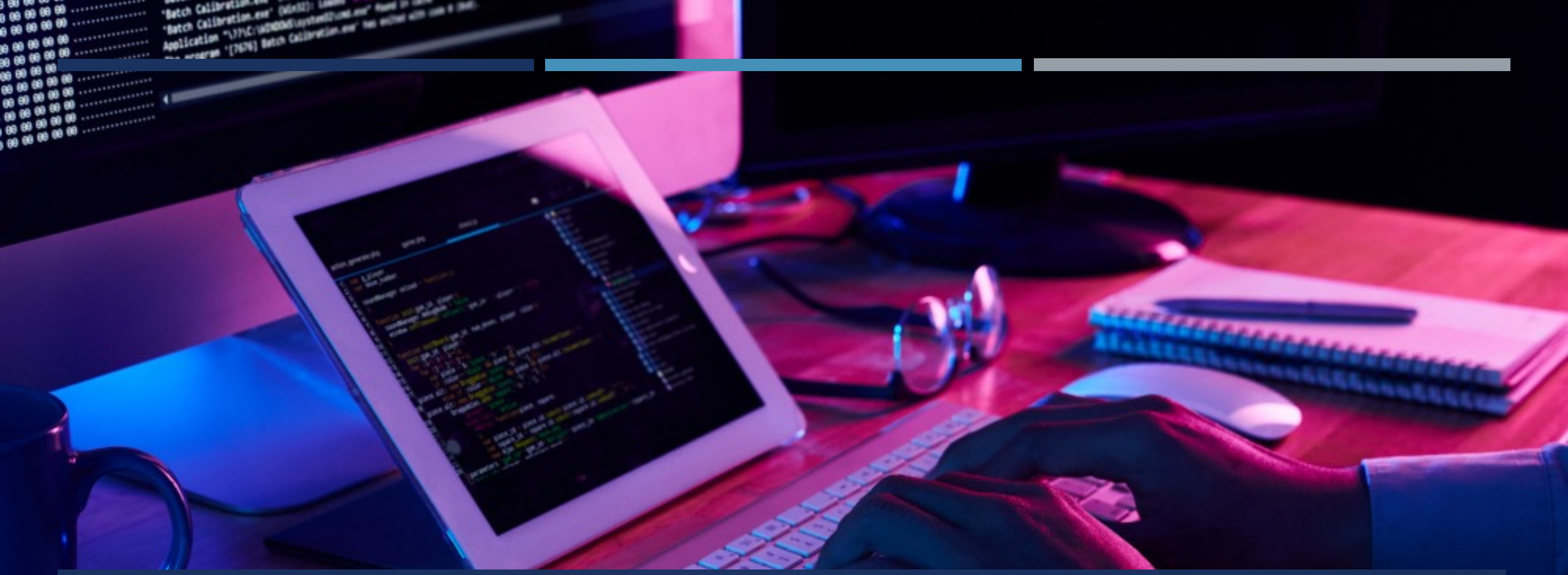

PROGRAMACIÓN WEB EN EL ENTORNO SERVIDOR

MF0492_3

Gobierno de Canarias
Servicio Canario de Empleo

FUERTEVENTURA 2000
Centro de Formación Profesional

PIEC
PLAN INTEGRAL DE EMPLEO DE CANARIAS

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL
SEPE

Realizado por Manuel Macías

¿Qué es Angular?

Angular es un framework Javascript para la creación de aplicaciones web SPA (**Single Page Application**).
Que una web sea SPA quiere decir que cuando el usuario entra en la web se carga todo el contenido de todas las páginas a la vez. Esto quiere decir que la primera carga nada más entrar es más lenta pero luego los cambios entre páginas son instantáneos.
Otra de las características de Angular es que utiliza Typescript , un lenguaje que luego se compila a Javascript y que sirve para tener tipado de variables, interfaces, inyección de dependencias, etc..

¿Qué es Angular?

Ventajas de usar Angular

- **Lenguaje Typescript**, tiene una sintaxis muy parecida a Java, con tipado estático. Esto te va a ayudar a que tu código tenga menos errores.
- **Sigue el patrón MVC** (Modelo - Vista - Controlador), con la vista separada de la lógica de negocio.
- **Basado en componentes**, es decir, piezas de código con vista que puedes reutilizar en otras páginas.
- **Comunidad muy grande** con multitud de tutoriales y librerías.
- **Inyección de dependencias**, un patrón de diseño que se basa en pasar las dependencias directamente a los objetos en lugar de crearlas localmente.
- **Programación reactiva**, la vista se actualiza automáticamente a los cambios en las variables.

