Material de estudio OBLIGATORIO EJE 2 Angular - Enlace de Datos, Directivas, Pipes e Intro a Servicios

Sitio:	Instituto Superior Politécnico Córdoba	Imprimido por:	Braian TRONCOSO
Curso:	Programador Web - TSDWAD - 2022	Día:	lunes, 15 mayo 2023, 11:36 PM
Libro:	Material de estudio OBLIGATORIO EJE 2 Angular - Enlace de Datos, Directivas, Pipes e Intro a Servicios		

Tabla de contenidos

1. Data Binding

Data Binding (enlace de datos)

Data Bingind es la forma en que el Angular nos permite mostrar el contenido dinámico en lugar de contenido estático, lo que se traduce como "comunicación" entre el código HTML (plantilla o template) y la lógica de programación (archivo .ts).

Es decir que, nos abstrae de la lógica get/set asociada a insertar y actualizar valores en el HTML y, de convertir las respuestas de usuario (inputs, clics, etc) en acciones concretas.

Escribir toda esa lógica antes, era tedioso y propenso a errores dado que debíamos trabajar con JavaScript (o alguna librería como por ej. jquery).

1.1. Tipos de Binding

Angular proporciona las siguientes categorías de enlace de datos sesgún la dirección del flujo de datos:

1. Interpolación: {{expression}}
2. Enlace de propiedad: [target]="expression"
3. Enlace de evento: (target)="statement"
4. Enlace bidireccional: [(target)]="expression"

En una imagen:

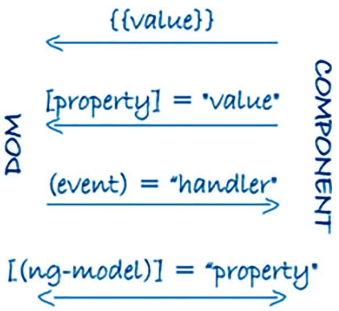


Figura: Formas de Data Binding
Fuente: https://docs.angular.lat/generated/images/guide/architecture/databinding.png

Observa que los binding types (a diferencia de los de interpolación) poseen un target (nombre) y están entre [], () o ambos [()].

Usa:

- () para enlazar del DOM al componente.
- [] para enlazar desde el componente al DOM.
- [()] para enlazar en ambos sentidos.

El target de un databinding puede ser una propiedad, un atributo, un evento, etc. Cada elemento público de una directiva está disponible para el binding en un template como se puede observar en el siguiente ejemplo:

Tipo	Target	Ejemplo
Property	src (element property) hero (component property) ngClass (directive property)	<img [src]="heroImageUrl"> <app-hero-detail [hero]="currentHero"> </app-hero- detail> <div [ngClass]="{'special': isSpecial}">
Event	click (element event) deleteRequest (component event)	<button (click)="onSave()">Save</button> <app-hero-detail (deleteRequest)="deleteHero()"
	myClick (directiva event)	/app-hero-detail> <div (myClick)="clicked=$event" clickable>click me
Two-way	Event y Property	<input [(ngModel)]="name">
Attribute	Attribute	<button [attr.aria-label]="help">help</b utton>
Class	Class property	<div [class.special]="isSpecial">Spec ial

Style	Style property	<button [style.color]="isSpecial ? 'red' : 'green'">

Fuente: https://angular.io/guide/binding-syntax

En resumen:

Tipo	**Sintaxis**	**Categoría**
Interpolation Property Attribute Class Style	{{expression}} [target]="expression" bind-target="expression"	One-way Desde el componente hacia el DOM
Event	(target)="statement" on-target="statement"	One-way Desde el DOM hacia el componente
Two-way	[(target)]="expression" bindon-target="expression"	Two-way En ambos sentidos

Tabla 1: Tipos de Data Binding
Fuente: https://angular.io/guide/binding-syntax

Interpolation

La interpolación es un mecanismo que nos provee Angular a fin de sustituir una expresión por un valor de cadena en el template (o vista). Es decir que, permite el flujo de datos desde el componente hacia el DOM.

Se utiliza generalmente, cuando recibimos datos del backend a través de los servicios, los cuales son visualizados en la vista (de lectura) pero no deben ser modificados dado que existen otros medios para ello disponibles en la aplicación.

Es decir, cuando Angular detecta la interpolación , la evalúa y trata de convertirla en una cadena, para luego renderizar en el template (desde el componente hacia el DOM)

Ejemplo:

Esto es una **{{interpolacion}}**

Para entenderlo mejor, modifiquemos la vista del headerComponent de nuestra aplicación a fin de que el mensaje de bienvenida sea enviado del componente a la vista mediante la interpolación:

Actual vista del headerComponent

```
<header class="fondo">
    <h1 class="mensaje-bienvenida">¡BIENVENIDOS a nuestra App!</h1>
    <h3>Es un placer ser parte de tu día a día</h3>
  </header>
```

Podríamos usar interpolación para modificar el mensaje de bienvenida desde el componente.

