Índice

1.Introducción

Angular es un framework para el desarrollo de aplicaciones web en la parte del cliente. Está basado en el popular AngularJS (versiones 1.x), pero ha cambiado mucho con respecto a su antecesor. Dos mejoras significativas son: un rendimiento muy superior, y una API simplificada con menos conceptos que aprender.

Angular está desarrollado en TypeScript, y es el lenguaje recomendado para construir aplicaciones con este framework. Este lenguaje es básicamente JavaScript con algunas características adicionales y el tipado de variables y funciones. Esto mejora mucho la depuración de aplicaciones en tiempo de desarrollo, el autocompletado por parte del editor (intellisense), etc. Al pasar por un proceso de compilación a JavaScript, el resultado será código compatible con todos los navegadores actuales.

Algunas de las principales características de Angular son:

- Introduce expresividad en el código HTML a través de la interpolación de variables, data-binding, directivas, etc.
- Tiene un diseño modular. Sólo se importan las características que necesitamos en la aplicación (mejora de tamaño y rendimiento). Además, permite separar nuestra aplicación en módulos de forma que el navegador vaya cargando en cada momento lo que necesita (lazy loading) → Mejora de tiempos de carga.
- Es fácil crear componentes reutilizables para la aplicación actual u otras.
- La integración con un servidor de backend basado en servicios web es muy sencilla.
- Permite ejecutar Angular en el lado del servidor para generar contenido estático que puedan indexar los motores de búsqueda (incapaces de ejecutar ellos Angular en el cliente). Esta característica se llama Angular Universal.
- Potentes herramientas de desarrollo y depuración: TypeScript, Augury (plugin de Chrome), frameworks de pruebas como Karma o Jasmine, etc.
- Integración con frameworks de diseño como Bootstrap, Angular Material, Ionic…
- En Angular, creamos aplicaciones de una página (SPA), donde la página principal se carga entera sólo una vez → Mejor rendimiento.
- Debido a su naturaleza de framework completo y modular, es la solución más efectiva para desarrollar grandes aplicaciones.

En resumen, vamos a aprender un framework completo y muy popular, desarrollado y usado por Google, y con unas mejoras significativas con respecto a la primera versión orientadas a facilitar el desarrollo de grandes aplicaciones.

2. Creación de un proyecto Angular

Aunque varios editores (Atom, Webstorm, etc.) se integran bien con Angular por medio de extensiones, en este curso recomiendo utilizar Visual Studio Code. Es un editor/IDE de código abierto que se integra muy bien con este framework, y es el que utilizaremos para ejemplos, resolver dudas, etc.

2.1. Creando un proyecto con Angular CLI

Para gestionar nuestros proyectos de Angular, vamos a usar la herramienta **Angular CLI**. Esta herramienta de línea de comandos permite generar un proyecto, componentes para el mismo, compilarlo y ejecutarlo con diferentes configuraciones (desarrollo, producción, …), lanzar baterías de pruebas automatizadas, etc.

Para instalar Angular CLI, debemos tener instalada la herramienta Node Package Manager (NPM), que viene incluida con NodeJS (Instalad siempre la versión LTS). Una vez cumplido esto, ejecutamos (como administrador):

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

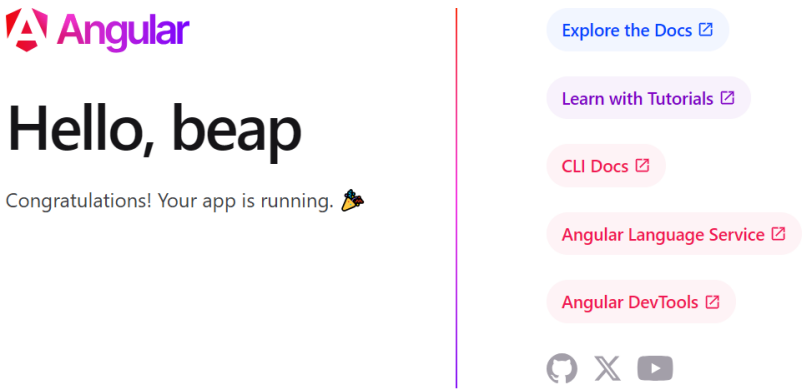
Podemos comprobar que todo ha ido bien con el siguiente comando:

```
ng --version
```

