

The Angular logo is a stylized 'A' composed of three concentric, curved segments in shades of blue. The word 'Angular' is written in a bold, black, sans-serif font, centered horizontally and positioned slightly above the middle of the image.

Angular

Angular

Angular es un framework basado en javascript desarrollado por Google lo que nos brinda un gran soporte en la amplia comunidad de desarrolladores que tiene.

La característica principal de Angular es que crea aplicaciones web de una página SPA (Single Page Application). Utiliza el lenguaje Typescript que es Javascript pero que engloba mas funcionalidades.

Angular

Para comenzar a trabajar con angular necesitamos preparar el entorno, y para ello visitamos la página oficial de Angular y seguimos los pasos, vamos a crear una nueva aplicación Angular desde la línea de comandos lo primero será comprobar que tenemos instalado Node.js

Node.js es un entorno de ejecución de código abierto que permite ejecutar código JavaScript en el lado del servidor. A diferencia de la ejecución típica de JavaScript en el navegador, Node.js permite a los desarrolladores utilizar JavaScript para construir aplicaciones del lado del servidor y gestionar operaciones del sistema de archivos, conexiones de red, bases de datos, y más cosas, pero ahora mismo solo nos vamos a preocuparnos porque esté instalado en nuestro sistema.

Abrimos un cmd en windows y escribimos `npm - --version`, si la versión es anterior a 8.5.0 podremos continuar.

Angular

Crearemos una carpeta para nuestro proyecto y allí mismo abriremos un nuevo cmd. O simplemente navegaremos por el cmd hasta la carpeta donde queramos crear el proyecto primero vamos a instalar la interfaz de linea de comandos de angular que facilita la creación, desarrollo y pruebas de aplicaciones en angular, para ello:

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

A continuación creamos nuestro proyecto con:

```
ng new nombreAplicacion - -- standalone=false.
```

Angular

Una vez creado el proyecto, navegaremos por el cmd hasta dentro de la carpeta de nuestro proyecto y una vez allí con el comando:

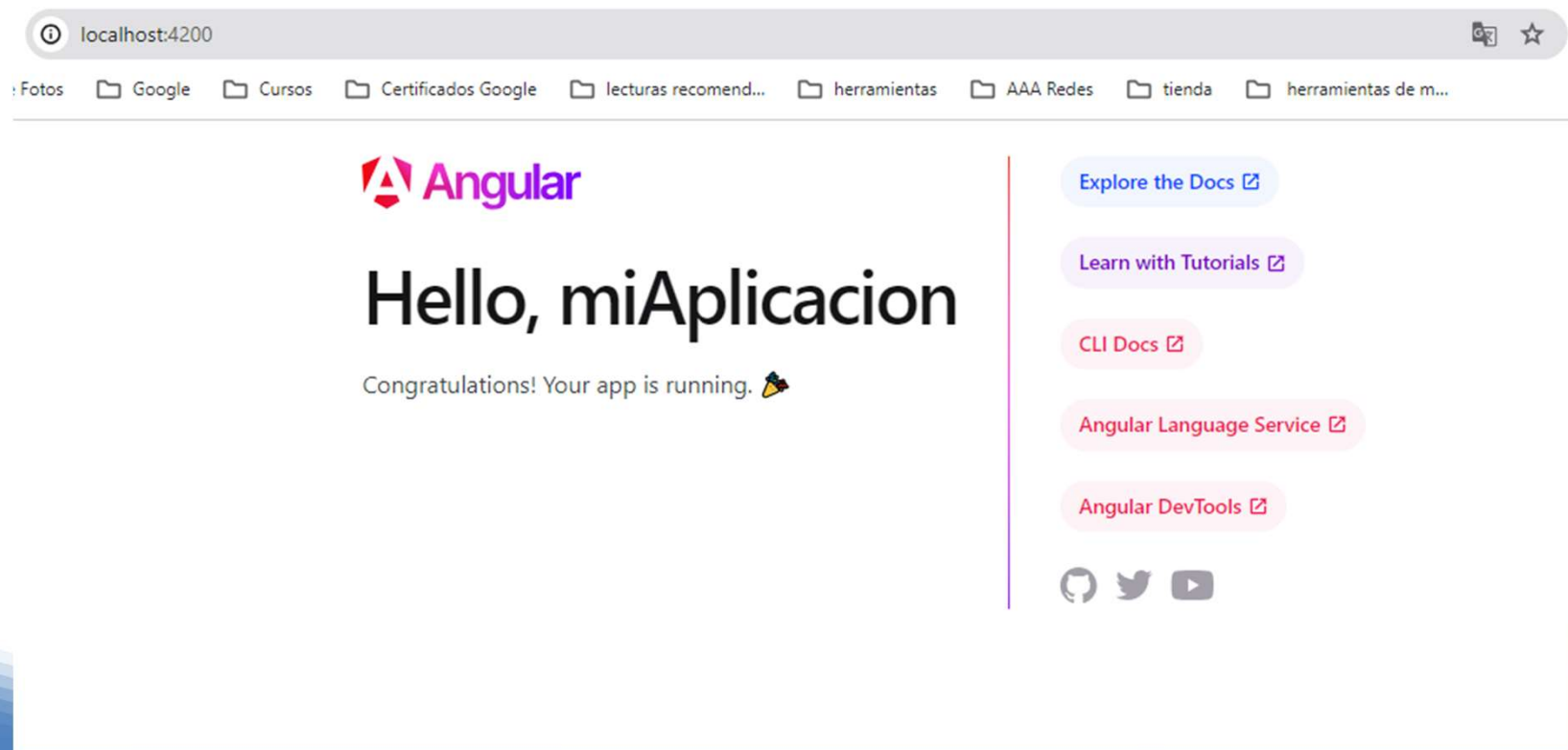
```
ng serve -o
```

Arrancaremos nuestra aplicación, que se visitará en el localhost:4200.