Para ello, ejecutar los siguientes pasos:

1. Editar el archivo header.component.ts a fin de agregar una nueva variable que contendrá el mensaje de bienvenida como sigue:

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

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {
  mensajeBienvenida="¡BIENVENIDOS a nuestra App!";
  constructor() { }

  ngOnInit() {
  }

}
```

2. Editar el archivo header.component.ts respetando la sintaxis de la interpolación como sigue:

```
<header class="fondo">
  <h1 class="mensaje-bienvenida">{{mensajeBienvenida}}</h1>
  <h3>Es un placer ser parte de tu día a día</h3>
</header>
```

Si ejecutas el comando ng-serve podrás ver que nada parece haber cambiado. Sin embargo, el texto del mensaje no es aportado por la vista sino por el componente.

Property Binding

Property Binding es un mecanismo que nos permite asignar valores a las propiedades de los elementos HTML y/ o directivas presentes en el template.

Se utiliza generalmente, cuando deseamos modificar de manera dinámica los atributos, clases, estilos y demás propiedades de un elemento html. Ej. la url de una imagen.

Para enlazar una propiedad de un elemento HTML debemos encerrar entre [] la propiedad del elemento HTML que deseamos configurar.

Ejemplos:

```
<img [src]="itemImagenUrl" alt="">
<span [innerHTML]="propertyTitle" ></span>
<p [class.border-danger]="!propertyValid"></p>
```

Veamos un ejemplo. Para ello, intentemos modificar la vista del Componente Servicio a fin de que el el texto Servicios tenga un background amarillo.

Para ello, ejecutar los siguientes pasos:

1. Editar el archivo servicios.component.ts a fin de agregar la variable que deseamos enlazar con el template como sigue:

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


@Component({
  selector: 'app-servicios',
  templateUrl: './servicios.component.html',
  styleUrls: ['./servicios.component.css']
})
export class ServiciosComponent implements OnInit {

  bgPropertyValid=true;
  constructor() { }


  ngOnInit(): void {
  }
```

Variable a enlazar

Nota: Observa que la variable para las clases debe ser booleana dado que, dependiendo de su estado: true o false, se aplicará o no el estilo.

2. Editar el archivo services.component.html a fin de incluir el enlace de propiedad.

```
<section class="mt-5" id="services">
    <div class="container">
        <div class="text-center">
                    Property Binding
            <h2 [class.bg-warning]="bgPropertyValid" class="text-uppercase">Servicios</h2>
        </div>
        <div class="row text-center">
            <div class="col-md-4">
                <span class="fa-stack fa-4x">
                    <i class="fas fa-circle fa-stack-2x text-warning"></i>
                    <i class="fas fa-laptop fa-stack-1x fa-inverse"></i>
                </span>
              ...
```

Si ejecutamos ng-serve, observamos el cambio en el ServiciosComponent

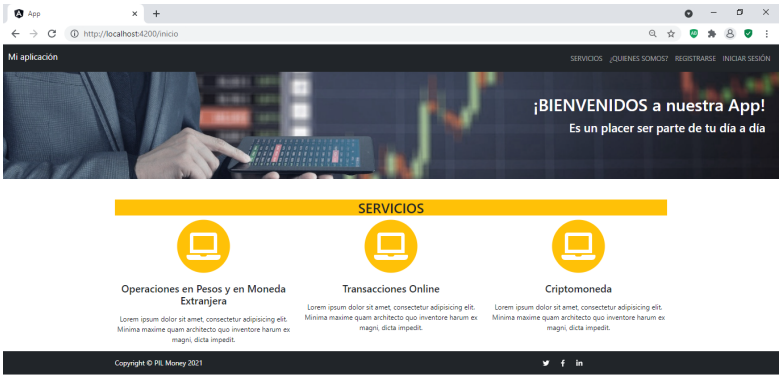


Figura: Background modificado en base al binding de tipo propiedad (property binding)

Event Binding

Es el mecanismo de data binding que nos permite trabajar con los eventos del DOM. El mismo permite además el flujo de datos del DOM hacia el componente.

Se usan generalmente en los formularios dado que permiten recuperar los datos que ingresó el usuario para así luego manipularlos y/o enviarlos al backend.

La sintaxis de un event binding es relativamente simple, basta con decorar el nombre del evento entre (). De esta manera, nos suscribimos al mismo. Además, debemos proporcionar el método a ejecutar cuando el evento suceda.

Ejemplo:

```
<button (click)="onSaludar()">Click aquí!</button>
```

Nota: es posible también subscribirnos al evento usando esta otra sintaxis:
<button on-click="onSaludar()">Pulse aquí</button>

De esta manera, el componente sabe que cuando el evento (click) se lance, se deberá ejecutar método onSave().

Nota: No te olvides de crear el método onSaludar()

¿Qué eventos podemos utilizar? Puedes econtrarlos en https://www.w3schools.com/jsref/dom_obj_event.asp ;

Veamos el ejemplo que simplemente muestre un alerta saludar. Para ello, ejecutar los siguientes pasos:

1. Editar el archivo servicios.component.html a fin de agregar un botón y subscribirnos al evento click como sigue:

```
<button class="btn btn-warning" (click)="onSaludar()">Click Aquí!</button>
```