Ventajas de usar Angular

- **Dispone de asistente por línea de comandos** para crear proyectos base (Angular cli).
- **Se integra bien con herramientas de testing**.
- **Buena integración con Ionic**, para adaptar aplicaciones web a dispositivos móviles.
- **Framework completo**. No necesitas instalar demasiadas librerías externas para hacer cosas comunes.
- **Apoyado por Google**. Esto consigue que por lo general tenga mejor soporte y documentación

Ventajas de usar Angular

- **Dispone de asistente por línea de comandos** para crear proyectos base (Angular cli).
- **Se integra bien con herramientas de testing**.
- **Buena integración con Ionic**, para adaptar aplicaciones web a dispositivos móviles.
- **Framework completo**. No necesitas instalar demasiadas librerías externas para hacer cosas comunes.
- **Apoyado por Google**. Esto consigue que por lo general tenga mejor soporte y documentación

Cómo es la estructura de un proyecto Angular

- 📁 **node_modules**: En esta carpeta se encuentran las librerías de angular y sus dependencias, cuando instalemos librerías se añadirán aquí. Generalmente no hay que tocar nada de esta carpeta.
- 📁 **src**: Aquí se encuentran los archivos que componen nuestra aplicación
  - 📁 **app**: Aquí se donde se van a encontrar los componentes, vistas, y servicios de la app. Por el momento hay un componente llamado app con sus respectivos archivos (css, html controlador, tests, etc)
    - 📄 **app.module.ts**: En este archivo se especifica los componentes que vamos a usar en la app web. Cuando creemos un componente tenemos que importarlo en este archivo.
  - 📄 **favicon**: El favicon de la web (icono que aparece en el navegador al lado del nombre de la pestaña)
  - 📄 **index.html**: Punto de entrada a nuestra web, este archivo se carga en todas las webs, por lo que puedes poner código para que se incluya en todas las vistas.
  - 📄 **main.ts**: Algunas configuraciones de Angular, de momento no nos hace falta tocarlo.
  - 📄 **polyfills.ts**: Configuraciones y código que se ejecutará antes de que se inicie la app. De momento tampoco nos hace falta tocarlo.

Cómo es la estructura de un proyecto Angular

- **styles.css**: Estilos css globales que se aplicarán en toda las vistas de la página.
- **test.ts**: Configuración para los tests. No es útil de momento.
- **tsconfig.app.json, tsconfig.spec.json y typings.d.ts**: Lo mismo que el anterior.

- **.angular-cli.json**: Archivo de configuración de la app.
- **.editorconfig**: Configuraciones a la hora de desarrollar, por ejemplo, como se identa el código.
- **.gitignore**: Archivo para que git ignore ciertas carpetas que no hace falta subir, como node_modules (cuando te bajas el proyecto ejecutas npm install para que descargue las dependencias en node_modules).
- **karma.conf.js**: Más configuraciones para los tests, esta vez los de Karma.
- **package-lock.json**: Árbol de dependencias que se crea automáticamente
- **package.json**: Archivo con las dependencias instaladas y los comandos que se pueden ejecutar con npm
- **protractor.conf.js**: Configuración para protractor, una herramienta para realizar tests en el navegador.
- **README.md** Archivo readme con información de la aplicación.
- **tsconfig.json**: Configuración para Typescript, el lenguaje de Angular.
- **tslint.json**: Configración del linter de TypeScript (un linter sirve para hacer comprobaciones del estilo del código que escribimos).

Cómo es la estructura de un proyecto Angular

- **styles.css**: Estilos css globales que se aplicarán en toda las vistas de la página.
- **test.ts**: Configuración para los tests. No es útil de momento.
- **tsconfig.app.json, tsconfig.spec.json y typings.d.ts**: Lo mismo que el anterior.

