MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Sitios web de página única

15

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Fundamentos de Angular 4

/ 3. Instalación y ejecución 4

/ 4. Módulos 5

/ 5. Caso práctico 1: “Reutilización de módulos” 6

/ 6. Módulos y componentes 6

/ 7. Metadatos de componentes, plantillas y vistas 7

/ 8. Ejemplo de plantillas y vistas 8

/ 9. Pipes y rutas 9

/ 10. Formularios 10

/ 11. Caso práctico 2: “Vinculación de la vista y el modelo” 10

/ 12. Resumen y resolución del caso práctico de la unidad 11

/ 13. Bibliografía 11

OBJETIVOS

Comprender los principios de Angular.

Saber crear un sitio web básico en Angular.

Comprender el binding dinámico.

Saber crear componentes.

Comprender el funcionamiento de las vistas.

Saber crear vistas con directivas Angular.

Saber utilizar tuberías.

/ 1. Introducción y contextualización práctica

La forma de crear contenido web se hace cada vez más flexible debido a las nuevas tecnologías que surgen en este entorno. Al movernos en un contexto de desarrollo en la parte del cliente, vemos que existe un gran número de librerías y *frameworks* para facilitar el desarrollo a los programadores.

En la actualidad, uno de los *frameworks* más utilizado es Angular, pues permite crear un tipo de web que puede ser visible en cualquier navegador y en cualquier dispositivo utilizando una única página.

Uno de los mayores inconvenientes actuales en el desarrollo web es que, para poder desarrollar en un entorno como Angular, es necesario conocer un gran número de tecnologías previamente, no solo que tengan que ver con el desarrollo web, sino también relacionadas con la gestión de paquetes, dependencias, etc. Además, es necesario estar familiarizados con el proceso de compilación y con la utilización de servidores web.

Fig. 1. Logo de Angular.

Al utilizar estas nuevas tecnologías, las páginas web dejan de ser interpretables directamente por el navegador, requiriendo de un procesamiento adicional que, en el caso de Angular, requieren su motor de interpretación.

Escucha el siguiente audio que describe el caso práctico que iremos resolviendo a lo largo de la unidad.

Audio intro. "Nuevas tendencias en el desarrollo web"
http://bit.ly/2WnkztM

/ 2. Fundamentos de Angular

Angular comenzó siendo una pequeña librería JavaScript que era necesaria incluir en el proyecto para facilitar el desarrollo y proporcionar determinadas funcionalidades. Sin embargo, el desarrollo de Angular ha permitido crear una nueva filosofía a la hora de crear soluciones web. Se trata de una tecnología que no solo se aplica a la parte cliente, sino que también se pueda aplicar a la parte del servidor, e incluso a entornos móviles.

Angular se puede considerar una plataforma y un *framework* para desarrollar páginas siguiendo la filosofía single-page (página única).

Para desarrollar en Angular, es necesario tener conocimientos, al menos, de HTML y TypeScript, de hecho, Angular está desarrollado en TypeScript.

La arquitectura de Angular se centra en unos conceptos fundamentales, como son:

- La utilización de ***NgModules***, que proporciona determinados contextos para la compilación de los componentes. Los módulos en Angular consisten en colecciones de código de la misma área funcional.
- Una aplicación Angular está formada por un conjunto de *NgModules*. Nótese que todo lo que esté precedido del prefijo ***Ng*** hace referencia a Angular.
- Normalmente, una aplicación en Angular tiene un módulo principal. Es posible añadir otros módulos para crear nuevas funcionalidades.
- Los componentes permiten definir vistas que se adaptan, dependiendo de la lógica del negocio.
- Podemos definir servicios que se utilizan por los componentes e incluyen funcionalidades muy específicas, que no están relacionadas con la vista.

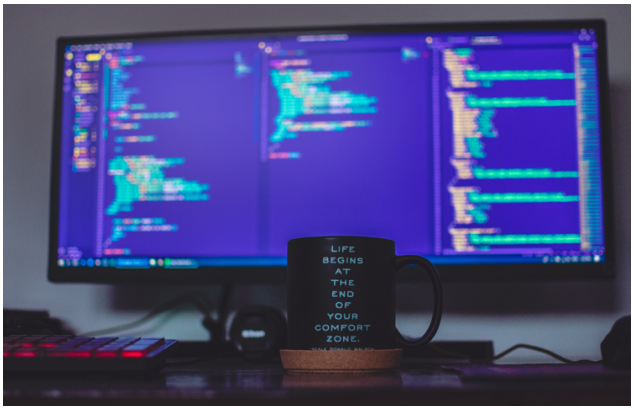

Fig. 2. Angular requiere nuevas ideas para el desarrollo web.