Una vez instalada la herramienta, creamos un proyecto Angular (las preguntas podéis contestarlas por defecto dándole al enter):

```
ng new NOMBRE-PROYECTO
```

Esto nos creará un directorio con el nombre del proyecto establecido. Arrástralo a visual studio code y para comprobar que funciona correctamente, entramos en el directorio y ejecutamos el comando **ng serve**. Esto lanzará un servidor web con nuestra aplicación. Entra en http://localhost:4200/ para verlo.



Esto sale creado por defecto en el archivo **app.component.html**

2.2. Como funciona Angular CLI

Angular CLI es una herramienta para crear y gestionar proyectos Angular. Permite automatizar tareas como la compilación, pruebas, generación de código, ejecución, etc. Lo que permite centrarnos en programar. Para ello se sirve de herramientas como Webpack, Jasmine, Karma, etc.

A lo largo de este curso explicaremos las diferentes partes que componen una aplicación Angular. Podemos generar un esqueleto para la mayoría mediante Angular CLI. Simplemente desde el directorio (o un subdirectorio) del proyecto Angular, ejecutamos el comando **ng generate** (**ng g**).

Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-new-module
Guard	ng g guard my-new-guard

2.3. Compilando el proyecto y generando el bundle (paquete)

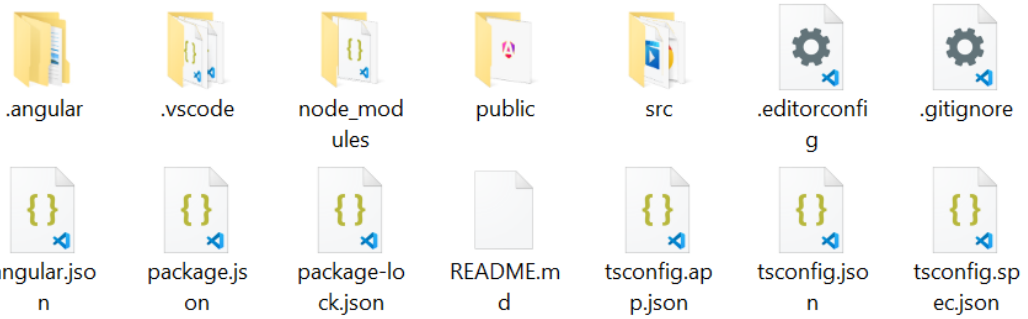
Subir todo el proyecto tal cual lo tenemos a un servidor web es innecesario y lento (mira el tamaño del directorio del proyecto). Esto ocurre porque hay muchas dependencias y archivos que instala NPM, la mayoría de las cuales son herramientas para el desarrollo, testing, y compilación que no necesitamos para una aplicación en producción, sólo para desarrollar.

Cuando compilamos la aplicación, Angular CLI usará WebPack para empaquetar todos nuestros archivos CSS, JavaScript, etc. en unos pocos archivos (minificados), resolviendo las dependencias entre las diferentes partes del código y dejando sólo lo estrictamente necesario para ejecutar la aplicación. Estos archivos, listos para subir al servidor web, se encuentran el directorio **dist/**.

El comando para compilar y empaquetar nuestra aplicación es **ng build**. Por defecto, construye a aplicación en modo desarrollo (pesa más y contiene información de depuración). Para compilar en producción se debe ejecutar **ng build –prod**. Esta opción generará un código más pequeño, sin información de depuración.

2.4. Analizando la estructura del proyecto

Vamos a analizar un poco qué directorios y archivos se crean cuando generamos un nuevo proyecto Angular.



