Angular

Introducción, primeros pasos, components...

Agenda

- Introducción
- Primeros pasos
- Componentes

Introducción

Librería vs Framework

React es una librería, Angular es un Framework.

Librería.
Tu código usa la librería

Framework.
El Framework usa tu código

Características

Framework

Basado en HTML + JavaScript

Podemos utilizar ECMAScript, TypeScript o Dart.

SPA (Single Page Application)

Primeros pasos

Instalación del CLI

Previo

Nos hace falta tener instalado **nodejs**

https://nodejs.org/es/download/

npm viene ya preinstalado en las versiones recientes de **nodejs**.

Instalando

Desde el terminal, instalamos de manera global el cli de Angular:

```
npm install -g @angular/cli
```

¿ Funciona?

Vamos a ejecutar un comando y ver que funciona como esperamos.

```
ng version
```

Creación de un proyecto

Usamos el **CLI** desde la línea de comandos

```
ng new <nombre proyecto>
```

¿Nos creamos un proyecto que se llame *lemoncode-master-angular* ?

```
ng new lemoncode-master-angular
```

Este comando es interactivo y nos hace preguntas para configurar el proyecto:

- **¿Queremos soporte a routing?**
- **¿ Como vamos a manejar los estilos (CSS, LESS, SASS..)?**
- **(... a futuro podría ampliarse este wizard)**

Arranque del proyecto

Utilizamos el CLI para arrancar el proyecto

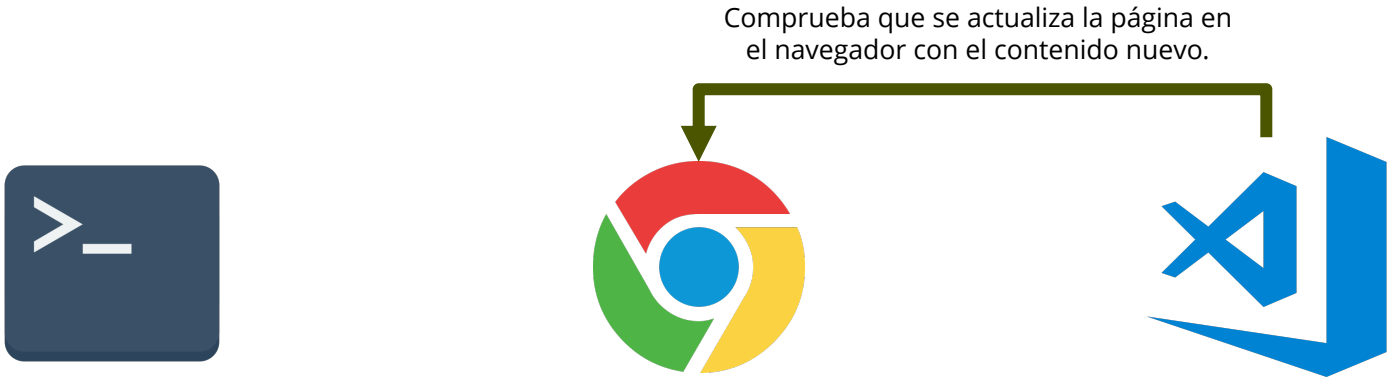
```
cd lemoncode-master-angular
```

```
ng serve
```

ng serve

“Hola Mundo”

Vamos a comprobar que nos funciona todo correctamente con un “Hola Mundo”. Para ello, sigue estos pasos:



Arranca el proyecto

```
ng serve
```

Introduce en el navegador la siguiente url:

http://localhost:4200

Modifica **src/app.component.html**

```
<h1>Hola Mundo</h1>
```

Componentes

Características

Es la pieza básica de Angular

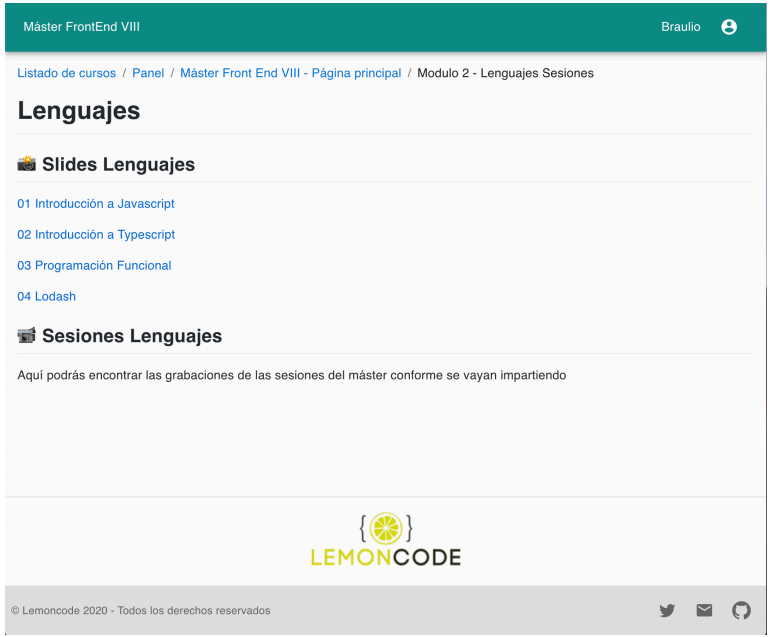
Se compone de html, css y js

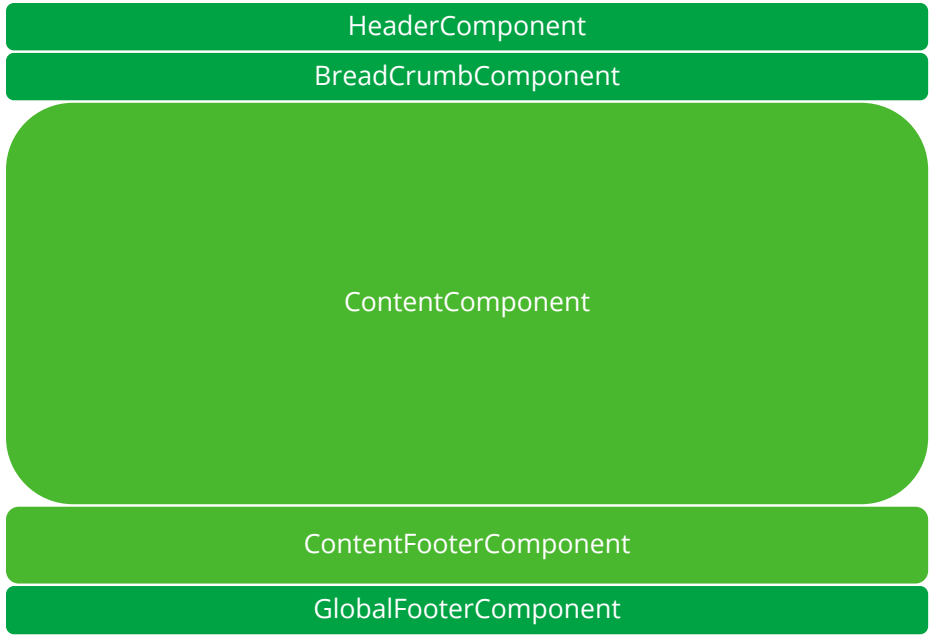
Controla un trozo de la pantalla llamado vista (view)

Es una clase configurada a través de la función decoradora @Component

Componentes

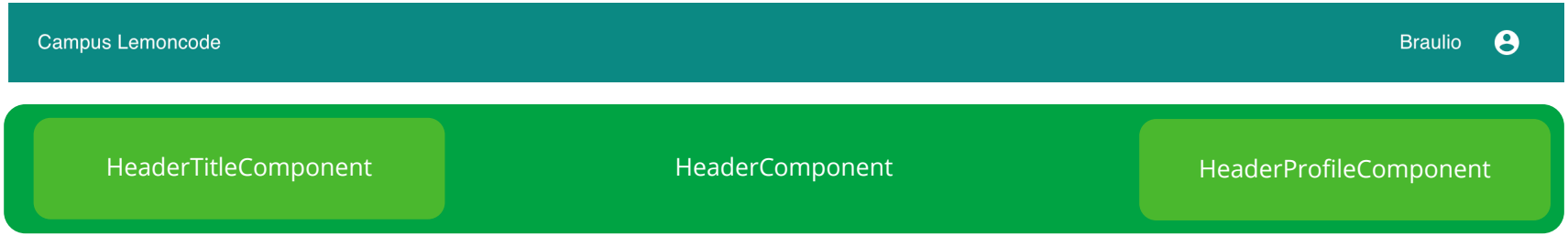
Controla un trozo de la pantalla llamado vista (view)