/ 3. Instalación y ejecución

Para poder comenzar el desarrollo en Angular, es necesario configurar el entorno y disponer de **conocimientos de**, al menos: **HTML, JavaScript, CSS** y **TypeScript**.

Para poder configurar el entorno correctamente, es necesario **instalar el gestor de paquetes *Node.js***, que ya hemos visto anteriormente.

Una vez que tenemos este paquete, hay que **instalar el CLI de Angular utilizando *npm***. Para ello, tecleamos la siguiente sentencia en una línea de comandos: ***npm install -g @angular/cli***.

Con esto seremos capaces de instalar el intérprete de comandos de Angular para poder comenzar a trabajar.

Una vez que disponemos del paquete instalador, podremos crear nuestro primer proyecto en Angular. Para ello, tecleamos el siguiente comando: ***ng new my-test-app***.

A través de este comando se va a crear un proyecto angular llamado ***my-test-app***, utilizando como directorio raíz el directorio actual. Es posible que la preparación de este comando se demore un tiempo, pues deberá descargar e instalar todos los paquetes necesarios.

Una vez que el proyecto ha sido creado correctamente, podremos ejecutar nuestra aplicación en el navegador. Para ello, es necesario disponer de un servidor web local que permite transformar las sentencias Angular a código HTML. Si ejecutamos la instrucción ***ng serve –open*** en la línea de comandos, se nos abrirá un navegador con nuestra página web creada. A partir de este momento, cada vez que realicemos un cambio en el código fuente, se visualizará en el navegador.

Fig. 3. Angular permite crear aplicaciones muy potentes.

Vídeo 1. "Mi primer proyecto con Angular"
https://bit.ly/3mql2G8

/ 4. Módulos

Como hemos visto en temas anteriores, se considera una buena práctica organizar el código en módulos para facilitar su mantenimiento y favorecer la reutilización.

En Angular, las aplicaciones son modulares por defecto y, además, ofrece su propio sistema de modularidad que se llama ***NgModules***. Toda aplicación en Angular dispone, al menos, de una clase ***NgModule***, que normalmente tiene el nombre de ***AppModule***.

```
import { NgModule }     from '@angular/
core';
import { BrowserModule } from '@angular/
platform-browser';
@NgModule({
 imports:     [ BrowserModule ],
 providers:   [ Logger ],
 declarations: [ AppComponent ],
 exports:     [ AppComponent ],
 bootstrap:   [ AppComponent ]
})
export class AppModule { }
```

Código 1. Clase NgModule.

Audio 1. "Módulos de Angular"
http://bit.ly/37lKSHg

En el ejemplo anterior se muestra una clase *NgModule* de una aplicación Angular. Como se pueda observar, la declaración del *NgModule* se divide en:

- **imports:** incluye todos los módulos que hay que importar para que este componente funcione correctamente.
- **providers:** permite indicar los constructores de los servicios que se utilizan en este módulo.
- **declarations:** permite declarar todos los elementos que pertenecen a este módulo.
- **exports:** permite indicar todos los componentes que son visibles y utilizables desde este módulo.
- **bootStrap:** hace referencia a la vista de la aplicación principal.

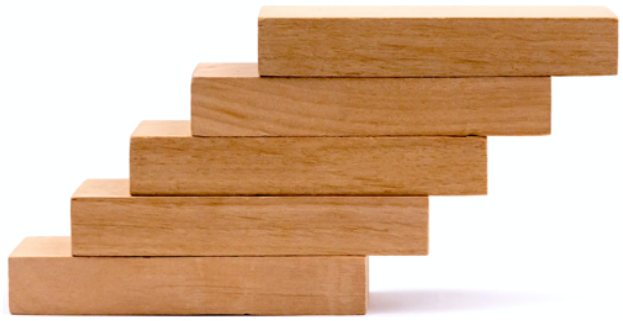

Fig. 4. Angular se organiza en módulos.

/ 5. Caso práctico 1: "Reutilización de módulos"

Planteamiento: Como responsables del desarrollo de una aplicación en Angular, decidimos utilizar este enfoque para el desarrollo de todas nuestras aplicaciones. La idea es basarnos en el desarrollo de módulos genéricos que puedan reutilizarse fácilmente a través del gestor de paquetes.

Nudo: ¿Crees que se trata de una buena alternativa?

