



Capítulo 1: JavaScript

Capítulo 2: Angular 8


CIBERTEC

Curso: Angular 8.0 Front - End Application Developer

Capítulo Nº 2 Angular 8



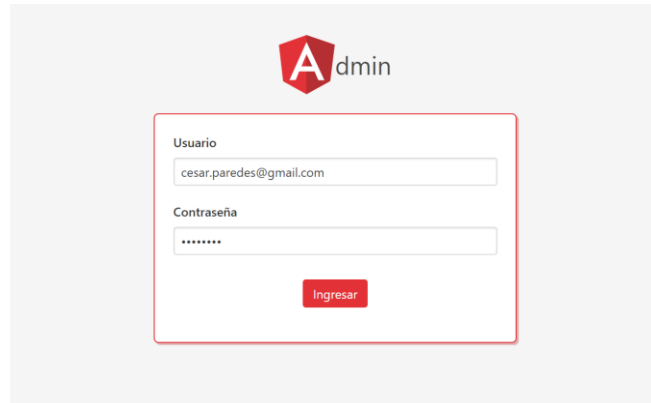
Objetivos

Al finalizar el capítulo, el alumno logrará:

- Reconocer la arquitectura de Angular 8.
- Ejecutar las mejores prácticas con Angular 8.
- Ejecutar las pruebas unitarias de servicios y componentes a aplicaciones con Angular 8.



Angular



[Demo](#) - [Usuarios disponibles](#)

Agenda

- Tema 1: Introducción a Angular 8
- Tema 2: Introducción a TypeScript
- Tema 3: Configuración de entorno
- Tema 4: Módulos
- Tema 5: Componentes
- Tema 6: Ciclo de vida del componente
- Tema 7: Templates, Directivas y Pipes
- Tema 8: Introducción a RxJS
- Tema 9: Servicios y peticiones HTTP
- Tema 10: Rutas
- Tema 11: Lazy Loading
- Tema 12: Formularios: Template-driven y Model-driven
- Tema 13: Unit Testing

Introducción a Angular 8

- Angular 8 es el **framework JavaScript estándar** para crear webs **SPA**, que es mantenido por el equipo de Google.
- **Angular 8 ha dado un salto de calidad** con respecto a la versiones anteriores del framework, como AngularJS, ahora utilizaremos **TypeScript** para definir nuestras clases, propiedades y métodos; el código es mucho más limpio y se ha mejorado el funcionamiento general del framework.
- Los components, son una parte importante de una aplicación web Angular.

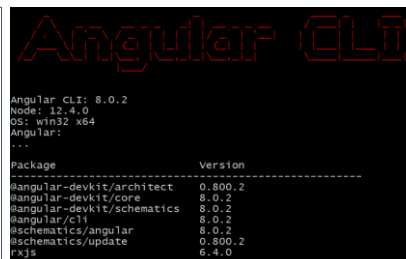


Introducción a Angular 8

- Angular 8, tiene un CLI, una herramienta que nos permite agilizar el desarrollo de aplicaciones con este framework, utilizando la línea de comandos.

```
# To install Angular CLI
npm install -g @angular/cli

# To check if It was installed correctly
ng --version
```



```
Angular CLI: 8.0.2
Node: 12.4.0
OS: win32 x64
Angular:
...
Package                         Version
-----
angular-devkit/architect        0.800.2
angular-devkit/core             8.0.2
angular-devkit/schematics       8.0.2
angular/cli                     8.0.2
@schematics/angular             8.0.2
@schematics/update              0.800.2
rxjs                             6.4.0
```



Introducción a TypeScript

- Es un lenguaje Open Source que extiende el lenguaje JS.
- TS fue creado por Microsoft.
- TS compila a Javascript.
- TS trabaja bajo el paradigma de POO basado C#.
- El dominio de TypeScript se usa tanto del lado del servidor como del lado del cliente.

```
# To install typescript
npm install -g typescript

# To compile a file to JS
tsc *.ts
```



Configuración de entorno

- Creando un proyecto usando el angular CLI.

```
# Create a new project
ng new angular-app

# Enter the created folder and execute
cd angular-app
ng serve -o
```



Configuración de entorno

- **angular-cli.json:** configuración del propio CLI.
- **package.json:** dependencias de librerías y scripts.
- **src/:** la carpeta donde están los siguientes archivos fuentes:
 - **index.html:** un fichero HTML índice estándar.
 - **main.ts:** fichero TypeScript de arranque de la aplicación.
 - **app/:** la carpeta con el código específico de tu aplicación:
 - **app.module.ts:** las aplicaciones son árboles de módulos y éste es su raíz.
 - **app.component.ts:** las páginas son árboles de componentes y éste es su raíz.