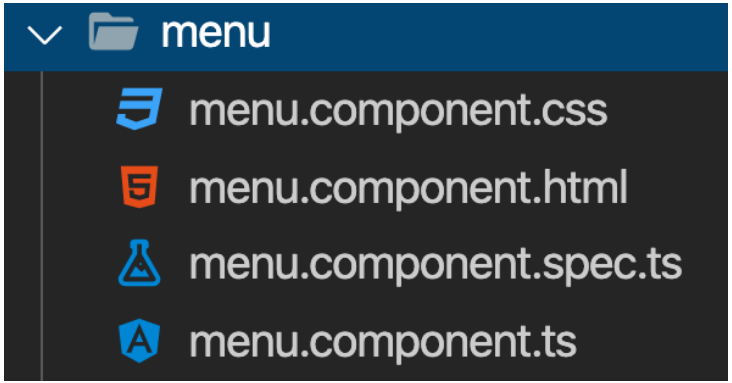
Componentes

Controla un trozo de la pantalla llamado vista (view)



Características

Se compone de html, css y js



[*.css] Estilos del componente

[*.html] Maquetación del componente

[*.spec.ts] Pruebas unitarias del componente

[*.ts] Lógica del componente

Características

Es una clase configurada a través de la función decoradora @Component

Las funciones **decoradoras** o decoradores como **@Component** reciben un argumento llamado objeto de *metadatos*.

Mediante los **metadatos** se configura el elemento de angular, en este caso un componente.

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

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

 constructor() {

 }

 ngOnInit(): void {
 }
}
```

informa a Angular de que la clase MenuComponent es un componente

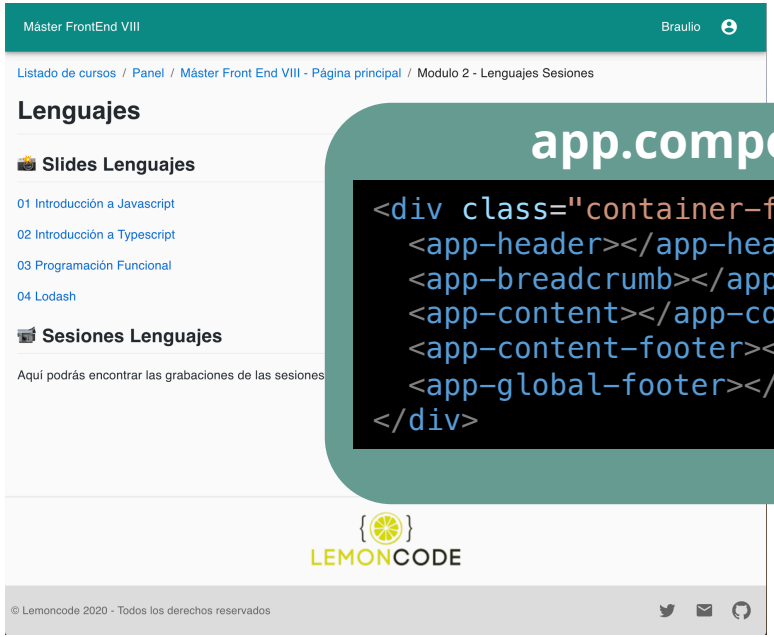
informa a Angular donde ubicar esa vista

informa a Angular cual es el HTML asociado

informa a Angular cuales son los css asociados

Uso - Componentización

Adios HTML's kilométricos



HeaderComponent [app-header]

BreadCrumbComponent [app-breadcrumb]

app.component.html