Podremos ver que nuestro proyecto está corriendo con la página de bienvenida de angular. En esta página por defecto además se nos muestran enlaces relevantes como la documentación y tutoriales.

Angular

Deberíamos ver algo así:



Angular

Hemos visto como levantar la aplicación, para pararla solo necesitaríamos pulsar CTRL+C se pararía y dejaría de correr nuestra app.

Si abrimos la carpeta de nuestro proyecto en vsCode veremos que se han creado varios ficheros y carpetas, en su mayoría son ficheros de configuración de angular que nosotros no vamos a tocar, casi todo lo que vamos a realizar está dentro de la carpeta src, vamos a darle un repaso

Angular

En la carpeta src nos encontramos:

- La carpeta app contiene la mayor parte del código fuente específico de tu aplicación Angular. Aquí encontrarás componentes, servicios, directivas y otros artefactos que forman parte de la lógica de tu aplicación.
- La carpeta assets se utiliza para almacenar archivos estáticos como imágenes, fuentes, o cualquier otro recurso que se deba incluir en la aplicación. Estos archivos son accesibles y se pueden cargar en la aplicación.
- Este archivo es el ícono que se mostrará en la pestaña del navegador para tu aplicación.
- El index.html es el punto de entrada principal de tu aplicación Angular. Contiene la estructura básica del HTML y suele incluir un elemento `<app-root>` que es el contenedor principal donde Angular renderizará la aplicación.
- main.ts es el archivo principal que inicia la aplicación Angular.
- El archivo main.server.ts es un archivo específico que se utiliza para iniciar la aplicación Angular en el lado del servidor.
- El archivo app.module.ts es esencial para la configuración y organización de una aplicación Angular. Define el módulo raíz, especifica los componentes, directivas y pipes que pertenecen a ese módulo, importa otros módulos necesarios, y configura diversos aspectos de la aplicación, incluido el componente raíz y el sistema de inyección de dependencias.
- Por último el styles.css para definir los estilos.

Angular.

Antes de continuar vamos a probar a abrir nuestra aplicación desde la terminal de vscode. Con nuestro proyecto en el espacio de trabajo si accedemos al menú view/terminal se nos abre una terminal en la propia carpeta del trabajo y desde ahí podremos acceder a abrir el proyecto.

Angular

En windows viene activo por defecto una política de seguridad que impide la ejecución de script desde la terminal. Si al intentar arrancar el proyecto en la terminal de vscode nos da un error tendremos que desactivar esa política, este paso se realiza solo una vez por lo que si no nos aparece el fallo es posible que anteriormente se haya desactivado.

```
PS C:\Users\Belén\Desktop\clases\0. Curso 23-24\HLC\2Trimestre\appAngular> ng server
ng : No se puede cargar el archivo C:\Users\Belén\AppData\Roaming\npm\ng.ps1
porque la ejecución de scripts está deshabilitada en este sistema. Para
obtener más información, consulta el tema about_Execution_Policies en
https://go.microsoft.com/fwlink/?LinkID=135170.
En línea: 1 Carácter: 1
+ ~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\Belén\Desktop\clases\0. Curso 23-24\HLC\2Trimestre\appAngular>
```

Angular

Para desactivar esta restricción debemos de abrir un powershell con privilegios de administrador y ejecutar la siguiente instrucción.

```
PS C:\Windows\system32> Set-ExecutionPolicy Unrestricted

Cambio de directiva de ejecución
La directiva de ejecución te ayuda a protegerte de scripts en los que no confías. Si cambias dicha directiva, podrías exponerte a los riesgos de seguridad descritos en el tema de la Ayuda about_Execution_Policies en https://go.microsoft.com/fwlink/?LinkID=135170. ¿Quieres cambiar la directiva de ejecución?
[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspender [?] Ayuda (el valor predeterminado es "N"): s
PS C:\Windows\system32>
```

Una vez confirmado podremos levantar nuestra aplicación desde la terminal de vscode sin problemas.

Angular

Una vez abierta nuestra aplicación vamos a echarle un ojo al index.html, es raro porque realmente no aparece nada, aparentemente el body y solo contiene un par de etiquetas:

```
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Y es que angular se estructura por componentes, evidentemente podríamos escribir lo que queramos dentro del body y se vería reflejado en la página principal, pero lo que actualmente aparece ¿donde está?.

Tendríamos que irnos al fichero app.component.html y todo lo que vemos en el index es el código que aparece en ese fichero, es un código genérico, podemos eliminarlo si queremos, pero para ver que funciona primero vamos a cambiarle el mensaje de bienvenida.

Angular

Una vez que hemos visto el fichero `app.component.html` y hemos cambiado el mensaje de bienvenida habremos notado que para nombrar a nuestra aplicación usamos una variable entre llaves:

```
</svg>  
<h1>Hola, {{ title }}</h1>  
<p>¡¡La aplicación está activa!! 🚀</p>  
</div>  
<div class="divider" role="separator" aria-label="Divider"></div>
```

La siguiente pregunta que se nos puede ocurrir es, donde se le da valor a esa variable, y donde se establece que este componente es el que debe aparecer en la página principal.

Angular

Revisemos el fichero app.component.ts:

```
1  import { Component } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { RouterOutlet } from '@angular/router';
4
5  @Component({
6    selector: 'app-root',
7    standalone: true,
8    imports: [CommonModule, RouterOutlet],
9    templateUrl: './app.component.html',
10   styleUrls: ['./app.component.css']
11  })
12  export class AppComponent {
13    title = 'miAplicacion';
14  }
15
```

Angular

No debe resultar difícil leer este fichero puesto que tiene poca información todavía.

Tenemos un componente cuyo selector es `app-root` que es la etiqueta que aparecía en el body, nos indica que el html que debemos colocar ahí es `app.component.html` y el correspondiente estilo.

Angular

Recapitulemos un poco, como en toda aplicación el archivo principal es el `index.html` que es donde se pondrá lo que se muestra en la página principal, en nuestro caso lo que aparece en el `index.htm` es un componente llamado `app-root`.

Este componente aparece definido dentro de `app.component.html`, aquí aparece el contenido y estilo del propio componente. Pero aun así este componente no está relacionado con el `index.html`.

En el fichero `app.component.ts` se crea una clase, `AppComponent`, en esta clase se definen parámetros y se utiliza un decorador `@Component` que relaciona el selector `app-root` con la descripción del componente `app.component.html`.

Aun así nos queda un último paso, en el fichero `app.module.ts` se referencia la clase `AppComponent` para que el componente sea finalmente reconocido.

Angular

Vamos a crear un componente, existen dos formas de hacerlo, vamos a ver primero la forma automática:

Con el comando:

```
ng generate component nombreComponente
```

Con este comando se hacen automáticos todos los pasos, que veremos posteriormente. A simple vista se ha creado una carpeta dentro de app, y dentro de ella 4 ficheros que representa, el estilo de nuestro componente, el contenido o vista de nuestro componente y además se ha generado un fichero para pruebas y un “controlador”, un fichero .ts que llevará la lógica de nuestro componente.

El fichero css y html no tienen gran información pero si entramos en el fichero componente.component.ts vemos que se ha creado una clase lista para exportar.

Angular

Esa clase presenta por arriba lo que se conoce como decorador. Un decorador se sitúa por encima de la clase y sirve para añadir metadatos, modificar el comportamiento de una función o aplicar patrones de diseño específico. En este caso el decorador `@Component` sirve para configurar los componentes. Dentro del decorador se define el selector que se utilizará en la llamada, además de la url de la vista y de la url del estilo.

El último paso que debemos tener en cuenta es en el fichero `app.module.ts` debe aparecer reflejado nuestro componente. Como lo hemos creado automáticamente no tenemos que preocuparnos por ello pero si lo hacemos de forma manual no se nos debe olvidar.

Angular

Por último para mostrar nuestro componente lo vamos a llamar por su selector dentro del fichero `app.component.html`.

Si todo ha salido bien veremos nuestro componente en la página principal de nuestro proyecto.

Angular

Vamos a empezar a practicar.

Crea un proyecto en Angular que se llamará Biblioteca, modifica la apariencia de la página principal quitando los datos que no son relevante y añadiendo lo necesario para que parezca la página de inicio de una biblioteca.

Crea un componente de forma automática que será un menú de navegación.

Crea de forma manual un componente que será un formulario de contacto. La forma de crear el componente de forma manual es simplemente creando la misma estructura de carpetas que se crean automáticamente.

Angular

Concepto de Interpolación:

La interpolación es la técnica utilizada para insertar valores de expresiones directamente en la vista de la aplicación. Permite la vinculación unidireccional de datos, lo que significa que puedes mostrar dinámicamente datos en tu interfaz de usuario.

En Angular la interpolación se realiza utilizando la doble llave '{{ }}'. Dentro de estas llaves puedes colocar expresiones que se evaluarán y se mostrarán en el lugar correspondiente de la plantilla.

Cuando creamos un proyecto nuevo en la `app.component.ts` está declarada la clase principal y dentro se declara la variable `titulo`, y se le da un valor. En el fichero `app.component.html` recuperamos esa variable con la expresión `{{titulo}}`. A eso lo llamamos interpolación.

Angular

La interpolación entonces se refiere a declarar una propiedad en la clase y llamarla en la plantilla.

Como tenemos nuestro proyecto con los cuatro div que creamos en la clase anterior. Vamos a crear alguna variable dentro del componente del div. Las variables van a ser el ancho y el largo solo en número. Y vamos a llamar a esas propiedades dentro del componente de forma que aparezca algo similar a lo siguiente:

Angular

Esto son variables introducidas desde la clase:

ancho: 400
alto: 400

Esto son variables introducidas desde la clase:

ancho: 400
alto: 400

Puedo modificar valores con operadores aritméticos, por ejemplo multiplico el alto por dos

el alto ahora vale: alto: 800

400 y 400 son valores que le he dado a una variable llamada ancho y otra llamada alto en la clase, además puedo realizar operaciones con estos valores.

Angular

Se nos permite el uso de condicionales ternarios dentro de las llaves de interpolación:

```
{{ancho>600? "Pantalla Grande" : "Pantalla Pequeña"}}
```

Evaluamos el valor de ancho y en función del resultado de la evaluación devolvemos un string u otro; Esto funciona también con variables, en lugar de string podemos llamar a variables declaradas en la clase en ese caso no necesitamos poner más llaves que las exteriores:

```
alto = 40;  
pantalla = "grande";  
monitor = "pequeño";  
  
19 </div>  
20 <br/><br/><br/><br/><br/>  
21 {{alto>100? pantalla: monitor}}  
22
```


Angular

Como vemos la idea es ir declarando parámetros en la clase y llamarlos en el html dentro de llaves dobles.

¿Recordáis la forma de construir una clase en java?, los parámetros de la clase debían tener un modificador de acceso para limitar que pudieran ser modificados desde fuera de la propia clase. Por ese motivo siempre los establecíamos a private. Esos modificadores afectan aquí también y si queremos que nuestros parámetros no sean accesibles los marcaremos como private.

Una vez que hemos realizado el ejercicio anterior y visto que funciona correctamente si a la variable ancho por ejemplo le damos el modificador de acceso de private nos genera un fallo que podemos ver en la terminal de visual, que nos indica que en el html la variable ancho no es accesible.

Angular

Para acceder a parámetros declarados como private la solución es declarar un get igual que hacíamos en java.

```
7
8
9
10
11
12
13
14
15
export class NavegacionCom
    private ancho = 400;
    alto = 400;
    getAncho() {
        return this.ancho;
    }
    <div class="mi-div">
    <p>Esto son variables introducidas desde la clase: </p>
    ancho: {{getAncho()}}
    <br/>
    alto: {{alto}}
    <hr/>
```

Si declaro la función getAncho ya podré acceder al valor de ancho, llamando a la función en la vista.

Angular

Vamos a jugar un poco con las variables.

Vamos a crear un proyecto con un componente que sea información de contacto y que nos muestre lo siguiente:

Nombre: xxx

Edad: xx

Contraseña: xx

Nombre y Edad serán parámetros de la clase y contraseña el resultado de concatenar nombre+edad;

La contraseña será un campo private y solo vamos a poder acceder a el fuera de la clase a través del método get.

Mostraremos toda la información en el nuestra página principal pero en el caso de que la edad del usuario sea menor de 18 no la mostraremos sino que diremos que es menor de edad, en caso contrario si mostraremos su edad.

Angular

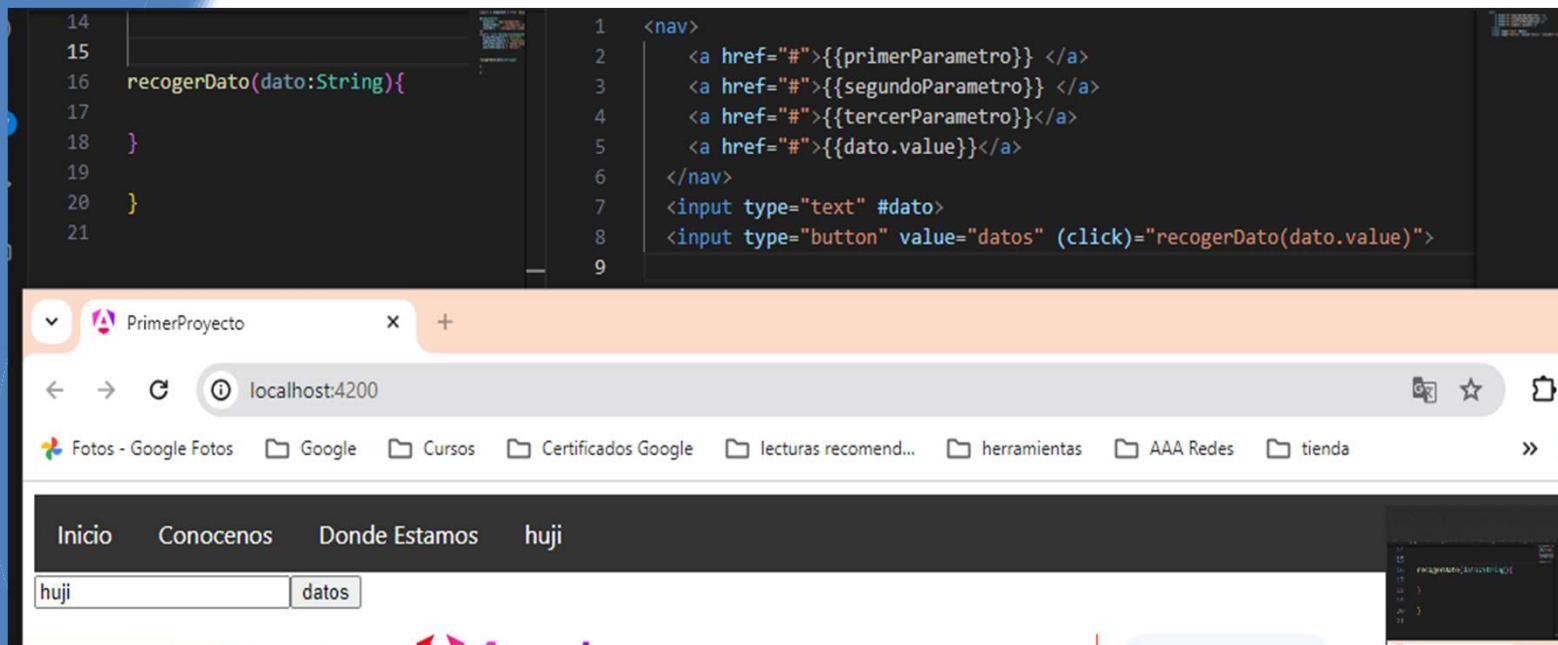