2. Editar el archivo servicios.component.ts a fin de crear el método onSaludar como sigue:

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

@Component({
  selector: 'app-servicios',
  templateUrl: './servicios.component.html',
  styleUrls: ['./servicios.component.css']
})
export class ServiciosComponent implements OnInit {
  bgPropertyValid=true;
  constructor() { }

  ngOnInit() {
  }

  onSaludar()
  {
    alert("Hola Mundo!!");
  }
}
```

3. Ejecutar el comando ng serve, para ver y evaluar la aplicación en tiempo de ejecución.

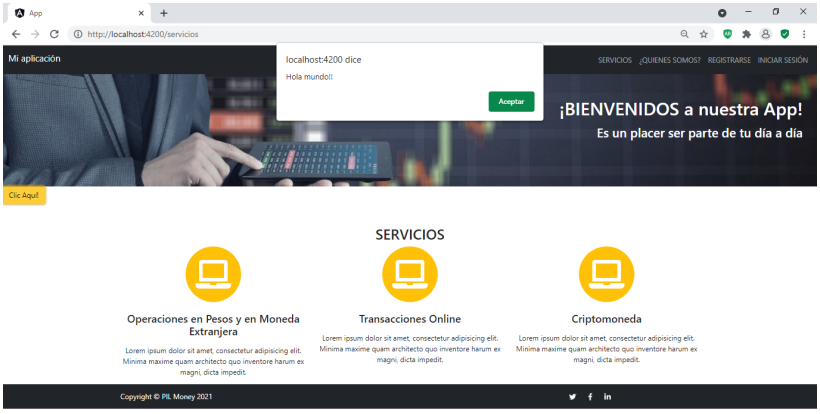
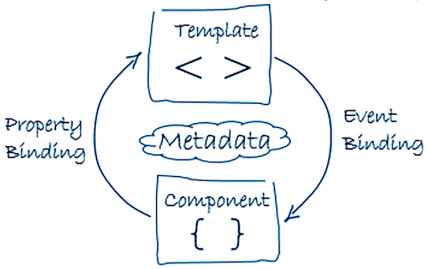


Figura: Event Binding

Modifica el ejemplo publicado en **stackbliz** (que te compartimos previemente) **a fin de aplicar el enlace de datos demostrado arriba.**

Two way binding

A menudo, se suele utilizar en conjunto Property Binding y Event Binding como se muestra en la siguiente imágen:



Fuente: https://docs.angular.lat/generated/images/guide/architecture/component-databinding.png

Sintaxis:
[(event)]=<<expresión>>

Ejemplo:

```
<p [(size)]="fontSizePx"></p>
```

En este tipo de binding, cualquier cambio relacionado con los datos en el archivo .ts siempre afecta a las vistas y los cambios realizados en las vistas también se ven reflejados en el .ts. Es decir que los cambios en cualquier lado de la aplicación se ven reflejados en el extremo contrario de manera inmediata y automática.

Expresiones

Angular nos permite trabajar con expresiones. Éstas se resolverán previo a devolver el resultado exactamente dónde la expresión se invocó.

Las mismas pueden:

- Escribirse entre llaves dobles: {{ expression }},
- En el interior de una directiva: ng-bind=" expression " .

Nota: las expresiones de angular son muy similares a las expresiones de JavaScript. Las mismas pueden contener literales, operadores y variables.(http://www.w3bai.com/es/angular/angular_expressions.html)

Ejemplo de expresiones:

{{propiedad_del_componente}}

{{ método_del_componente}

{{ 2020 + 1 }}

{{ ! valorBoleano)}}

Las expresiones son utilizadas al momento de realizar modificaciones en los elementos HTML presentes en los componentes de la aplicación y lo hacen en algunos casos mediante directivas. Concepto que abordaremos a continuación.

Puedes ver y ejecutar el código de un ejemplo sencillo en http://www.w3bai.com/es/angular/tryit.php?filename=try_ng_expression_2

2. Directivas

Son un mecanismo de angular para manipular el DOM por lo que, nos permite modificar el comportamiento de los elementos html, atributos, demás propiedades.

En otras palabras, las directivas son simplemente instrucciones interpretadas por el compilador, quién se encarga de recorrer el documento, localizarlas y ejecutar los comportamientos que se especifican en las mismas.

Directivas más comunes en Angular:

Figura: Tipos comunes de directivas.

Como vimos previamente, son las más utilizadas en Angular y son capaces de modificar:

- La vista HTML
- Un archivo .ts Typescript que define el comportamiento.
- Un selector CSS que define cómo es utilizado en el template.
- Opcionalmente un archivo .css que define el estilo del componente y un archivo .ts para las pruebas unitarias (https://docs.angular.lat/guide/built-in-directives).

Las directivas de atributos más comunes son:

- **ngClass.** Permite agregar o remover clases CSS en función de un estado o expresión de manera dinámica.
- **ngModel.** Permite Two-way binding (Desde/Hacia el DOM) de manera dinámica.
- **ngStyle.** Permite agregar o eliminar estilos de componente.

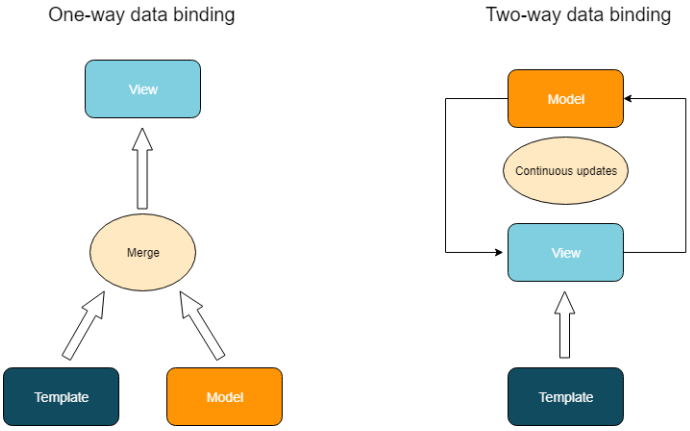


Figura: Representación gráfica de Binding en dos sentidos
Fuente: https://www.altexsoft.com/media/2018/10/One-and-two-way-data - bind.png

A continuación se describen las directivas ngClass y ngModel:

Directiva: ngClass

Permite establecer dinámicamente clases CSS en los elementos HTML por medio de las expresiones.

Ejemplo:

Supongamos que queremos cambiar el estado de un botón según esté apagado o encendido: OFF/ON:

En la vista:

```
<button class="btn" [ngClass]="{on: estadoPositivo, off:
!estadoPositivo}" (click)="cambiarEstado()">{{texto}}</button>
```

En el archivo ts:

```
estadoPositivo: boolean = true;
texto:string="si";

cambiarEstado()
{
  this.estadoPositivo = !this.estadoPositivo;
  if (this.estadoPositivo)
  {this.texto="si";}
  else
  {this.texto="no";}
}
```

Dónde on y off son clases CSS y, de acuerdo al estado de la variable estadoPositivo, el elemento button implementará una u otra clase.

Figura: ejemplo de ngClass

Directiva ngModel

Es un enlace que permite el flujo de datos entre la variable (componente) y el elemento de la vista (DOM) en ambos sentidos.

Ejemplo:

Si deseamos que el valor de la variable se muestre en un input type (one way binding):

```
<input type="text" [ngModel]="nombre">
```

Si deseamos que el valor de la variable se muestre en un input type pero además deseamos acceder luego al dato que el usuario ingresó (two way binding):

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

Nota: Se requiere implementar FormsModule y declarar la variable nombre en el archivo .ts

Directivas de estructura

Permiten manipular la estructura de la vista agregando o eliminando elementos de una manera muy sencilla.

A continuación se describen las directivas ngIf, ngFor y ngSwitch:

Directiva ngFor

***ngFor:** es una directiva repetidora que permite recorrer un array y para cada uno de sus elementos replicar una cantidad de elementos en el DOM.

Sintaxis:

***ngFor**="expresion"

Ejemplo:

Continuando con nuestro ejemplo y para observar cómo funciona el ngFor, ahora vamos a crear una tabla de movimientos en nuestra aplicación.

Para ello, ejecutar los siguientes pasos:

1. Crear el componente movimientos. Para ello, ejecutar: ng g c pages/movimientos

2. En el archivo movimientos.component.ts crear una variable movimientos como sigue (por ahora hardcodeada, luego deberemos reemplazar con los datos que vienen del backend)

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

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

Variable movimientos

```
  movimientos=[{operacion:"Extracción",monto:1500}, {operacion:"Depósito", monto:1520}];
  constructor() { }

  ngOnInit(): void {
  }
}
```

3. En el archivo movimientos.component.html completar las filas de la tabla haciendo uso del ngFor:

```
<h1 class="display-5">Últimos Movimientos</h1>
<table class="table">
    <thead>
        <th>Operación</th>
        <th>Monto</th>
    </thead>
    <tbody>
```

directiva ngFor

```
        <tr *ngFor="let element of movimientos" >
            <td>{{element.operacion}}</td>
            <td>{{element.monto|currency}}</td>
        </tr>
    </tbody>
</table>
```

4. Ejecutar el comando ng serve (si no está corriendo el servidor), y luego ir a http://localhost:4200/home/movimientos para ver y evaluar la aplicación en tiempo de ejecución.

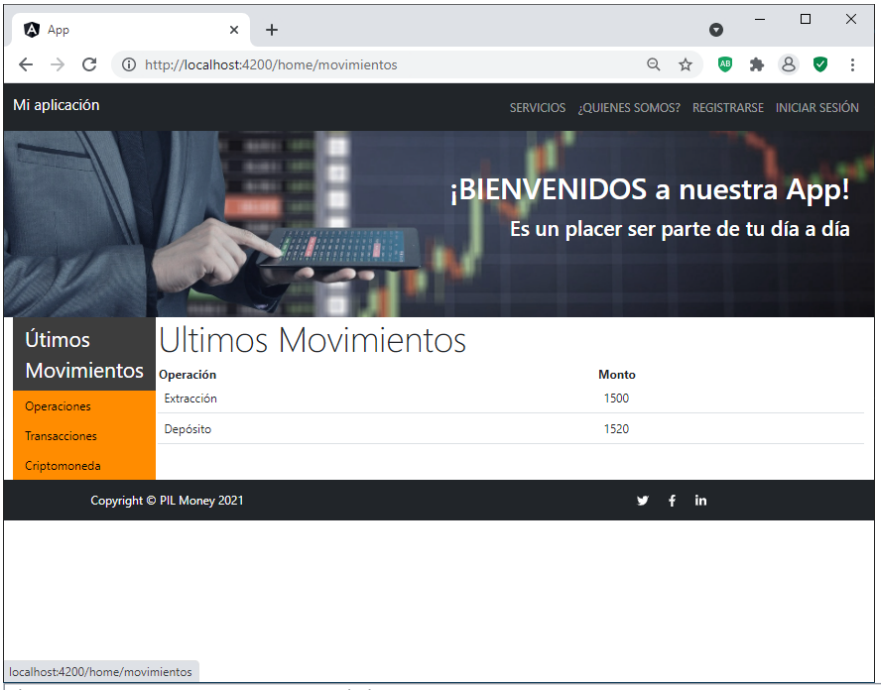


Figura: Uso del ngFor - Tabla movimientos

Nota: No te olvides que para que funcione hay que configurar las rutas.

Modifica el ejemplo publicado en stackbliz (que te compartimos previemente) a fin de agregar el componente de movimientos junto a la directiva ngFor.

Directiva ngIf

***ngIf:** permite ocultar o mostrar un elemento html a partir de una expresión.

Sintaxis:

***ngIf="expresion"**

Ejemplo:

Continuando con nuestro ejemplo y para observar cómo funciona el ngIf, ahora vamos a crear variable que nos permita ver esa tabla de movimientos.

Para ello, ejecutar los siguientes pasos:

1. En el archivo movimientos.component.ts crear una variable booleana que utilizaremos luego para mostrar o no la tabla movimientos

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

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

Variable booleana