```
<div class="container-fluid">
  <app-header></app-header>
  <app-breadcrumb></app-breadcrumb>
  <app-content></app-content>
  <app-content-footer></app-content-footer>
  <app-global-footer></app-global-footer>
</div>
```

nent [app-content]

ContentFooterComponent [app-content-footer]

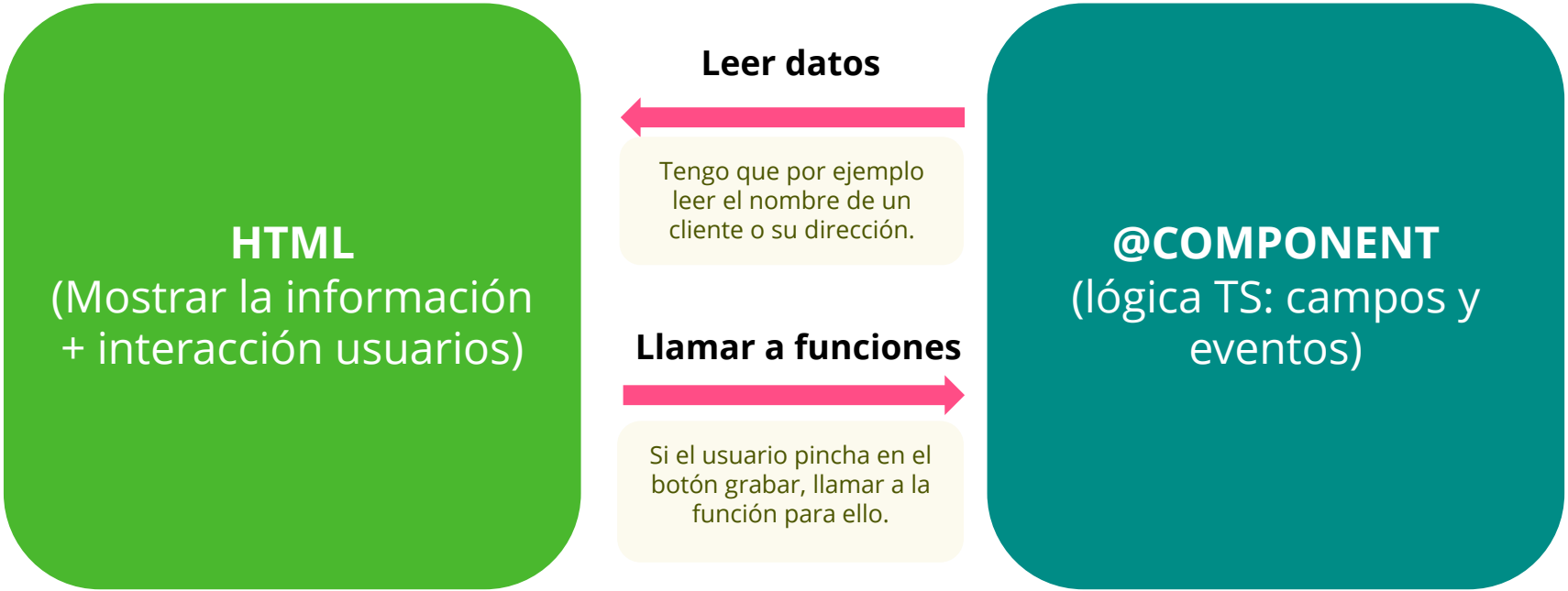
GlobalFooterComponent [app-global-footer]

Sintaxis de plantilla

Data Binding

Presentación y datos

Una clase asociada a un componente (**@component**) necesita estar enlazada con su plantilla (**html**) a través de lo que se denomina *data-binding*.



Tipos de binding

One way data binding

Sólo está activo en una dirección

Me permite leer datos de un componente pero no actualizarlos

Me permite invocar a funciones de un componente desde el HTML

Determinista: tengo control de cuándo se actualiza qué

Two way data binding

Activo en los dos sentidos

Me permite leer datos de un campo en el componte y que cuando cambie en el UI automáticamente se actualice en el campo

Esto puede llegar a ser peligroso, pierdo control de cuándo se actualiza qué

One way data binding

{{}} - Interpolación