Módulos

- Los módulos son **contenedores en donde se almacenan los componentes y servicios** de una aplicación. En Angular, cada programa se puede ver como un árbol de módulos jerárquico. A partir de un módulo raíz, se enlazan otros módulos en un proceso llamado, **importación**.
- Antes de importar cualquier módulo hay que definirlo.
- En Angular, **los módulos se declaran como clases de TypeScript**, decoradas con la función `@NgModule()` que recibe un objeto como único argumento.
- En las propiedades de ese objeto es donde se configura el módulo.

Módulos

- Cuando se crea un proyecto usando el Angular CLI, el módulo `app.module.ts` es creado por defecto.

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

import { AppComponent } from './app.component';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule ],
  providers: [],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```



Módulos

- El módulo **App** también se conoce como **módulo raíz**, porque de él surgen las demás ramas que conforman una aplicación.
- **@NgModule** contiene un objeto con las propiedades `declarations`, `import`, `providers` y `Bootstrap`:
 - **Declaration:** Esto incluirá los componentes creados.
 - **Import:** Es un listado de módulos necesarios para ser utilizados en la aplicación.
 - **Providers:** Esto incluirá los servicios creados.
 - **Bootstrap:** Esto incluye el componente principal de la aplicación para comenzar la ejecución.



Componentes

- Los módulos son contenedores. Lo primero que vamos a guardar en ellos, serán componentes.
- Los componentes son los bloques básicos de construcción de las páginas web en Angular.
- Contienen una parte visual en html (View) y una funcional en Typescript (Controller).

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

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



Componentes

- Los componentes, como el resto de artefactos en Angular, serán **clases TypeScript decoradas** con funciones específicas.
- Para los componentes, el decorador es **@Component** que recibe un objeto de definición de componente.
- Los componentes definen nuevas etiquetas HTML.
- El nombre de la nueva etiqueta se conoce como **selector**.

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



Componentes

- **Template:** de forma simplificada, o cuando tiene poco contenido, puede escribirse, directamente, en la propiedad `template` del objeto decorador; pero es más frecuente encontrar la plantilla en su propio archivo `html` y referenciarla como una ruta relativa en la propiedad `templateUrl`.
- La propiedad **`styles`** y su gemela **`stylesUrl`** permiten asignar **estilos CSS, SASS o LESS** al componente. Estos estilos se incrustan durante la compilación en los nodos del *DOM* generado. Son exclusivos del componente y facilitan el desarrollo granular de aplicaciones.

Componentes

- En la clase del componente, normalmente, expone propiedades y métodos para ser consumidos e invocados de forma declarativa desde la vista.
- Una aplicación web en Angular se monta como un árbol de componentes. El componente raíz ya viene creado, por defecto, con CLI de angular.

```
# To create a new component
ng generate component header

# To see the options of this command
ng generate component --help
```


Ejercicio Nº 2.1: Crear una aplicación usando Angular CLI

Al finalizar el laboratorio, podrás:

- Crear una aplicación Angular 8 usando el CLI de angular.

Ciclo de vida del componente

- Mediante la implementación de unas interfaces, podemos ejecutar funcionalidad en determinado punto del ciclo de vida de un componente.

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

@Component({...})
export class AppComponent implements OnInit, OnDestroy {
  constructor() {}

  ngOnInit() {
    console.log('Component Init');
  }

  ngOnDestroy() {
    console.log('Component Destroy');
  }
}
```

Templates, Directivas y Pipes

- Para interpolar data del componente, en el template se usa las doble llaves “`{{ attr }}`”.
- Para modelar la data que se usará en el componente, se usan clases TypeScript.
- Para iterar y mostrar los elementos de una lista, usamos la directiva ***ngFor***, anteponiendo un “`*`”.
- Para condicionar qué contenido vamos a mostrar, usamos la directiva ***ngIf***, anteponiendo un “`*`”.
- Para crear una directiva se usa el decorador ***Directive***.
- Para crear un pipe se usa el decorador ***Pipe***.



Templates, Directivas y Pipes

- **Directiva:** las directivas en angular, siempre ha sido una de las formas más fácil de usar pequeños fragmentos de código. Son pequeñas sentencias, que se anexan al html y éste tiene su función nativa en javascript. Las directivas tienen 3 tipos:
 - Componentes.
 - Directivas estructurales.
 - Directivas Atributo.
- **Pipe:** son funciones que aceptan un valor de entrada y retorna dicho valor transformado.



Templates, Directivas y Pipes