```
  mostrarMovimientos: boolean=true;
  movimientos=[{operacion:"Extracción",monto:1500}, {operacion:"Depósito", monto:1520}];
  constructor() { }

  ngOnInit(): void {
  }
}
```

2. En el archivo movimientos.component.html agregar la directiva ngIf para mostrar o no la tabla:

```
<h1 class="display-5">Últimos Movimientos</h1>
```

Directiva ngIf

```
<table *ngIf="mostrarMovimientos" class="table">
    <thead>
        <th>Operación</th>
        <th>Monto</th>
    </thead>
    <tbody>
        <tr *ngFor="let element of movimientos" >
            <td>{{element.operacion}}</td>
            <td>{{element.monto}}</td>
        </tr>
    </tbody>
</table>
```

Luego, dependiendo de la variable se mostrará o no la tabla.

Desafío. Modifica el ejemplo publicado en stackbliz (que te compartimos previemente) a fin de agregar ocultar los enlaces de Registro e Inicio de sesión mediante la directiva ngIf si el usuaro está autenticado y mostrar el enlace "Cerrar Sesión".

Nota: La funcionalidad de login la veremos más adelante por lo que sólo, deberás configurar un atributo en el componente y dependiendo de su valor se muestren unos u otros enlaces en la barra de navegación.

Directiva ngSwitch

***ngSwitch:** Muestra u oculta un elemento entre varios dependiendo de una condición específica.

Sintaxis:

- **[ngSwitch]="expresion"** //se define en el contenedor
- ***ngSwitchCase="valor numérico"** //se define en el elemento a mostrar
- ***ngSwitchDefault** //se define en el elemento por defecto (si ninguna de las anteriores se cumple)

Ejemplo:

Continuando con nuestro ejemplo, podemos observar el componente integrante creado previamente.

El mismo cuenta con la directiva ngSwitch para mostrar el perfil de uno u otro integrante dependiendo del parámetro recibido en la ruta:

Directiva ngSwitch

```
<div [ngSwitch]="id">
    <div *ngSwitchCase="1" class="container">
        <div class="row">
            <div class="col">
                <img   src="assets/img/team/1.jpg" alt="">
            </div>
            <div class="col">
                <h1 class="mt-5 display-4 ">Federico Gonzales</h1>
                <h3>Desarrollador Web</h3>
                <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Impedit adipisci tempore
labore? Accusantium debitis ipsam, quae necessitatibus quaerat molestias non enim expedita commodi
quibusdam quam rerum numquam voluptate ipsum doloribus?</p>
            </div>
        </div>
    </div>
```

Directiva ngSwitchCase permite mostrar o no el componente dependiendo de la expresión.

