Angular (Introducción).

Desarrollo Web en Entorno Cliente.

Profesor: Juan José Gallego García

Índice :

Introducción

Angular, (o también llamado Angular 2, no confundir con el primer framework desarrollado, AngularJS) es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página (SPA). Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC) o según algunos autores más correctamente Modelo-Vista-Modelo de Vista (MVVM) por algunas de sus características que veremos más adelante relacionada con la actualización entre vistas y datos.

Para entender cómo trabaja Angular veremos en primer lugar un ejemplo básico realizando todos los pasos necesarios desde la instalación del framework, hasta comprobar el funcionamiento de la aplicación una vez desplegada. En los siguientes apartados explicaremos poco a poco los elementos que forman un proyecto de aplicación y cómo se relacionan entre ellos.

Ejemplo ‘Hola mundo’

1. Instalamos el CLI para angular `npm install -g @angular/cli`
2. Comprobamos la versión instalada `ng version` (para últimas versiones)
3. En algún directorio creamos el espacio de trabajo y generamos el esqueleto de una aplicación base ejecutando : `ng new holaMundo --no-standalone`
4. Aceptamos todo por defecto (pausa para aclarar SSR y SSG).
5. Cambiamos al directorio que se ha creado con el nombre de la aplicación: `cd holaMundo`
6. Probamos la aplicación generada, con `ng serve -o` ‘**-o**’ para iniciar automáticamente el servidor en el navegador, http://localhost:4200/
7. Solo nos queda compilar y desplegarlo en un servidor (Apache p.e. XAMP). Si queremos compilar para desplegarlo en el directorio raíz del servidor ejecutamos en consola: `ng build`, pero si queremos compilarlo para ->

alojarlo en algún directorio, por ejemplo en la ruta : http://localhost/holaAngular/ , ejecutamos `ng build --base-href=/holaAngular/` ó `ng build --base-href=''` (ruta relativa).

8. Finalmente copiamos dentro del directorio elegido en el servidor, el contenido generado en **/dist/hola-mundo/browser** tras la compilación. **IMPORTANTE**, para un despliegue correcto, que no haya problemas de enrutamiento, en el caso del servidor (**apache**) y **PHP**, usaremos la primera forma de **build** (con directorio) y añadiremos el siguiente archivo al directorio del servidor .
   Para **NodeJS** se podría arreglar activando el **hash** en el módulo de routing de Angular `RouterModule.forRoot(routes, { useHash: true })`pero es más correcto de la siguiente forma: primero haremos igual que para apache, compilamos con **build** (con directorio) y segundo, redireccionamos todas las peticiones de rutas que no vayan dirigidas al servidor hacia el archivo INDEX.HTML de Angular. Ejemplo ServerNodeJS

Podemos comprobar que nuestro proyecto base funciona . A continuación haremos un estudio de los pasos que se han dado, es decir, qué archivos se han ido generando, en que orden se ejecutan (punto de entrada a la aplicación), y cómo está estructurado un proyecto Angular para saber en qué directorios hay que insertar nuestro código a implementar.

Estructura de directorios de un proyecto Angular

Hemos visto que al crear el nuevo proyecto con ng new ..., nos ha creado un directorio con el nombre de la aplicación con el siguiente contenido:

- Primero en el directorio raíz una serie de archivos de los cuales destacar, package.json, que contiene entre otros las dependencias del proyecto (librerías de módulos), tsconfig.json , configuración para el compilador de TS, angular.json , contiene los parámetros de configuración para el proyecto de Angular actual.
- Directorio /dist contiene el directorio en el cual están los archivos sintetizados para producción generados durante la compilación.
- Directorio /e2e para el desarrollo de las pruebas. Viene de "end to end" testing (a partir de Angular 12 no se crea por defecto).
- Directorios /node_modules que ya conocemos para contener las librerías instaladas y mantenidas por **npm**.

- Directorio /src es el directorio principal que contiene la estructura de los archivos y directorios de nuestra aplicación en tiempo de desarrollo :
  a. En el directorio raíz entre otros archivos de estilos, testeo e icono para la web (**favicon.ico**)(en las últimas versiones se ha pasado al directorio **public** que veremos a continuación), tenemos dos muy importantes, **index.html**, que es el único archivo html con el que va a funcionar nuestra aplicación, no tiene a penas código html, ya que será generado por los componentes que se le añadan dinámicamente junto a los archivos **.js** enlazados. Podemos ver que contiene en el **body** una etiqueta personalizada **<app-root>** que hace referencia a un componente web que más adelante estudiaremos.
     Por otro lado, está el archivo **main.ts** , escrito en TypeScript como los demás archivos de código, es el punto de entrada en tiempo de desarrollo durante la ejecución, dispara el módulo principal de entre todos los módulos que contiene una aplicación típica de Angular.
  b. Directorio /src/app contiene los archivos de módulos (**--.module.ts**) que enlazan con su propio conjunto de componentes, los cuales constituyen la lógica de negocio junto a las vistas de la aplicación .

c. Directorio /src/assets (en las versiones más recientes ha pasado al directorio /public) contiene los archivos estáticos del proyecto, imágenes, archivos PDF, y otros elementos. Estos archivos también se moverán a "**dist**" para que estén disponibles en la aplicación una vez subida al servidor web desde donde se va a acceder.
d. Directorio /src/environments, contiene opciones de configuración de compilación para entornos de destino particulares. Por defecto, hay un entorno de desarrollo estándar sin nombre y un entorno de producción ("prod"). Se puede definir configuraciones de entorno de destino adicionales (no se crea por defecto en las últimas versiones).