```
<h1>{{ apellido }}</h1>
```

Muestra por pantalla los datos del componente que le indiquemos

Podemos pasarle un campo

Podemos pasarle una expresión

Podemos pasarle una invocación a una función

[] – Atributos, propiedades, clases, estilos

```
<img [src]="urlFoto"/>
```

Puedo modificar atributos, propiedades y clases de un elemento HTML

Podemos pasarle un campo

Podemos pasarle una expresión

Podemos pasarle una invocación a una función

() - Eventos

```
<button (click)="grabar()"/>
```

Permite ejecutar una función del componente cuando se da un evento

Podemos montar más de una línea de código si queremos

Podemos recoger el contexto del evento

La variable que tiene la información de contexto del evento se llama $event

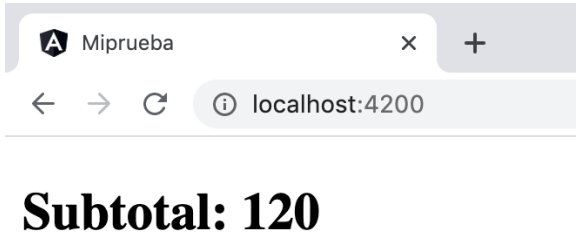
{{}} – Interpolación - ejemplos

Mostrando el valor de una propiedad de nuestra clase asociada al componente.

app.component.html

```
<h2>Subtotal: {{precio}}</h2>
```

Resultado



app.component.ts

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

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

export class AppComponent {
  precio = 120;
  iva = 42;
}
```

https://codesandbox.io/s/clever-snowflake-ebi3q

{{}} – Interpolación - ejemplos

Mostrando el valor de una expresión en base valores de nuestra clase asociad al componente.

app.component.html

```
<h2>Subtotal: {{precio}}</h2>
<h1>Total: {{precio + iva}}</h1>
```

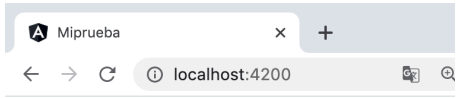
app.component.ts

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

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

export class AppComponent {
  precio = 120;
  iva = 42;
}
```

Resultado



Subtotal: 120

Total: 162

https://codesandbox.io/s/nifty-carson-pg9fx

[] – Atributos - ejemplos

Modificando un atributo de un elemento del DOM en base valores de nuestra clase asociada al componente.

app.component.html

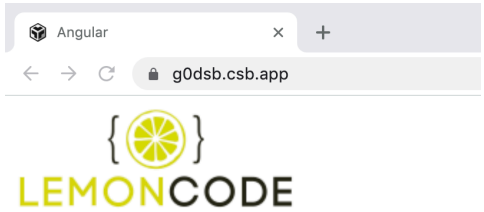
```
<img width="240px" [attr.src]="urlLemoncode" />
```

app.component.ts

```
import { Component } from "@angular/core";

@Component({
selector: "app-root",
templateUrl: "./app.component.html",
styleUrls: ["./app.component.css"]
})
export class AppComponent {
  urlLemoncode = "https://static1.squarespace.com/1592";
  urlBasefactor = "https://www.basefactor.com//footer.png";
}
```

Resultado



https://codesandbox.io/s/crimson-snowflake-g0dsb

[] – Propiedades - ejemplos

Al igual que con los atributos, podemos asignar directamente propiedades (mejor eficiencia que con atributos).

app.component.html

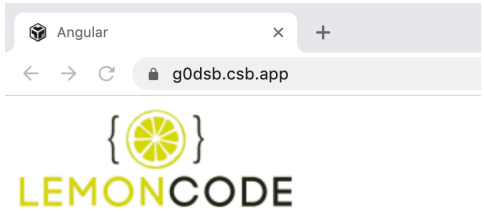
```
<img width="240px" [src]="urlLemoncode" />
```

app.component.ts

```
import { Component } from "@angular/core";