Hasta aquí está interesante, pero lo verdaderamente interesante sería poder introducir valores en la página que queden recogidos en variables..... Vamos a ver como podríamos hacerlo.

Hablamos de crear un puente entre el controlador y la vista y que se puedan comunicar. En Angular hablamos de Property Binding cuando ese puente se realiza para pasar propiedades, y Event Binding cuando intervienen eventos.

En primer lugar en nuestro componente necesitaremos introducir un input para poder introducir datos y un botón para recoger el evento click. Para localizar nuestro elemento dentro del DOM le daremos un nombre precedido de # . Y crearemos un evento click para recoger el dato.

Angular

Para el evento click necesitamos definir una función que nombraremos en el controlador aunque por el momento no necesitamos desarrollar mas. Con este código conseguimos que al hacer click en el botón el contenido del campo input pueda ser utilizado.



The image shows a development environment with two main parts: a code editor and a web browser.

Code Editor (Left): Displays a TypeScript function named `recogerDato` in a file named `recogerDato.ts`. The function signature is `recogerDato(dato:String){`. The body contains a closing curly brace `}` on line 18, followed by another closing curly brace `}` on line 20, and a final closing curly brace `}` on line 21.

Code Editor (Right): Displays HTML code in a file named `recogerDato.html`. It features a navigation bar `<nav>` with four links: `{{primerParametro}} `, `{{segundoParametro}} `, `{{tercerParametro}}`, and `{{dato.value}}`. Below the navigation bar is a text input `<input type="text" #dato>` and a button `<input type="button" value="datos" (click)="recogerDato(dato.value)">`.

Web Browser: The browser window shows the application running on `localhost:4200`. The page has a dark header with navigation links: `Inicio`, `Conocenos`, `Donde Estamos`, and `huji`. Below the header, there is a form with a text input containing the value `huji` and a button labeled `datos`.

Declaramos un input de tipo texto donde introduciremos el dato.

Declaramos un input de tipo button. le damos un texto (value) al botón y le asociamos el evento click que llamará a la función recoger Datos que cogerá el valor del input cuya id(#) será dato.

Angular

Inténtalo tu mismo, crea un componente que recoja el nombre, apellido, y profesión de una persona y lo muestre cuando hagamos click en un botón.

Angular

Existen otros eventos que podemos utilizar para mostrar de forma dinámica datos que vayamos introduciendo en los cuadros de texto:

```
32
33
34 <input type="text" #dato>
35 <input type="button" value="dato" (click)="recogerDato(dato.value)">
36 <br/><br/>
37 <label>nombre</label>
38 <input type="text" #nombre (keyup)="0">
39 <br/>
40 <label>apellido</label>
41 <input type="text" #apellido (blur)="recogerDatoConBlur(apellido.value)">
42 <br/>
43
44
45
```

```
recogerDato(dato: String) {
}
recogerDatoConBlur(dato: String)
}
```

Dentro de las funciones en el controlador definiremos que tenemos que hacer con los datos recogidos.

Angular

Por el momento hemos visto como desde el controlador se pasan valores a la vista y hemos intuido un poco con los eventos que desde la vista se pueden pasar al controlador datos que devolverá a la vista. Lo que pasa es que los métodos que hemos visto hasta el momento parecen vacíos.

Vamos a ver de forma dinámica como podemos alterar las propiedades de nuestra vista con los datos que vamos introduciendo de forma dinámica.

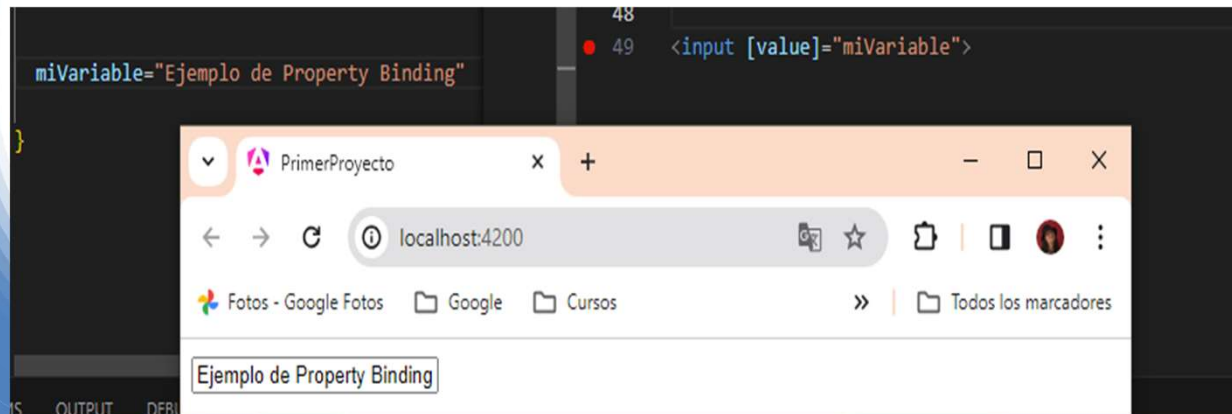
Angular

Hablemos ahora de Property Binding, esto es una forma de enlazar propiedades de un componente a propiedades del DOM. Permite que los valores en el componente sean reflejados en el DOM, de modo que cualquier cambio en el componente se refleje automáticamente en la interfaz de usuario.

A continuación veremos un ejemplo de sintaxis básico, un ejemplo con eventos y otro con estilos.

Angular

Comencemos por la parte básica los Property Binding se utilizan con corchetes, de forma que se puede declarar una variable en el controlador y pasarla al DOM de la siguiente forma:

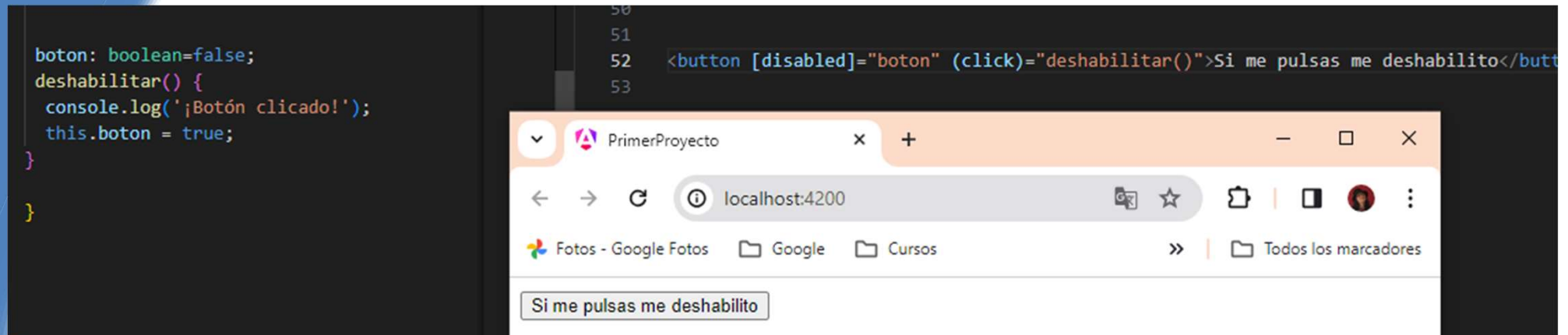


En el controlador declaramos la variable. En la vista utilizamos la variable para darle valor a un elemento del DOM

La diferencia con lo visto anteriormente es que antes utilizábamos el valor de la variable pero para reflejarla en la vista, ahora utilizamos la variable para interactuar con el DOM

Angular

Podemos usar esta propiedad también con eventos:



Comenzamos en el controlador declarando la variable boton a false, aunque podría hacerse como “boton=false” es buena práctica declarar el tipo, porque será más fácil leer y depurar errores.

Declaramos una función deshabilitar que informa de que hemos clicado el botón y cambiamos el valor de la variable.

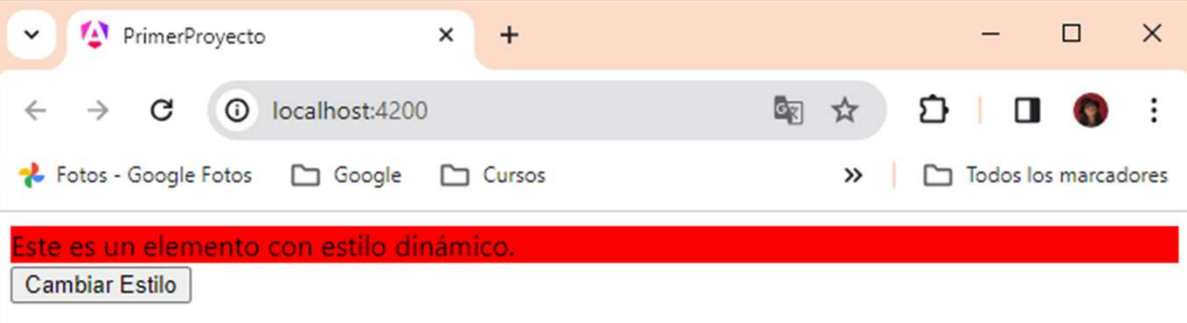
En la vista a la propiedad disabled le damos el valor de boton antes de clicar y con el evento click llamamos a la función deshabilitar.

Angular.

Veamos como podemos darle estilos dinámicos a nuestra vista.

```
TS navegacion.component.ts U X
primerProyecto > src > app > navegacion > TS navegacion.component.ts > ...
42 colorFondo: string = 'red';
43 esEstiloResaltado: boolean = false;
44 cambiarEstilo() {
45   this.colorFondo = this.esEstiloResaltado ? 'red' : 'blue';
46   this.esEstiloResaltado = !this.esEstiloResaltado;
47 }
48 }