```
<h2>
  {{ title | uppercase }}
</h2>

<div class="list" *ngIf="items.length else noItems">
  <ul>
    <li *ngFor="let item of items">
      {{ item }}
    </li>
  </ul>
</div>

<ng-template #noItems>
  There are no items
</ng-template>
```

Ejercicio Nº 2.2: Elaborar componentes, templates, directivas y pipes

Al finalizar el laboratorio, podrás:

- Elaborar componentes, templates, directivas y pipes.

Introducción a RxJS

- **La programación Reactiva** está orientada al manejo de streams de datos asíncronos y la propagación del cambio.
- Un stream es un flujo de datos.
- Manejo del patron Observer.



Reactive X
RxJS



Servicios y Peticiones HTTP

- En Angular, la presentación es cosa de los componentes. La lógica y los datos tienen su lugar en servicios compartidos.
- Como casi todo en Angular, los servicios son clases TypeScript.
- Su propósito es contener lógica de negocio, clases para acceso a datos o utilidades de infraestructura.
- Estas clases son perfectamente instanciables desde cualquier otro fichero que las importe; pero, Angular nos sugiere y facilita que usemos su sistema de inyección de dependencias (DI).



Servicios y Peticiones HTTP

- La particularidad de las clases de servicios está en su decorador: ***Injectable***.
- Los servicios se deben registrar como un proveedor en algún módulo.

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

@Injectable()
export class ProductService {

  constructor() {

  }

}
```



25

Servicios y Peticiones HTTP

- La librería @angular/common/http trae el módulo HttpClientModule, con el servicio inyectable HttpClient, que se debe declarar como dependencia en sus propios constructores.
- Para cada verbo ***http***, tenemos su método en el servicio HttpClient.



26

Ejercicio Nº 2.3: Ejecutar el consumo de datos desde servicios externos

Al finalizar el laboratorio, el alumno logrará:

- Comunicar una aplicación Angular 8 con una API REST.



Rutas

- Básicamente, significa, navegación entre páginas.
- Para ello, Angular nos provee el módulo ***RouterModule***.
- Mediante las rutas, podemos restringir el acceso de los usuarios a ciertas rutas.
- Podemos navegar entre páginas, sin refrescar el navegador.

```
<router-outlet></router-outlet>
```



Ejercicio Nº 2.4: Implementar rutas y configurar Lazy Loading

Al finalizar el laboratorio, podrás:

- Agregar distintas rutas a una aplicación Angular 8.
- Configurar Lazy loading, para cargar módulos a demanda.



Formularios Template-driven y Model-driven

- Los formularios son el punto de entrada de información a nuestros sistemas. Llevan con nosotros, desde el inicio de la propia informática, y se han comido una buena parte del tiempo de programación. En *Angular*, han prestado una atención a ellos, facilitando su desarrollo, desde pantallas simples a complejos procesos.
- Hay dos formas de trabajar con formularios en Angular:
 - **Template-driven:** todo el trabajo y lógica es en el **html**.
 - **Model-driven:** todo el trabajo y lógica es en la **clase**.



Ejercicio Nº 2.5: Elaborar formularios y manejo de validaciones

Al finalizar el laboratorio, podrás:

- Agregar formularios y validaciones en Angular 8.
- Identificar las dos formas de trabajar con formularios.



Unit testing para servicios y componentes

- El angular CLI instala toda las dependencias para trabajar con testing usando el framework Jasmine.
- **Jasmine** es framework para hacer testing a código Javascript.
- **Karma** es un corredor de tests creado por Angular.
- Para correr los tests se debe ejecutar lo siguiente:

```
ng test
```



Unit testing para servicios y componentes

- Ejemplo de tests con Jasmine.

```
function greet() {  
  return 'hello';  
}  
  
describe('Hello', () => {  
  if('Says hello', () => {  
    expect(greet()).toEqual('hello');  
  })  
});
```

Ejercicio Nº 2.6: Unit testing para servicios y componentes

Al finalizar el laboratorio, podrás:

- Hacer unit testing a componentes y servicios de Angular 8.

Lecturas adicionales

Para obtener información adicional, puede consultar:

- Angular 8
 - [Angular.io](https://angular.io)
 - [Angular posts](#)
 - [Angular 8: Upgrading & Summary of New Features](#)
 - [Learn RxJS](#)



Resumen

En este capítulo, aprendiste:

- El uso del framework Angular 8 para la capa de presentación de sus proyectos web. En este capítulo, se implementó un CRUD de productos, aplicando los conceptos core del framework Angular 8.



Tarea Nº 2: Implementar un carrito de compras

- Crear un proyecto Angular 8.
- Integrar un proyecto Angular 8 con el backend mediante una API