- **.angular-cli.json**: Archivo de configuración de la app.
- **.editorconfig**: Configuraciones a la hora de desarrollar, por ejemplo, como se identa el código.
- **.gitignore**: Archivo para que git ignore ciertas carpetas que no hace falta subir, como node_modules (cuando te bajas el proyecto ejecutas npm install para que descargue las dependencias en node_modules).
- **karma.conf.js**: Más configuraciones para los tests, esta vez los de Karma.
- **package-lock.json**: Árbol de dependencias que se crea automáticamente
- **package.json**: Archivo con las dependencias instaladas y los comandos que se pueden ejecutar con npm
- **protractor.conf.js**: Configuración para protractor, una herramienta para realizar tests en el navegador.
- **README.md** Archivo readme con información de la aplicación.
- **tsconfig.json**: Configuración para Typescript, el lenguaje de Angular.
- **tslint.json**: Configración del linter de TypeScript (un linter sirve para hacer comprobaciones del estilo del código que escribimos).

¿Qué es un componente?

Un componente es una parte de la web que tiene vista, estilos y lógica. Las etiquetas HTML en cierto modo no dejan de ser componentes, por ejemplo, la etiqueta <input> es un componente porque, de forma predeterminada, se pinta con unos estilos y con la lógica necesaria para que el usuario pueda hacer clic y escribir en ellos.

En otras palabras, con los componentes web puedes crear tus propias etiquetas HTML que podrás reusar en toda la aplicación web.

¿Qué es un componente?

Los componentes de Angular van más allá y pueden incluso recibir objetos javascript completos y arrays para su configuración. Por ejemplo puedes crear un componente para pintar una lista de usuarios. Este componente recibirá como entrada un array de usuarios a pintar. De esta forma consigues en puedas reutilizar esta lista en otra parte de la web, por lo que es fundamental que los hagas lo más genéricos posibles para que se puedan reusar.

¿Qué es un componente?

Método automático
Si estamos usando Angular CLI, en nuestro proyecto, existe un comando muy útil para generar la estructura básica de un componente.

Por ejemplo, imaginemos que queremos crear un componente que sirva para crear el menú principal de una web y que lo queremos llamar navbar por el momento. Tan solo tienes que ejecutar en la terminal:

ng generate component navbar

¿Qué es un componente?

La última palabra es el nombre del componente. Si es un componente de varias palabras se recomienda que le des un nombre en kebab-case, es decir, separada cada palabra por un guión, por ejemplo:

user-list.

¿Qué es un componente?

Veamos ahora el archivo .ts también llamado controlador que es el que define la lógica del componente. La estructura básica ya la has visto, pero por si acaso es esta:

```
// app/navbar/navbar.component.ts

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

@Component({
  selector: "app-navbar",
  templateUrl: "./navbar.component.html",
  styleUrls: ["./navbar.component.css"],
})
export class NavbarComponent {
  constructor() {}
}
```

¿Qué es un componente?

Lo primero que se hace es importar Component de @angular/core. Esto sirve para poder invocar justo debajo @Component, una especie de método que recibe un objeto con la configuración del componente. Dentro de esa configuración hay tres propiedades:

selector: El selector es el nombre que va a tener la etiqueta HTML que sirve para poder usar este componente, para este ejemplo del navbar será <app-navbar></app-navbar>, es decir, desde el HTML de cualquier otro componente poniendo esa etiqueta se pintará el navbar.

¿Qué es un componente?

Angular tiene una convención de nombre para el selector, kebab-case (el nombre de los selectores tiene que ser una palabra seguida de un guión y otra palabra: app-ejemplo).

templateUrl: La ruta al fichero .html de ese componente para crear la vista.

styleUrl: La ruta al fichero .css de ese componente para crear los estilos.

¿Qué es un componente?

Por último hacemos export class con el nombre del componente (sin guiones aquí). Dentro del export se crea el método constructor (vacío por el momento).

¡OJO!. El nombre que usas cuando exportas el componente (NavbarComponent en el ejemplo de arriba) es el que luego tienes que poner entre llaves para importar el componente dentro del app.module, en este caso:

```
import { NavbarComponent } from "./navbar/navbar.component";
```

¿Qué es un componente?

Cómo usar los componentes dentro de otros

Si ya has creado el componente (puedes añadir alguna etiqueta HTML en el .html) y si ya lo tienes importado correctamente en el app.module.ts, puedes probar que el componente funciona correctamente simplemente creando etiqueta dentro del archivo app.component.html.

```
<!-- app.component.html -->
<div>
<app-navbar></app-navbar>
 </div>
```

OnInit

Angular, como otros frameworks Javascript tiene una forma de poder ejecutar código cuando el componente carga por primera vez, en Angular es un método que se llama **ngOnInit.**

Es importante saber que cuando el componente se destruye en una vista y se vuelve a cargar también se vuelve a lanzar el ngOnInit.