Desenlace: Como hemos visto en el apartado sobre Angular, es posible crear módulos que pueden ser reutilizados en diferentes contextos. Se trata de una buena práctica para poder reutilizar código y realizar una buena organización de este. A través del gestor de paquetes, es posible reutilizar código de una manera sencilla y productiva.

Fig. 5. La reutilización de módulos es una de las características que hacen a Angular más potente.

/ 6. Módulos y componentes

a. **Módulos**

Un módulo en Angular ofrece un **contexto para la compilación de sus propios componentes**. Dicho módulo dispondrá, siempre, de un **componente raíz** que se va a crear durante el arranque de la aplicación. **Cada módulo puede tener un número indeterminado de componentes que puede cargar a través de un enrutador**.

Angular carga un conjunto de módulos JavaScript que, desde el punto de vista del usuario, pueden verse como librerías.

Cada librería en Angular hay que especificarla con el prefijo: ***@angular***.

Para instalar librerías adicionales, es necesario utilizar el gestor de paquetes ***npm***.

A continuación, se muestra un ejemplo de importación de librerías:

```
import { Component } from '@angular/core';
import { BrowserModule } from
'@angular/platform-browser';
```

Código 2. Importación de librería.

b. **Componentes**

En Angular, un componente es un **elemento que es capaz generar una vista.** Normalmente, la lógica de aplicación se define en componentes que disponen de una vista y una clase asociada.

La clase puede interactuar con la vista a través del API de Angular, utilizando como lenguaje TypeScript o, en su defecto, JavaScript. Todos los componentes hay que crearlos bajo el directorio: ***src/app***.

```
export class ClassTest implements OnInit {
 students: Student[];
 selectedItem: Student;

 constructor(private service: StudentService) { }

 ngOnInit() {
  this.students = this.service.getStudents();
 }

 selectStudent(student: Student) {
this.selectedItem = student; }
}
```

Código 3. Componentes.

/ 7. Metadatos de componentes, plantillas y vistas

a. **Metadatos de componentes**

Angular utiliza el **patrón de diseño decorador** para los componentes. Utilizando la anotación ***@Component***, es posible identificar la clase y especificar los metadatos de esta.

En el ejemplo anterior, se creaba la clase ***ClassTest***, pero **no se considera un componente hasta que se marque con el decorador que incluye los metadatos**. Utilizando la información de los metadatos, Angular es capaz de obtener la información necesaria para crear y presentar el componente y su vista correspondiente. Angular es capaz de asociar una plantilla con el componente que, en conjunto, definen la vista. Además, los metadatos pueden indicar cómo acceder al documento HTML de la vista y qué servicios se necesitan.

```
@Component({
 selector:   'app-student-list',
 templateUrl: './student-list.component.html',
 providers:  [ StudentService ]
})
export class ClassTest  implements OnInit {
/* . . . */
}
```

Código 4. Metadatos.

b. **Plantillas y vistas**

Para definir la vista de un componente, es necesario asociar una plantilla con la clase.

Una **plantilla** consiste en un **documento HTML que le indica a Angular cómo representar el componente**. De esta forma, una plantilla se asemeja a un documento HTML "normal", pero con la diferencia de que puede contener sentencias embebidas de Angular. Así, se puede alterar el comportamiento del HTML y los datos del DOM. La plantilla está enlazada con el componente, con lo que existe una coordinación entre los datos del componente y los de la plantilla, pudiendo ser actualizados automáticamente.

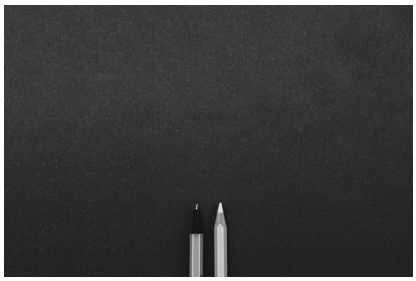

Fig. 6. Angular permite la utilización de plantillas y vistas.

/ 8. Ejemplo de plantillas y vistas

```
<h2>Student List</h2>
<p><i>Pick a Student from the list</i></p>
<ul>
  <li *ngFor="let item of students"
(click)="selectStudent(item)">
    {{student.name}}
  </li>
</ul>

<app-student-detail *ngIf="selectedItem"
[student]="selectedItem"></app-student-detail>
```

Código 5. Interacción con el DOM de la vista.

En el ejemplo anterior se muestra un código HTML que tiene en su interior sentencias de Angular para poder facilitar la interacción con el DOM en la propia vista.