```
    <div *ngSwitchCase="2" class="container">
        <div class="row">
            <div class="col">
                <img   src="assets/img/team/2.jpg" alt="">
            </div>
            <div class="col">
                <h1 class="mt-5 display-4 ">Diana López</h1>
                <h3>Desarrollador Web</h3>
                <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Impedit adipisci tempore
labore? Accusantium debitis ipsam, quae necessitatibus quaerat molestias non enim expedita commodi
quibusdam quam rerum numquam voluptate ipsum doloribus?</p>
            </div>
        </div>
    </div>
```

Para verlo funcionando en nuestra aplicación, ejecutar el comando ng serve (si no está corriendo el servidor) e ir a http://localhost:4200/quienes-somos/1 (el último valor es el que determinará qué integrante del equipo mostrar.

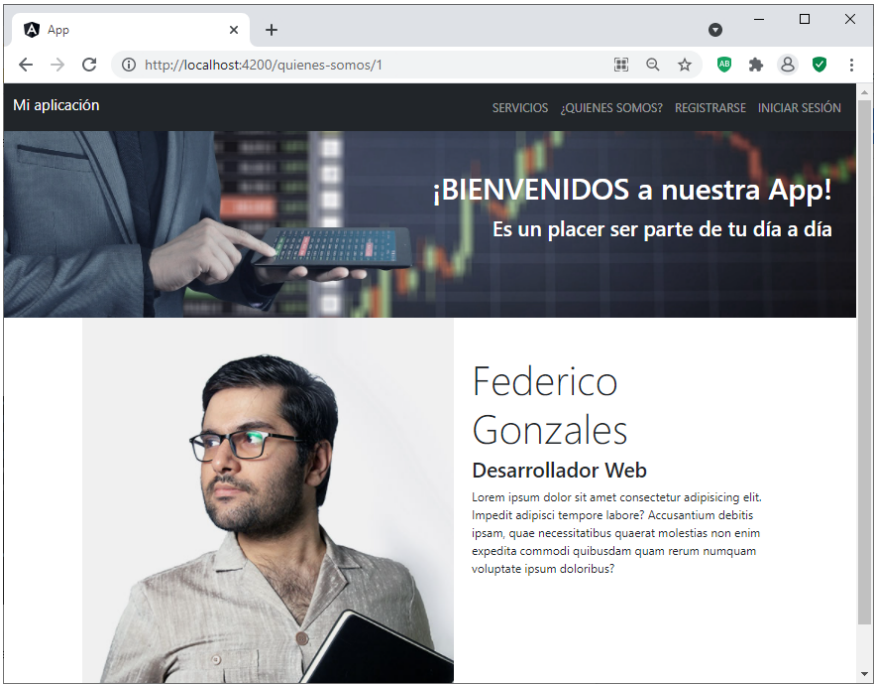


Figura: Uso del ngSwitch - Mostrar Perfil

Desafío. Modifica el ejemplo publicado en stackbliz (que te compartimos previemente) a fin de agregar la funcionalidad demostrada arriba.

3. Pipes

Pipes

Es el mecanismo que nos permite transformar los valores de entrada en valores de salida. Es decir, nos permite especificar el formato.

Los pipes se declaran una sola vez y pueden ser utilizados en toda la aplicación.

Angular provee pipes integrados para las transformaciones de datos, las siguientes son pipes integrados:

- **DatePipe**, cambia el valor de fecha.
- **UpperCasePipe**, transforma texto en Mayúscula.
- **LowerCasePipe**, transforma el texto en minúscula.
- **CurrencyPipe**, transforma un número en una cadena de moneda de acuerdo a ciertas reglas.
- **DecimalPipe**, transforma un número en una cadena con punto decimal.
- **PercentPipe**, Transforma un número en una cadena de porcentaje.

Nota: Puedes obtener la lista completa de pipes en https://angular.io/api/common#pipes

Ejemplo:

Continuando con nuestro ejemplo, vamos a modificar la tabla de movimientos a fin de que los montos se muestren en formato moneda. Además agregaremos la fecha con su formato en español arriba de la tabla.

Para ello, ejecutar los siguientes pasos:

1. En el archivo movimientos.component.ts crear una nueva variable para la fecha

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

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

Variable de fecha

```
  hoy= new Date();
  mostrarMovimientos: boolean=true;
  movimientos=[{operacion:"Extracción",monto:1500}, {operacion:"Depósito", monto:1520}];
  constructor() { }
  ngOnInit(): void {
  }
}
```

2. En el archivo movimientos.component.html agregar los pipes:

```
<h1 class="display-5">Últimos Movimientos</h1>

<h2>{{hoy|date: "d/M/yy"}}</h2>
<table *ngIf="mostrarMovimientos" class="table">
    <thead>
        <th>Operación</th>
        <th>Monto</th>
    </thead>
    <tbody>
        <tr *ngFor="let element of movimientos" >
            <td>{{element.operacion}}</td>