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

OnInit

Diferencias entre ngOnInit() y el constructor() en Angular

Como otros lenguajes, typescript también tiene un constructor de clase, en este caso el constructor se ejecuta antes que el ngOnInit().

Normalmente se usa el constructor para inicializar variables, y el ngOnInit para inicializar o ejecutar tareas que tienen que ver con Angular. Todo esto lo podemos poner directamente en el constructor y funcionaría de la misma manera, pero no está de más tener más separado el código para que sea más mantenible.

Sistemas de Vistas o Visualización

La manera más rápida de mostrar una variable definida en un controlador (fichero .component.ts) es usar la sintaxis:

** {{ exampleString }} **

Lo bueno de esta sintaxis es que dentro de las dos llaves el código se ejecutará como si fuera de Javascript, es decir, puedes hacer:

La suma de 5 y 4 es {{ 5 + 4 }}

Sistemas de Vistas o Visualización

O concatenar strings:

** {{ exampleString1 + "-" + exampleString2 }} **

y ponemos un input junto a un botón para mostrarlas: O incluso fechas:

** {{ new Date() }} **

Sistemas de Vistas o Visualización

ngFor para mostrar listas de elementos

¿Qué pasa si tenemos un array de información?, para este caso Angular viene con etiquetas especiales para recorrer arrays o listas:

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

Sistemas de Vistas o Visualización

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

Es importante poner el * antes del ngFor para que Angular lo detecte adecuadamente.
Como ves, dentro del **ngFor** usamos sintaxis ES6, en concreto hacemos un **foreach**. Es decir, dentro del **ngFor** se crea la variable **item** que en cada iteracción del bucle actualizará su valor por el valor en ese momento.
Cuando Angular sirva la página se encargara de crear un elemento <li> por cada elemento del array, y dentro de cada uno, mostrará su contenido. Si en lugar de hacer el ngFor dentro un <li> lo haces dentro de un , por ejemplo, se creará un div por cada elemento del array.

Sistemas de Vistas o Visualización

ngIf para mostrar elementos de forma condicional

En ocasiones, necesitamos que un elemento se muestre en pantalla dependiendo de una condición. Esto se consigue por medio de ngIf:

```
<div *ngIf=”array.length < 3">
Esto se muestra si el array tiene menos de 3 elementos</div>
```

Sistemas de Vistas o Visualización

ngIf para mostrar elementos de forma condicional

Si por ejemplo, tenemos una variable boolean (true, o false) definida en el controlador, también podemos mostrar contenido o no dependiendo del valor de esa variable:

```
<div *ngIf="condicion">
 Esto se muestra si la variable condicion es true (o distinto de null
y undefined)
</div>
```

Sistemas de Vistas o Visualización

ngIf para mostrar elementos de forma condicional

Dentro de los if también podemos llamar a funciones dentro de su controlador, por ejemplo:

```
<div *ngIf="getVal() > 3">Ejemplo</div>
```

En este caso el string "Ejemplo" solo se mostrará si el valor de retorno de la función getVal definida en el controlador es mayor a 3;

Sistemas de Vistas o Visualización

Estilos y atributos del HTML dinámicos

Puedes hacer bind, es decir, hacer que un atributo del HTML tenga el valor de una variable definida en el controlador.

Por ejemplo:

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

En este caso el valor src de la imagen tendrá el valor que tenga esa variable en ese momento.

Sistemas de Vistas o Visualización

Estilos y atributos del HTML dinámicos

Si queremos, por ejemplo, cambiar el color de fondo de una etiqueta html, pero poniendo el color que tenemos guardado en una variable, se hace así:

```
<div class="circle" [style.background]="color"></div>
```

Sistemas de Vistas o Visualización

Binding de click y para los inputs

Por ejemplo para controlar cuándo el usuario hace click en un botón:

```
<button (click)="onClick()">Click me!</button>
```

Es decir, este tipo de eventos va entre paréntesis. Después en el controlador, tenemos que crear un método con el mismo nombre, en este caso, onClick, que será la que se ejecute al pulsar el botón.

```
// button-example.component.ts
export class ExampleComponent implements OnInit {
exampleString: string;
exampleArray: string;
constructor() { this.exampleArray = [];
}
ngOnInit() {
}
onClick(){
console.log("Botón pulsado").
}
}
```

Sistemas de Vistas o Visualización

Recoger el valor en los inputs

También puedes almacenar el valor que escribe el usuario en los recuadros input de HTML.