Como se puede observar, se utiliza una sentencia iterativa como ***ngFor***, la cual permite iterar dentro del documento HTML. Esta sentencia define el comportamiento en bucle ***for*** con una cadena de caracteres y permite añadir tantos elementos al documento HTML como elementos posea la lista sobre la que se está iterando.

Si nos fijamos, es una forma totalmente diferente de controlar el DOM. Si utilizásemos JavaScript, tendríamos que generar el código HTML desde el propio lenguaje. Al utilizar Angular, este código es autogenerado. **En Angular**, cuando se utiliza **{{..}}** hacemos referencia al componente asociado, de tal forma que **si cambia el componente, cambia la visualización en el HTML**. Este mecanismo se conoce como **enlace dinámico**. Si no utilizásemos Angular, sería necesario manipular el DOM directamente con JavaScript/TypeScript para poder emparejar cada uno de los valores de componente y la vista. Este enlace que ofrece Angular es bidireccional: si cambia el componente, también cambia la vista, y viceversa.

Supongamos un escenario en el que, en lugar de tener una etiqueta en la vista, tenemos un cuadro de texto que se puede escribir. En este caso, al introducir datos en la vista, el componente también estaría actualizado con el valor de entrada.

Fig. 7. Las plantillas en Angular son una herramienta muy potente.

/ 9. Pipes y rutas

a. **Pipes**

Angular permite utilizar tuberías para realizar transformaciones de visualización desde el componte a la plantilla. Una clase que defina el decorador ***@Pipe*** permite crear una función que transforme los valores de entrada en una salida con un formato determinado.

b. **Rutas**

Gracias a la utilización de aplicaciones de una única página en Angular, los usuarios permanecen siempre en una única página y lo único que cambia es la vista según la interacción que realice el usuario. Para controlar la navegación, es necesario utilizar el enrutado que permite mapear URLs con componentes.

Para trabajar con rutas en Angular, es necesario crear una aplicación con las rutas activadas. Para ello, hay que realizar lo siguiente desde la línea de comandos:

```
ng new routing-app --routing
ng generate component first
ng generate component second
```

Código 6. Creación de aplicación con rutas activadas.

Con las líneas anteriores, Angular creará una aplicación con las rutas activadas y, además, crearemos dos componentes para poder probar. Para poder **fijar las rutas**, tenemos que dirigirnos al archivo ***AppRoutingModule*** e incluir lo siguiente:

```
const routes: Routes = [
 { path: 'first-component', component: FirstComponent
},
 { path: 'second-component', component:
SecondComponent },
];
```

Código 7. Fijar las rutas.

Así, cuando pongamos un enlace que se dirige a un componente, no hay que poner la URL completa, solo es necesario indicar lo que tengamos en el ***path*** de ese componente.

Por ejemplo, si queremos dirigirnos al primer componente en un enlace, incluiríamos: **'/first-component'**. De esta forma, evitamos utilizar rutas absolutas.

Audio 2. "Integración avanzada de componentes: Geolocalización"
http://bit.ly/2Ksld6O

Vídeo 2. "Rutas en Angular"
https://bit.ly/3oWizoD

/ 10. Formularios

La utilización de formularios es una parte **fundamental** de muchas aplicaciones en la actualidad, pues **se utilizan para permitir que los usuarios interactúen de manera general con el sistema**.

Angular ofrece diferentes enfoques para manejar formularios: reactivo y basado en plantillas. El procesamiento de los datos de ambos tipos de formularios es diferente por lo que tendremos que utilizar el que más convenga dependiendo de la situación. Generalmente, los **formularios reactivos** son más escalables, reutilizables y comprobables. Los **formularios basados en plantillas** son útiles para poder añadir un formulario simple a una aplicación como, por ejemplo, un formulario de registro.

Los formularios reactivos se basan en modelos para manipular las entradas del formulario cuyos valores pueden cambiar a lo largo del tiempo. Hay que considerar t**res pasos para usar los controles de un formulario**:

1. Registrar correctamente el módulo del formulario.
2. Crear una instancia del control de formulario en el correspondiente componente.
3. Registrar el control de formulario en la plantilla.

Es **importante** tener en cuenta que los pasos anteriores no contemplan la vista que, además, hay que crear para mostrar el formulario.