@Component({
selector: "app-root",
templateUrl: "./app.component.html",
styleUrls: ["./app.component.css"]
})
export class AppComponent {
  urlLemoncode = "https://static1.squarespace.com/1592";
  urlBasefactor = "https://www.basefactor.com//footer.png";
}
```

Resultado



https://codesandbox.io/s/unruffled-browser-uir7q

[ ] – Clases - ejemplos

Nos permite aplicar clases CSS dependiendo de campos y expresiones que hayan en nuestra clase de componente.

app.component.html

```
<div width="200px" height="200px"
   [class.fondo-azul]="esAzul">
Mira el color de fondo
</div>
```

Resultado

app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  esAzul = true;
}
```

app.component.css

```
.fondo-azul {
background: blue;
}
```

https://codesandbox.io/s/cool-mccarthy-cychr

[ ] – Estilos - ejemplos

Al igual que con las clases, podemos aplicar estilos.

app.component.html

```
<div width="200px" height="200px"
   [style.background]="esAzul ?  'blue': 'white'">
Mira el color de fondo
</div>
```

app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  esAzul = true;
}
```

Resultado

https://codesandbox.io/s/cold-darkness-qon41

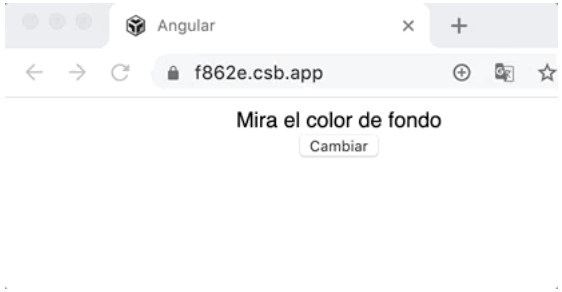
( ) – Eventos - ejemplos

Reacciona a eventos del DOM invocando a funciones del componente clase.

app.component.html

```
<div>
  <div width="200px" height="200px" [class.fondo-azul]="esAzul">
    Mira el color de fondo
  </div>
  <button (click)="cambia()">Cambiar</button>
</div>
```

Resultado



app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  esAzul = false;

  cambia() {
    this.esAzul = ! this.esAzul;
  }
}
```

app.component.css

```
.fondo-azul {
background: blue;
}
```

https://codesandbox.io/s/sweet-williams

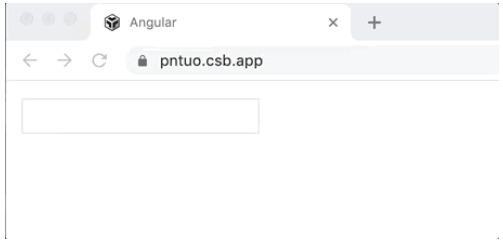
Two way data binding - ngModel

Permite mostrar un dato de la clase componente en un elemento, y automáticamente actualizarlo si dicho componente lo modifica.

app.component.html

```
<div>
<input [(ngModel)]="nombre"/>
{{nombre}}
</div>
```

Resultado



https://codesandbox.io/s/tender-euler-pntuo

app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  nombre = '';
}
```

app.module.ts

```
@NgModule({
declarations: [AppComponent],
imports: [BrowserModule, FormsModule],
```

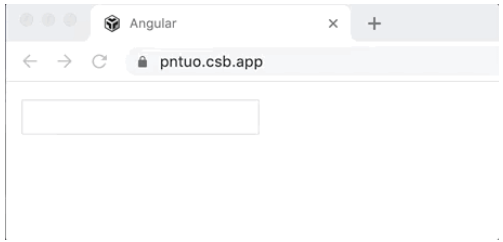
Sin Two Way binding

Podemos tener un método para actualizar o directamente en el DOM

app.component.html

```
<div>
<input [value]="nombre"
(keyup)="actualizaNombre($event.target.value)"/>
{{nombre}}
</div>
```

Resultado



app.component.ts

```
import { Component } from "@angular/core";