Para recoger el valor del input mientras escribe el usuario, antes tenemos que asegurarnos que hemos importado el módulo de FormsModule de Angular, en el archivo app.module.ts, es decir:

Sistemas de Vistas o Visualización

Recoger el valor en los inputs

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";
import { FormsModule } from "@angular/forms";
import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
@NgModule({
declarations: [AppComponent],
imports: [BrowserModule, AppRoutingModule, FormsModule],
providers: [],
bootstrap: [AppComponent],
})
export class AppModule {}
```

Sistemas de Vistas o Visualización

Recoger el valor en los inputs

Una vez hecho esto podemos continuar creando los inputs. Para bindear el input con la variable:

```
<input [(ngModel)]="name" />
```

El ngModel con esa sintaxis lo tienes que poner siempre, pero name en este ejemplo es una variable y puedes poner el nombre que quieras siempre y cuando tengas una variable que se llame igual.

Sistemas de Vistas o Visualización

Recoger el valor en los inputs

Al usar la sintaxis [(...)] lo que estamos haciendo es **two way data binding**. El paso de información se hace en dos sentidos: entre el html y el controlador y al revés.

Esto quiere decir que si actualizamos el valor en el html o en el controlador en el otro sentido se actualizará automáticamente.

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

Si ahora muestras la variable name, debajo del input, te darás cuenta de que según escribes en el formulario se va mostrando abajo automáticamente:

Cómo crear servicios

Una pieza fundamental en la mayoría de webs y aplicaciones es la de la capa de los datos. El acceso a los datos lo puedes hacer perfectamente desde los propios componentes, pero, lo recomendable es que esta capa de acceso a los datos esté separada.

Lo que se hace en Angular es crear un fichero separado para esto llamado servicio. Los servicios son clases que se encargan de acceder a los datos para entregarlos a los componentes. Lo bueno de esto es que puedes reaprovechar servicios para distintos componentes.

Cómo crear servicios

Para crear un servicio, podemos crear manualmente un archivo llamado nombre-componente.service.ts o podemos dejar que angular cli lo cree por nosotros:

```
ng generate service nombre-componente --module=app
```

El atributo --module=app indica que el servicio que se va a crear se va a importar directamente en el app.module.ts. Si lo creas de forma manual o no pones este parámetro tienes que importar tu servicio en el app.module.ts:

Cómo crear servicios

Un servicio tiene la siguiente estructura:

Lo primero es importar Injectable para poder usar esa etiqueta de Angular y así poder inyectar el servicio. En ese caso, dentro del inyectable se ha puesto providedIn: 'root' para que el servicio se auto-importe y no tengas que importarlo en el app.module.ts.

Debajo se pone el export con el nombre del servicio, normalmente es NombreComponentService. Bien, tienes que pensar que se va a usar esta clase para acceder a los datos. Para este ejemplo se ha creado un array de mensajes inicializado a array vacío, justo debajo del export. Ese array serán nuestros datos.

Por último se sitúan los métodos de añadir mensajes y borrar todos los mensajes (aquí tienes que poner los que necesites para acceder o modificar tus datos).

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

@Injectable({
providedIn: "root",
})
export class MessageService {
messages: string[] = [];

add(message: string) {
this.messages.push(message);
  }

clear() {
this.messages = [];
  }
}
```

Cómo crear servicios

Cómo usar los servicios dentro de los componentes

La anotación @Injectable indica que el servicio puede ser inyectado mediante Inyección de dependencias en Angular, es decir, primero tenemos que importar el servicio en el componente:

```
import { ServiceName } from "../serviceName.service";
```

A continuación lo inyectamos en el constructor del componente:

```
constructor(private service: ServiceName) { }
```

Ya podemos usar los métodos del servicio en el componente service.nombreFuncion().

Cómo usar las pipes para dar formato en la vista

Las pipes son filtros o funciones que pones directamente en la vista para el dar formato a un dato que estés pintando.

Por ejemplo, imagina que quieres poner una fecha, ‘Fri Apr 15 1988 00:00:00 GMT-0700’. Si quieres ponerla en formato MM/DD/YYYY, con lo que sabes hasta ahora, vas a tener que crear una función en el componente que se ejecuta cuando se pone la vista. Pues con las pipes vas a poder crear funciones que hacen esto automáticamente y que puedes reutilizar a lo largo de tu aplicación.