En el directorio principal del proyecto veremos archivos de configuración para **NPM** (package.json), **Angular CLI** (angular.json), el compilador de **TypeScript** (tsconfig.json). Estos archivos son todos configurables a gusto del desarrollador o equipo de desarrollo.

También encontramos estos subdirectorios importantes:

- **node_modules**: Dependencias de la aplicación instaladas con NPM.
- **src**: Donde está nuestra aplicación

2.5. Carga inicial de la aplicación

¿Cómo sabe Angular, en el caso de haber varios componentes, cual el principal y donde la aplicación se inicia? Esto se puede ver en el archivo **src/main.ts**:

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

Vemos que AppComponent es el componente principal de esta aplicación, el resto de componentes irán dentro.

Al crear el proyecto, por defecto te ha creado la página inicial de Angular. Borra el html de ejemplo de **app.component.html** y copia el siguiente como inicial:

```
<div style="text-align: center">
  <h1>Welcome to {{ title }}!</h1>
</div>
```

3. Componentes

Cuando creamos el proyecto, se creó un componente principal de la aplicación (el resto van dentro) en **src/app/app.component.ts**. Vamos a analizarlo:

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

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

- **imports** → En este array añadimos módulos externos (de Angular o creados por nosotros). Los elementos exportados por esos módulos (componentes, etc.) serán accesibles por el módulo actual.
  - **Component**: Esto importa el decorador @Component. Los decoradores son funciones (equivalentes a las anotaciones de Java, por ejemplo), que especifican metadatos que describen una clase o un método. Estos metadatos son interpretados por Angular en este caso para saber qué tipo de objeto del framework (componente, servicio, módulo, filtro, etc.) representa la clase implementada.
  - **RouterOutlet**: se usa para permitir que se muestren componentes dependiendo de la ruta seleccionada en la aplicación.
- **selector** → Este es el nombre de la etiqueta que se utilizará en la vista (HTML). Mira en **src/index.html** donde verás la etiqueta **<app-root>** ya creada (ahí es donde se carga la aplicación). Cuando Angular se carga, reemplazará el contenido de esa etiqueta por la plantilla del componente.
  - Por defecto, todos los componentes se crean con un selector que tiene el prefijo **app**. Este prefijo sería mejor que tuviera las iniciales de nuestro proyecto (o fuese más descriptivo). Esto se puede configurar al crear el proyecto → **ng new mi-proyecto --prefix mp**. Posteriormente también se puede en el archivo **angular.json** (busca el atributo "**prefix**").
- **templateUrl** → Este archivo representa la vista asociada al componente. El HTML que se cargará dentro del selector cuando esté todo cargado.
- **styleUrls** → Angular permite asignar una o varias hojas de estilo CSS (o SASS o LESS usando la propiedad --style al crear un proyecto) a un componente. Estos estilos se aplicarán **sólo** al componente asociado.
  - Los estilos generales para toda la aplicación se definen en **src/styles.css**.

- **exports** → En este array añadiremos los componentes, directivas, etc. del módulo actual (declarations) que queramos exportar. Es decir, cuando otro módulo importe el actual, qué cosas podrá usar.

    - La clase AppComponent tiene una propiedad declarada llamada **title**. Esta propiedad es pública (por defecto) y de tipo string (en una asignación TypeScript asigna el tipo del valor asignado a la variable, no hace falta especificarlo).

      Si observas la plantilla (**src/app/app.component.html**), verás algo como esto:

```
<div style="text-align: center">
  <h1>Welcome to {{ title }}!</h1>
</div>
```

Esto se llama **interpolación** (lo veremos en la siguiente sección). Esto significa que en la plantilla {{title}} está vinculada a la propiedad title de la clase AppComponent. Cuando el componente se cargue, el valor se sustituirá y aparecerá el texto "**Welcome to angu**". Cambia el texto de la propiedad "title" y verás el efecto.