navigacion.component.html U X
primerProyecto > src > app > navegacion > <> navegacion.component.html > ...
59 <div [style.background]="colorFondo" [class.resaltado]="esEstiloResaltado">
60   Este es un elemento con estilo dinámico.
61 </div>
62 <button (click)="cambiarEstilo()">Cambiar Estilo</button>
```



En el controlador:

Declaramos una variable color de fondo que será de tipo string y tendrá valor 'red'. Declaramos otra variable estiloResaltado, que será boolean e inicializada a false. Declaramos una función donde mediante condicional comprobamos si el estilo es resaltado y se le da un valor u otro, y además se cambia el valor de la variable estilo resaltado.

En la vista:

Creamos un div con color de fondo y clase resaltada en función del valor de la variable del controlador. y un botón que al hacer click llamamos a la función para cambiar el estilo

Angular

Recordemos Property Binding cuando hablamos de propiedades. Variables declaradas en el controlador y utilizadas en la vista para actualizar propiedades en los elementos del DOM. La propiedad se sitúa entre corchetes y se le asigna el valor de la propiedad en el controlador.

Event Binding, este puente se tiende desde la vista al capturar un evento y la información la pasamos al controlador para que ejecute el código que normalmente se desarrolla dentro de una función

Angular

Veamos otro ejemplo, esta vez vamos a capturar el valor de un radio-button. Recordemos que los radio buttons sirven para seleccionar una opción entre varias, entre los atributos del elemento tenemos un `name` que debe ser igual para todos y un `value` que será lo que se muestre en la vista.

Podemos hacerlo de varias formas, por un lado podemos capturar el evento que para este caso puede ser más complicado, o con una función como la vista anteriormente.

Angular

En primer lugar en la vista declaramos por un lado dos radio button para definir genero a estos le vamos a pasar el evento.

En segundo lugar y puesto que igualmente el evento click lo vamos a tener que pasar en todos los radio button y por tanto llamar a la misma función en cada opción en lugar de acceder al evento y sus parámetros vamos a pasar el valor directamente en formato cadena.

Genero:

```
<input type="radio" name="genero" value="femenino" (click)="seleccionarGenero($event)">  
<input type="radio" name="genero" value="masculino" (click)="seleccionarGenero($event)" >
```

Genero2:

```
<input type="radio" name="genero" value="femenino" (click)="seleccionarGenero2('femenino')">  
<input type="radio" name="genero" value="masculino" (click)="seleccionarGenero2('masculino')" >
```

Angular

Ahora en el controlador vamos a definir las funciones:

```
seleccionarGenero(event: Event) {  
  console.log(<HTMLInputElement>event.target.value)  
}  
genero: string = "";  
seleccionarGenero2(option: string) {  
  this.genero = option;  
  console.log("Este genero es con String: " + this.genero)  
}
```

En la primera función recogemos el evento. El evento es un objeto y como tal tiene sus propiedades, en este caso si hacemos un `console.log` de `event.target` nos informa que es un elemento tipo `HTMLInputElement` y por tanto antes de acceder al valor hay que moldearlo para acceder a su valor.

En la segunda función al pasarle directamente el valor como cadena puedo acceder a ella en cualquier momento.

Angular

Ejercicio:

Crea un nuevo proyecto Angular, crea un componente contador que tendrá input text de solo lectura con el valor 0. Debajo debe de haber dos botones uno de incremento y otro de decremento a medida que se vayan pulsado el valor del input debe de modificarse.

Incrementa y decrementa en 1.

Angular

Directivas:

En Angular, las directivas son un conjunto de funcionalidades que extienden el comportamiento del DOM (Document Object Model) en una aplicación web. Son construcciones fundamentales para la creación de aplicaciones dinámicas y reactivas. Las directivas permiten manipular y personalizar la apariencia y el comportamiento de los elementos HTML en función de la lógica de la aplicación. Hay dos tipos principales de directivas en Angular:

Directivas Estructurales:

- Son responsables de manipular la estructura del DOM, es decir, controlan la creación, eliminación o modificación de elementos en la página. Ejemplos comunes incluyen ngIf, ngFor y ngSwitch.

Directivas de Atributos:

- Se utilizan para cambiar la apariencia o el comportamiento de un elemento HTML específico al agregar, modificar o eliminar atributos. Ejemplos incluyen ngClass, ngStyle y ngModel.

Angular

Las directivas en Angular se identifican por el prefijo ng, seguido por el nombre de la directiva. Además de las directivas incorporadas, Angular permite crear directivas personalizadas para abstraer y reutilizar lógica específica de la aplicación.

En resumen, las directivas en Angular son una parte esencial del framework que proporciona un mecanismo para extender y controlar el comportamiento del DOM, permitiendo así la creación de aplicaciones web más interactivas y dinámicas.

Angular

Vamos a crear un nuevo proyecto para trabajar con las directivas.

Para darle algo de estilos vamos a instalar bootstrap y para que tenga toda su funcionalidad jquery y popper. Para ello:

```
npm install bootstrap jquery popper.js - -- save
```

Una vez instaladas debemos registrarlas en el fichero angular.json, buscamos los estilos y los scripts y añadimos las siguientes líneas de código.

Angular

En el fichero angular.json:

```
  "styles": [  
    "src/styles.css",  
    "node_modules/bootstrap/dist/css/bootstrap.min.css"  
  ],  
  "scripts": [  
    "node_modules/jquery/dist/jquery.slim.min.js",  
    "node_modules/popper.js/dist/umd/popper.min.js",  
    "node_modules/bootstrap/dist/js/bootstrap.min.js"  
  ],  
  "server": "src/main-server.ts"
```

Angular

Hasta el momento hemos creado entradas de texto que nos permitían acceder a los datos pero para manejar mejor la recogida de datos vamos a importar el modulo de formularios que nos ofrece una forma estructurada de manejar la entrada de usuarios y validar datos.

Para ello en el fichero `app.module.ts` añadimos en el apartado de imports “FormsModule”

Angular

En el fichero `app.components.ts` vamos a cambiar el `title` y poner uno que haga referencia a lo que estamos trabajando. Vamos a realizar un listado de usuarios por lo tanto cambiamos el nombre a “Listado de usuarios”

Vamos a crear una clase `Usuario` (no un componente) un modelo con el que vamos a poder crear varios objetos de tipo usuario, y para ello creamos en la carpeta `app` un nuevo fichero que llamaremos `usuarios`.

Angular

En nuestro fichero usuario.models.ts comenzamos con la declaración de la clase y la definición de atributos:

```
vas > directivas > src > app > TS usuario.model.ts > Usuario
export class Usuario{
  nombre:string="";
  apellido:string="";
  edad:number=0;

  constructor(nombre:string, apellido:string, edad:number){
    this.nombre=nombre;
    this.apellido=apellido;
    this.edad=edad;
  }
}
```

Esto recuerda
mucho a Java
¿no creéis?

Angular

Volvemos al fichero `app.component.ts` y vamos a agregar debajo de título algunos usuarios, pero para ello debemos importar la clase `usuario` que acabamos de crear.

```
Directivas > directivas > src > app > TS app.component.ts > AppCo
1  import { Component } from '@angular/core';
2  import { Usuario } from '../usuario.model';
3
4  @Component({
5    selector: 'app-root',
6    templateUrl: './app.component.html',
7    styleUrls: ['./app.component.css']
8  })
9  export class AppComponent {
10    title = 'Listado de Usuarios';
11    usuarios: Usuario[] = [
12      new Usuario("Ana", "Aranda", 39),
13      new Usuario("Marta", "Cuevas", 9),
14      new Usuario("Carlos", "López", 3),
15    ];
16  }
17
```

Angular

Para mostrar nuestros Usuarios vamos al html (app.component.html) y vamos a mostrar estos usuarios de prueba, usamos un for (*ngFor), declaramos una variable (let usuario) del array de usuarios declarado en la clase del componente. Y por cada instancia de usuario imprimimos su nombre apellido y edad.

```
Directivas > directivas > src > app > <> app.component.html > div.container
1  <div class="container">
2    <div class="row">
3      <div class="col">
4        <div *ngFor="let usuario of usuarios">
5          {{usuario.nombre}} {{usuario.apellido}} {{usuario.edad}}
6        </div>
7      </div>
8    </div>
9  </div>
```

Angular

Si observamos la salida por pantalla al levantar el servidor serán los usuarios que hemos creado de muestra, uno debajo de otro, pero podemos añadir algo más y es indicar el índice de cada vuelta de bucle y eso lo hacemos declarando una variable dentro del `ngFor` e igualándola a `index`:

```
<div *ngFor="let usuario of usuarios; let i=index">  
  {{i+1}} {{usuario.nombre}} {{usuario.apellido}} {{usuario.edad}}  
</div>
```

1 Ana Aranda 39
2 Marta Cuevas 9
3 Carlos López 3

Angular

Pero además de mostrar los de muestra queremos poder crear nuestros propios usuarios que se añadan al array de usuarios declarado en `app.component.ts` configuramos el html con un formulario para poder recoger los datos, y atención a la directiva `ngModel` que nos va a facilitar mucho las cosas:

```
<div class="container">
  <div class="col">
    <h1>{{title}}</h1>
    <br/><br/>
    <form class="form-group row">
      <div class=" form-group col-md-3">
        <label for="nombre"> Nombre: </label>
        <input type="text" name="nombre" id="nombre" placeholder="Nombre" class="form-control" [(ngModel)]="datoNombre">
      </div>
    </form>
  </div>
</div>
```

Angular

La directiva `ngModel` se utiliza para establecer un enlace entre el elemento del formulario y una propiedad del componente, que necesitamos crear, porque ahora mismo no la tenemos, pero lo que escribamos en el cuadro de texto va a viajar hasta el `app.component.ts` y el valor se va a almacenar en la variable `dato nombre` que deberemos crear allí, y lo mismo haremos con el resto de campos. Como queda si lo creáis guardáis y actualizáis?

Angular

Si todo va bien y funciona solo nos queda añadir un botón de tipo submit con el evento click que eso ya lo vimos y que llame a una función añadirUsuarios

```
export class AppComponent {  
  title = 'Listado de Usuarios';  
  usuarios: Usuario[] = [  
    new Usuario("Ana", "Aranda", 39),  
    new Usuario("Marta", "Cuevas", 9),  
    new Usuario("Carlos", "López", 3),  
  ];  
  
  añadirUsuario(){  
    let miUsuario = new Usuario(this.datoNombre, this.datoApellido, this.datoEdad)  
    this.usuarios.push(miUsuario);  
  }  
  
  datoNombre:string="";  
  datoApellido:string="";  
  datoEdad:number=0;  
}
```

Declaramos una variable `miUsuario` y la igualamos a un objeto de tipo `Usuario` que rellenamos con los valores de los campos del formulario. Aunque las variables están vacías, en el momento de hacer click recojen los valores del formulario con `ngModel` y podemos usarlas.

Angular

Ejercicio 3

Crea un nuevo proyecto para listar asignaturas, las propiedades de asignaturas serán temática, profesor, nº de horas. Realiza un listado como el visto en el ejemplo anterior, pero aplicale tus propios estilos.

Angular

Ejercicio 2.

Vamos a crear una calculadora, para ello vamos diseñar dos campos de texto donde podremos introducir números.

Debajo cuatro botones con las cuatro operaciones básicas.

Por ultimo un div donde aparezca el resultado de las operaciones.

Utiliza la directiva ngModel para recoger los valores del numero1 y numero2.