La ventaja de las pipes es que te vas a ahorrar mucho código en los componentes, lo que facilitará su lectura y mantenimiento.

Cómo usar las pipes para dar formato en la vista

Cómo usar las pipes

Los pipes se usan poniendo el carácter | al poner una variable, por ejemplo podemos poner:

```
{{ 'BBBBB' | lowercase }}
```

Y lo que imprimirá será: bbbbb

También se pueden combinar varias pipes, por ejemplo:

```
{{ 459.67 | currency: 'USD' | lowercase }}
```

Lo que imprimirá: $459.67 (se han aplicado las dos pipes, pero en este caso, no hay letras para ponerlas en minúsculas).

Cómo usar las pipes para dar formato en la vista

Pipes por defecto. Todas las que vienen con Angular

Angular ya viene con varias pipes listas para usarse:

- Currency: Pipes para el formateo de monedas. Para usarlas hay que pasar a la pipe la moneda que queremos usar, por ejemplo: `{{ 345.76 | currency: 'EUR' }}` Lo que imprimira: €345.76

- Date: Para el formateo de fechas. Igual que en el pipe de currency, hay que pasar un parámetro a la pipe dependiendo del formato de fechas que queremos, por ejemplo:

  `{{myVar | date: 'shortDate'}}` myVar es una variable creada en el componente de tipo Date.

Cómo usar las pipes para dar formato en la vista

Los formatos de fecha que podemos pasarle a estas pipes son:

- **'short':** por ejemplo, 6/15/15, 9:03 AM
- **'medium':** por ejemplo, Jun 15, 2015, 9:03:01 AM
- **'long'**: por ejemplo, June 15, 2015 at 9:03:01 AM GMT+1
- '**full**': por ejemplo, Monday, June 15, 2015 at 9:03:01 AM GMT+01:00
- '**shortDate**': por ejemplo, 6/15/15
- '**mediumDate**': por ejemplo, Jun 15, 2015
- '**longDate**': por ejemplo, June 15, 2015
- '**fullDate**': por ejemplo, Monday, June 15, 2015
- '**shortTime**': por ejemplo, 9:03 AM
- '**mediumTime**': por ejemplo, 9:03:01 AM
- '**longTime**': por ejemplo, 9:03:01 AM GMT+1
- '**fullTime**': por ejemplo,9:03:01 AM GMT+01:00

Cómo usar las pipes para dar formato en la vista

Decimal: Para mostrar números con coma decimal. Como argumento, recibe números indicando cuantos decimales queremos antes y después de la coma, por ejemplo:

```
<p>{{ 3.14159265 | number: '3.1-2' }}</p>
```

Indicamos que queremos 3 números antes de la coma y de 1 a 2 elementos tras la coma, lo cual imprimirá: 003.14

Cómo usar las pipes para dar formato en la vista

JSON: Si imprimimos directamente una variable que contiene un JSON, Angular imprimirá [object Object], pero si usamos esta pipe, directamente imprimirá todo el JSON:

```
<p>{{ myVal | json }}</p>
```

myVal es la variable del componente en la que guardamos el JSON. Esta pipe imprimirá:

{ moo: 'foo', goo: { too: 'new' }}

Cómo usar las pipes para dar formato en la vista

LowerCase y UpperCase: Como su nombre indican, para transformar texto a mayúsculas o a minúsculas:

```
<p>{{ 'prueba' | uppercase }}</p>

<p>{{ 'PRUEBA' | lowercase }}</p>
```

Cómo usar las pipes para dar formato en la vista

Percent: Para imprimir porcentajes, se usa igual que los decimales, indicando el número de dígitos que queremos antes y después de la coma:

```
<p>{{ 0.38679 | percent: '2.1-2' }}</p>
```

Lo que imprimirá: 38.68%

Slice: Para recortar arrays. Se pasan dos números, el índice del principio y el índice del final:

```
<p>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>
```

El resultado será: 2,3

Cómo usar las pipes para dar formato en la vista

Pipes personalizadas. Crea tus propias pipes

Si queremos hacer algo más específico, también podemos crear nuestra propia pipe.

Para crear pipes, Angular cli viene con un comando para crearlas directamente:

```
ng generate pipe
```

Programación web en el entorno servidor

GRACIAS

MANUEL MACÍAS

TUTORIAS@MANUELMACIAS.ES

SERVIDOR