@Component({
selector: "app-root",
templateUrl: "./app.component.html",
styleUrls: ["./app.component.css"]
})
export class AppComponent {
  nombre = '';

  actualizaNombre(value) {
    this.nombre = value;
  }
}
```

https://codesandbox.io/s/stupefied-proskuriakova-ien6v

Sintaxis de plantilla

Directivas estructurales

Lógica UI

En Angular se extiende el HTML para soportar binding... Pero hay más casuística...

¿ Qué pasa si quiero mostrar u ocultar una part del DOM en base a una condición?

¿Y si quiero iterar por una lista de resultados?

Para eso tenemos las directivas estructurales (+esteroides al HTML)

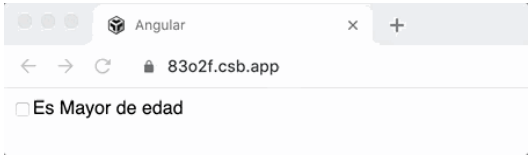
*ngIf

Añade o elimina del DOM un elemento HTML o un componente Angular basándose en una condición.

app.component.html

```
<div>
  <div>
    <input type="checkbox"
       [checked]="esMayor"
      (change)="cambiaEsMayor()"
    />
    <label>Es Mayor de edad</label>
  </div>
  <div *ngIf="esMayor">
    <label>NIF</label>
    <input [value]="nif"/>
  </div>
</div>
```

Resultado



app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  esMayor = false;
  nif = "12345678X";

  cambiaEsMayor() {
    this.esMayor = !this.esMayor;
  }
}
```

https://codesandbox.io/s/amazing-euclid-83o2f

*ngFor

Nos permite iterar por una lista y aplicarle un HTML dado a cada elemento.

app.component.html

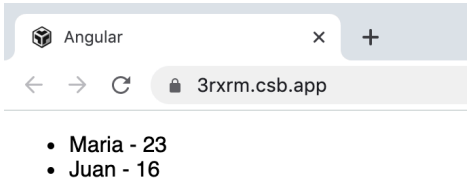
```
<div>
  <ul>
    <li *ngFor="let persona of personas">
      {{persona.nombre}} - {{persona.edad}}
    </li>
  </ul>
</div>
```

app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  personas = [
    {nombre: 'Maria', edad: 23},
    {nombre: 'Juan', edad: 16}
  ];
}
```

Resultado



https://codesandbox.io/s/condescending-feather-3rxrm

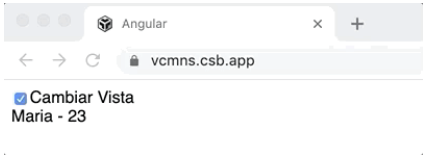
*ngSwitch

Análogo al *switch / case* de **ES6**, permite cambiar vistas.

app.component.html

```
<div>
  <input type="checkbox" [checked]="vista"
(change)="cambiaVista()" />
  <label>Cambiar Vista</label>
  <div [ngSwitch]="vista">
    <div *ngSwitchCase="'total'">
      {{persona.nombre}} - {{persona.edad}}
    </div>
    <div *ngSwitchCase="'edad'">
      {{persona.edad}}
    </div>
  </div>
</div>
```

Resultado



app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  vista = "total";
  persona = { nombre: "Maria", edad: 23 };

  cambiaVista() {
    this.vista = this.vista === "total" ? "edad" : "total";
  }
}
```

https://codesandbox.io/s/cranky-rain-q9t7h

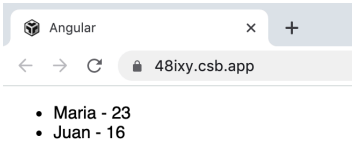
ng-container

No se pueden combinar dos

app.component.html

```
<div>
  <ul>
    <ng-container *ngFor="let persona of personas">
      <li *ngIf="persona.nombre">
        {{persona.nombre}} - {{persona.edad}}
      </li>
    </ng-container>
  </ul>
</div>
```

Resultado



app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  personas = [
    { nombre: "Maria", edad: 23 },
    { nombre: "", edad: 31 },
    { nombre: "Juan", edad: 16 }
  ];
}
```

https://codesandbox.io/s/dank-resonance-48ixy

¡Muchas gracias!

@lemoncoders

@basefactorteam

https://github.com/lemoncode