera enviar el array **"students"** al componente padre.

```
sendStudents() {
  this.studentsChanged.emit(this.students);
}
```

5. En el HTML utilizar la función **sendStudents()** para emitir el evento cuando sea necesario. Por ejemplo, cuando se presione un botón.

```
<button (click)="sendStudents()">Enviar estudiantes</button>
```

DECORADOR @Output – HIJO -> PADRE

EN EL COMPONENTE PADRE:

1. En el HTML, utilizamos la sintaxis **(nombreEvento)="handler"** para recibir el evento emitido por el componente hijo.

   ```
   <app-hijo (studentsChanged)="recibirEstudiantes($event)"></app-hijo>
   ```

   Estamos llamando al **@Output studentsChanged** el cual se encarga de emitir el evento, igualado a la función encargada de enviar el dato (array **"students"**) con **$event** como parámetro.

2. En el TS, declaramos una función para manejar el evento recibido y asignar el valor del array **"students"** a una variable.

   ```
   students: any[];

   recibirEstudiantes(students: any[]) {
     this.students = students;
   }
   ```

De esta forma, cuando se presione el botón en el componente hijo, se emitirá el evento **"studentsChanged"** y el componente padre recibirá el evento y asignará el valor del array **"students"** a la variable **"students".**

SERVICIOS

En Angular, un **servicio** es una clase que proporciona funcionalidades específicas que pueden ser reutilizadas a través de varios componentes.

Para crear un servicio en Angular, se debe utilizar el comando:
ng generate service nombre-del-servicio

1. En la terminal, escribimos el comando **ng generate service servicios/alumnos** para crear un servicio llamado **"AlumnosService"** dentro de una carpeta llamada **"servicios"**. Este comando generará un archivo llamado **"alumnos.service.ts"** dentro de la carpeta "servicios" con la estructura básica del servicio.

2. En el archivo **"alumnos.service.ts"** declaramos una variable llamada **"students"** con el array de objetos de alumnos y una función que retorna dicho array.

```
let students = [
    { name: 'John Smith', age: 25, grade: 85 },
    { name: 'Jane Doe', age: 22, grade: 90 }
];
 getEstudiantes() {
   return this.students;
 }
```

3. Importamos el servicio en el TS del componente donde queremos utilizarlo.

```
import { EstudiantesService } from './servicios/estudiantes.service';
```

4. Inyectamos el servicio en el constructor del componente e inicializamos una variable donde vamos a guardar los datos.

```
students: any[];

constructor(private estudiantesService: EstudiantesService) {
  this.students = this.estudiantesService.getEstudiantes();
}
```

ROUTING

El **routing** en Angular es una de las características más importantes de la plataforma.
Permite dividir una aplicación en diferentes secciones o páginas, y navegar entre ellas de manera sencilla.

Pasos para incluir routing en un proyecto Angular:

1. Importar los módulos RouterModule y Router en el fichero *app.module.ts*

```
import { RouterModule, Routes } from '@angular/router';
```

2. Crear el array que contiene las rutas disponibles en la aplicación

```
const rutas: Routes = [
  { path: '', component: HomeComponent },
  { path: 'portfolio', component: PortfolioComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'display/:ref', component: DisplayComponent },
  { path: 'precio/:ref', component: PrecioComponent }
];
```

3. Declarar el sistema de routing en el *"imports"* del decorador **@NgModule**

```
@NgModule({
  imports: [RouterModule.forRoot(rutas)],
})
```

4. Definir el template del componente principal en el fichero *app.component.htm*

   <router-outlet> es el lugar donde los componentes asociados a las rutas se renderizan en la página.
   Cuando un usuario hace clic en un enlace de navegación, la ruta activa cambia y el componente correspondiente se muestra en el *<router-outlet>*.

```
<nav id="header--nav">
  <ul>
    <li><a routerLink="/">HOME</a></li>
    <li><a routerLink="/portfolio">PORTFOLIO</a></li>
    <li><a routerLink="/about">ABOUT</a></li>
    <li><a routerLink="/contact">CONTACT</a></li>
  </ul>
</nav>
<router-outlet></router-outlet>
```

Objeto ‘Navigate’

Sirve para redirigir a los usuarios a otras vistas dentro de la misma aplicación. Podemos llevarnos datos a la misma.

1. Importar el módulo ‘Router’ y añadirlo al constructor del componente

```
import { Router } from '@angular/router';

constructor(private router: Router) {}
```

2. Crear un método/función que nos redirige al componente junto al parámetro

```
display(cocina: number) {
  this.router.navigate(['/display', cocina]);
}
```

3. Asociar el método/función a un elemento HTML para realizar la acción

```
<img (dblclick)="display(cocina.ref)"/>
```

En el caso de:

```
{ path: 'display/:ref', component: DisplayComponent }
```

Define una ruta que permite acceder a la vista de del componente ‘Precios’ y permite pasar un parámetro dinámico a través de la URL (:’ref’).

En dicho componente debemos seguir los siguientes pasos:

1. Crear una variable donde se va a guardar el valor del parámetro indicado en la ruta.

```
refCocina: any;
```

2. Importar el módulo **‘ActivatedRoute’** y añadirlo al constructor del componente.

```
import { ActivatedRoute } from '@angular/router';

constructor(private activateRoute: ActivatedRoute) {}
```

3. Obtener el valor del parámetro ‘ref’ y guardarlo en la variable ‘refCocina’

```
ngOnInit(): void {
  this.refCocina = this.activateRoute.snapshot.params['ref'];
}
```

Objeto ‘Location’

Se utiliza para interactuar con el URL de la aplicación y realizar acciones como ir hacia atrás/adelante en el historial del navegador, o actualizar el URL sin recargar la página.

1. Importar el módulo ‘Location’ y añadirlo al constructor del componente

```
import { Location } from '@angular/common';

constructor(private location: Location) { }
```

2. Crear un método/función que realice la acción

```
back() {
  this.location.back();
}
```

3. Asignar dicho método/función a un elemento HTML

```
<button (click)="goBack()">Go Back</button>
```