            <td>{{element.monto|currency}}</td>
        </tr>
    </tbody>
</table>
```

Pipe para la fecha, establece el formato día/mes/año

Pipe para los montos de dinero, establece el $ y dos decimales

3. Ejecutar el comando ng serve (si no está corriendo el servidor), y luego ir a http://localhost:4200/home/movimientos para ver el funcionamiento de los pipes.

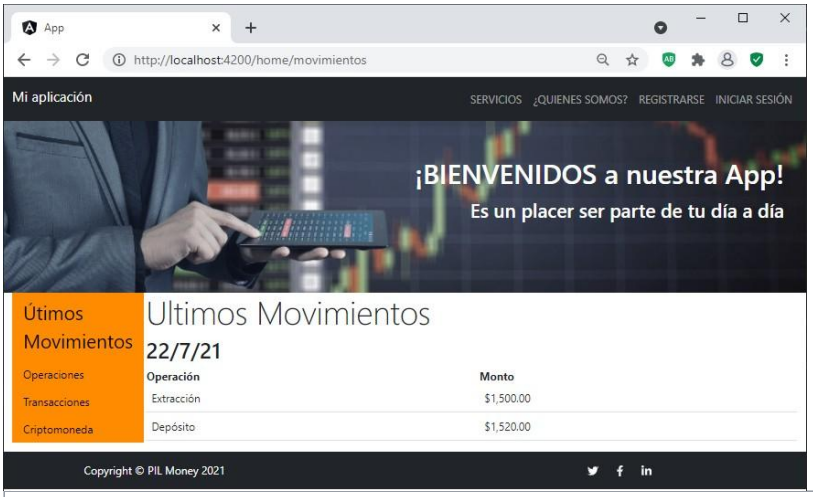


Figura: Uso de Pipes

Modifica el ejemplo publicado en stackbliz (que te compartimos previemente y vienes trabajando) a fin de agregar el componente de movimientos los pipes correspondientes al formato dinero.

4. Servicios

Servicios

Se puede llegar a presentar la situación de que dos o más componentes hagan uso de los mismos datos y para poder ser utilizados son referenciados en cada uno de los componentes. Pero a la hora de mantener la simplicidad Angular incorpora el concepto de Servicios permitiendo así que estos datos serán referenciados por los componentes desde los servicios ayudando a la simplicidad en el componente.

Los componentes delegan actividades a los servicios como ser: obtener datos necesarios, validar los mismos o registrar la información.Es decir que los servicios:

- Son proveedores de datos.
- Ayudan a mantener la lógica de acceso a los mismos.
- Proveen la operatoria del negocio.
- Manipulan de datos en la aplicación.
- Invocar a un servidor HTTP para consumir una API

Podemos pensar entonces que un "servicio es una categoría amplia que abarca cualquier valor, función o característica que necesite una aplicación. Un servicio es típicamente una clase con un propósito limitado y bien definido. Debe hacer algo específico y hacerlo bien.

Angular distingue los componentes de los servicios para aumentar la modularidad y la reutilización. Al separar la funcionalidad relacionada con la vista de un componente de otros tipos de procesamiento, puedes hacer que tus componentes sean ágiles y eficientes.

Idealmente, el trabajo de un componente es permitir la experiencia del usuario y nada más. Un componente debe presentar propiedades y métodos para el enlace de datos, para mediar entre la vista (representada por la plantilla) y la lógica de la aplicación (que a menudo incluye alguna noción de modelo)."(https://docs.angular.lat/guide/architecture-services)

Crear servicios

1. Ir a la consola DOS o "Símbolo del Sistema" del sistema operativo o Terminal de VSCode.

2. Ejecutar el comando: **ng generate service <<service-name>>**

o su abreviado: **ng g s <<service-name>>**

A continuación, AngularCLI generará un archivo <<nombre>>.servicio.ts el cual contendrá las siguiente líneas de código:

Anatomía de la clase de un servicio

Los servicios, como el resto de artefactos en Angular, son clases TypeScript decoradas con funciones específicas como podemos observar en el archivo <<nombre del servicio>>.services.ts:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class CuentaService {

  constructor() { }
}
```

En este caso el decorador @Injectable() define el servicio identificando la clase y su metadata que permita a Angular inyectarlo a un componente como dependencia.

¿Qué es Inyección de dependencias?

Los componentes consumen servicios; es decir, que podemos inyectar un servicio en un componente, dándole acceso al componente a ese servicio.

Figura 36: Inyección de dependencias
Fuente: https://docs.angular.lat/guide/architecture-services

De esta manera, el inyector crea dependencias y mantiene un contenedor de instancias de dependencia que reutilizará si es posible.

Es importante mencionar además que, se requiere de un proveedor, dado que éste le dice a un inyector cómo obtener o crear una dependencia.

Para cualquier dependencia que necesites en tu aplicación, debes registrar un proveedor con el inyector de la aplicación, con el fin de que el inyector pueda utilizar el proveedor para crear nuevas instancias. Para un servicio, el proveedor suele ser la propia clase de servicio." (https://docs.angular.lat/guide/architecture-services)

La Inyección de dependencias permite mantener las clases componentes ligeras y eficientes.

Los componentes no obtienen datos del servidor. Dicha tarea es delegada a los servicios. (https://docs.angular.lat/guide/architecture).

Es un patrón utilizado en la programación orientada a objetos que ayuda en la necesidad de creación de objetos de una forma práctica y con versatilidad en el código.

Los servicios deben tener idealmente un objetivo bien definido y un propósito.

Nota: Angular además permite la inyección de servicios de terceros a través de la inyección de dependencias.

¿Cómo funciona?

Cuando Angular crea una nueva instancia de un componente, determina qué servicios u otras dependencias necesita ese componente al observar los tipos de parámetros del constructor.

Si Angular descubre que un componente depende de un servicio, primero verifica si el inyector tiene instancias existentes de ese servicio. Si una instancia de servicio solicitada aún no existe, el inyector crea una utilizando el proveedor registrado y la agrega al inyector antes de devolver el servicio a Angular.

Cuando todos los servicios solicitados se han resuelto y devuelto, Angular puede llamar al constructor del componente con esos servicios como argumentos (https://docs.angular.lat/guide/architecture-services)

Finalmente el componente puede hacer uso del mismo.

Ejemplo:
Continuando con nuestro ejemplo y, a fines de comprender como crear y consumir servicios vamos a crear un servicio que nos provea de los últimos movimientos.

Nota: a fines demostrativos los datos estarán en duro en el servicio pero estos deberían ser obtenidos de una API que se conecte con la base de datos.