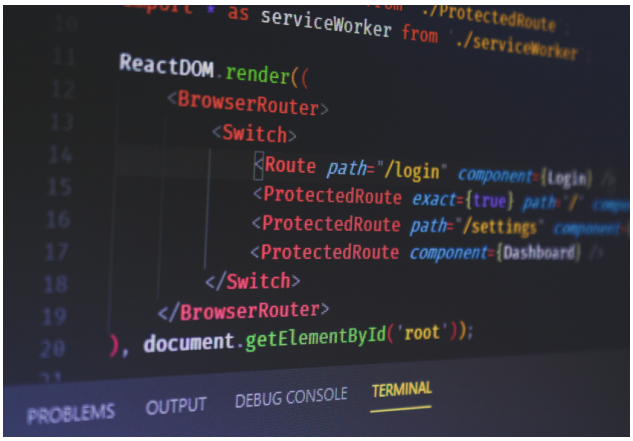

Fig. 8. Angular introduce formularios reactivos.

/ 11. Caso práctico 2: "Vinculación de la vista y el modelo"

Planteamiento: En los formularios que introducimos en nuestra aplicación Angular, observamos que tenemos que acceder al DOM constantemente para obtener los resultados del formulario, no obteniendo los datos de manera automática de la vista.

Nudo: ¿Es normal este funcionamiento?

Desenlace: Como hemos visto en el apartado sobre formularios reactivos en Angular, este tipo de formularios permite una vinculación dinámica de datos.

Gracias a este funcionamiento, no es necesario que se consulte constantemente el DOM para poder manejarlo. Esto se traduce en que el DOM actualiza directamente el modelo desde la vista.

El funcionamiento que tenemos, en nuestro caso, no es correcto y puede estar causado por una vinculación errónea entre la vista y el modelo.

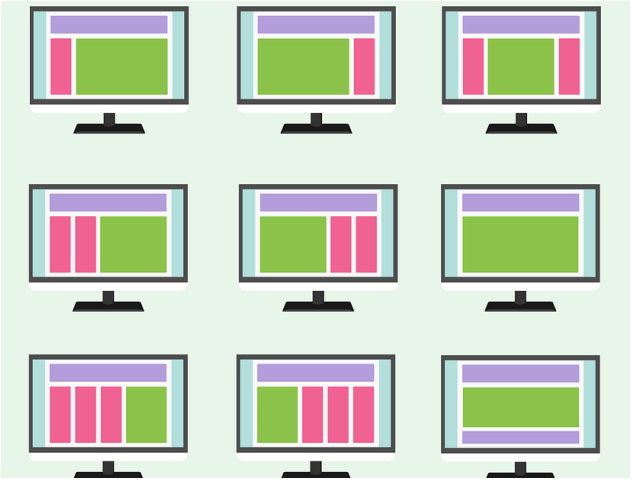

Fig. 9. Cuidado con las vinculaciones necesarias, ya que afectarán a la presentación de los datos.

/ 12. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos presentado **Angular** y sus principales funcionalidades para poder crear sitios web de una única página. Gracias a esta funcionalidad, el usuario puede permanecer todo el tiempo en una única URL y transitar a una vista u otra dependiendo de la interacción que realice. Para el desarrollador, este tipo de páginas ofrece la posibilidad de introducir mayor dinamismo, pues es más sencillo realizar cambios en las rutas de las páginas.

Como hemos visto en el tema, la **creación de componentes** en Angular requiere utilizar patrones de diseños como el decorador. Asimismo, un componente puede estar vinculado a una vista, existiendo un enlace dinámico bidireccional entre la vista y el componente. Gracias a esta forma de funcionamiento, se introduce una nueva forma de controlar el DOM. Además, es posible realizar páginas web dinámicas sin tener que generar código de JavaScript, ya que la vista es la única responsable de generar su visualización.

Resolución del caso práctico inicial

Como hemos visto a lo largo del tema, cuando utilizamos Angular para el desarrollo de páginas web *single-page*, es posible ir añadiendo diferentes *frameworks* para facilitar el desarrollo.

A través de estas librerías, podemos aumentar el número de funcionalidades de nuestro sitio web con poco esfuerzo.

Al utilizar Angular, el lenguaje nativo es TypeScript, sin embargo, podemos seguir utilizando todas las librerías y código JavaScript que tengamos o creemos de nuevas. También es posible diseñar la página utilizando CSS y HTML.

Una de las ventajas que tenemos con Angular es la posibilidad de utilizar *binding* de datos con unas comunicaciones más sencillas de gestionar e implementar.

Fig. 10. Angular admite diversas tecnologías.

/ 13. Bibliografía

Mohammed, Z. (2019). *Angular projects: build nine real-world applications from scratch using Angular 8 and TypeScript*. Birmingham, UK: Packt Publishing.

Sanctis, V. (2020). *ASP.NET Core 3 and Angular 9*. Third Edition. City: Packt Publishing.