Para ello, ejecutar los siguientes pasos:

1. Ir a la consola DOS o "Símbolo del Sistema" del sistema operativo o Terminal de VSCode.
2. Ejecutar el comando: ng g s services/cuenta
   Nota: services es el nombre de la carpeta dónde deseo se cree el archivo.

   A continuación, AngularCLI crea un archivo cuenta.service.ts

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class CuentaService {

  constructor() { }
}
```

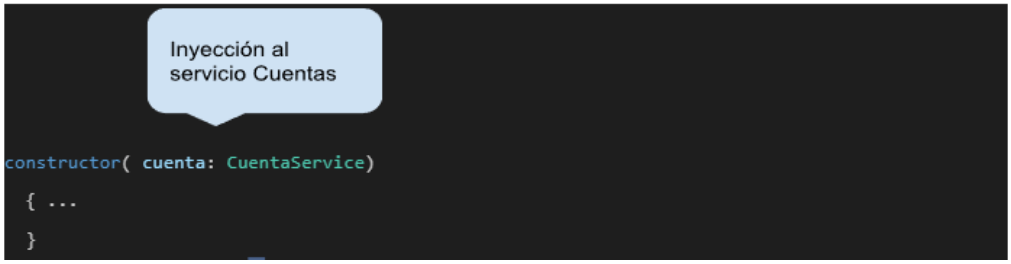
3. En el archivo cuenta.service.ts, crear un método que devuelva los últimos movimientos.

```
ObtenerUltimosMovimientos()
 {
  return [{operacion:"Extracción",monto:1500}, {operacion:"Depósito", monto:1520}];
 }
```

4. Importar el servicio cuentas

```
import { CuentaService } from 'src/app/services/cuenta.service';
```

5. Inyectar el servicio en el constructor del componente movimientos (archivo movimientos.component.ts)



6. Consumir el servicio.

```
  constructor( cuenta: CuentaService)
  {
     this.movimientos=cuenta.ObtenerUltimosMovimientos();
  }

  ngOnInit(): void {
  }
}
```

Ejecutar el comando ng serve (si no está corriendo el servidor), y luego ir a http://localhost:4200/home/movimientos para ver el formulario. Si bien no se observan cambios, ahora estamos consumiendo un servicio para traer los datos de los últimos movimientos.

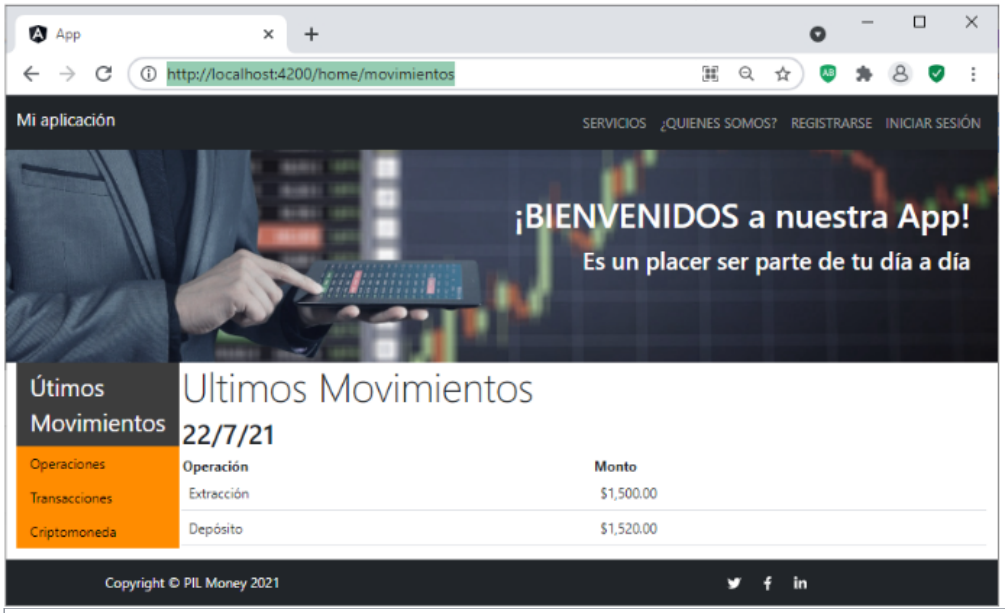


Figura: Datos obtenidos del servicio Cuenta

En el ejemplo planteado podemos observar que obtenemos los datos ahora desde un servicio y es igualmente de efectivo como si los datos estuvieran integrados en el archivo ".ts" del componente.

Usando un servicio en un template

Es importante agregar que, desde el HTML, también se puede acceder a servicios creados para mostrar propiedades o invocar sus métodos a través de los eventos. (siempre que se haya realizado la correspondiente inyección de dependencias).

```
<p>
  Nro. Cuenta: {{Cuenta.Nro}}
</p>
```

Modifica el ejemplo publicado en stackbliz (que te compartimos previemente y vienes trabajando) a fin de crear el servicio cuenta.service.

5. Referencias

https://docs.angular.lat/

https://desarrolloweb.com/articulos/introduccion-teorica-observables-angular.html

https://angular.io/

https://davidjguru.medium.com/single-page-application-un-viaje-a-las-spa-a-trav%C3%A9s-de-angular-y-javascript-337a2d18532

https://startbootstrap.com/theme/agency