Antes de seguir los pasos exactos de ejecución de la aplicación, veremos en el siguiente apartado cómo se estructura una aplicación Angular a nivel de unidades funcionales (módulos, componentes, etc..)

Estructura funcional de una aplicación Angular.

La estructura funcional de una aplicación Angular se basa en ciertos conceptos fundamentales. Los bloques de construcción básicos son los **NgModules** (**módulos**), que proporcionan un contexto de compilación para los **componentes**, es decir estos módulos contienen/enlazan con un conjunto de componentes. NgModules recopila código relacionado en conjuntos funcionales; Una aplicación angular se define mediante un conjunto de NgModules. Una aplicación siempre tiene al menos un módulo raíz que permite el arranque y, por lo general, tiene más módulos con otras funcionalidades.

Los **componentes** definen vistas (plantilla HTML+ CSS), que son conjuntos de elementos de pantalla que Angular puede elegir y modificar de acuerdo con la lógica (a través de **directivas**) y los datos de su programa (a través de **enlace de datos**).

Los componentes usan **servicios** , que proporcionan una funcionalidad específica que no está directamente relacionada con las vistas. Los proveedores de servicios pueden inyectarse en los componentes como dependencias (**ID**) , haciendo que su código sea modular, reutilizable y eficiente.

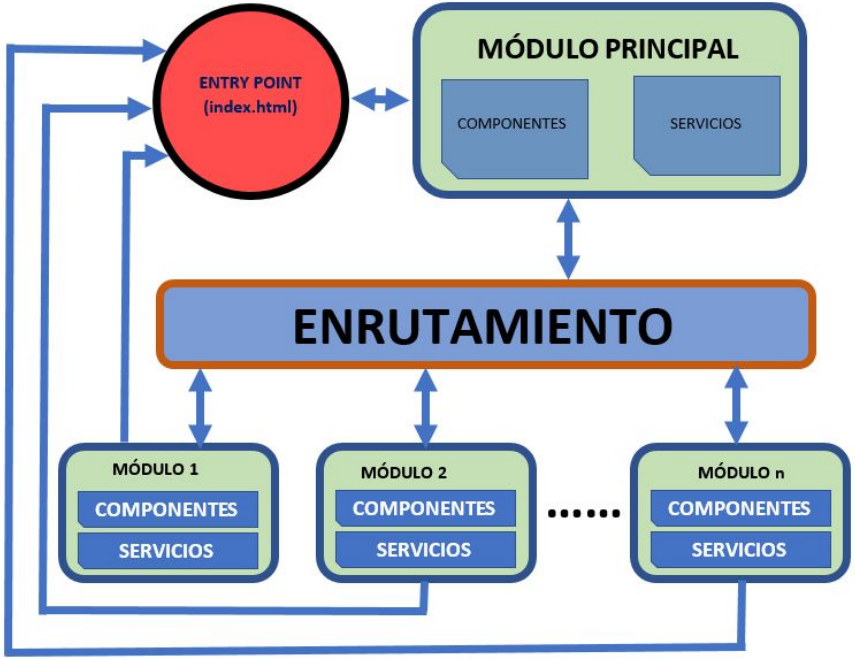
Tanto los componentes como los servicios son simplemente clases (en TS), con decoradores que marcan su tipo y proporcionan metadatos que le indican a Angular cómo usarlos.

Los componentes de una aplicación suelen definir muchas vistas, ordenadas jerárquicamente. Angular proporciona el servicio de enrutamiento (Router) para ayudar a definir rutas de navegación entre vistas. El enrutador proporciona capacidades de navegación sofisticadas en el navegador.

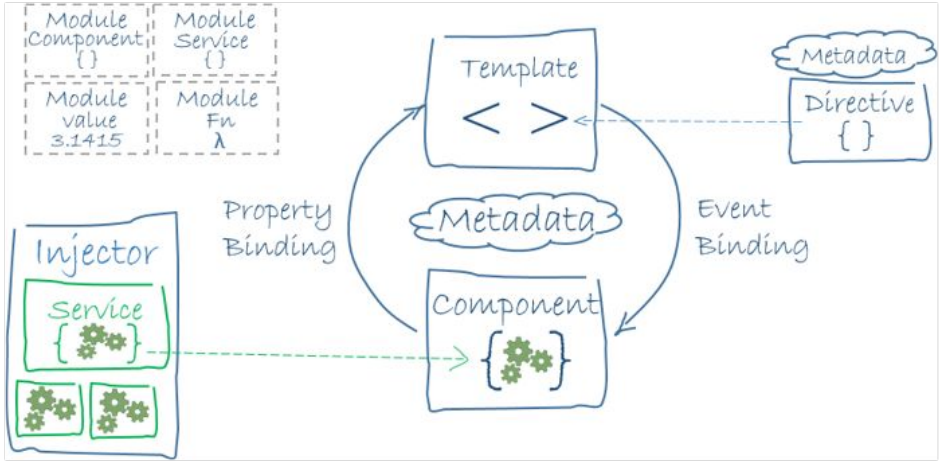
Estamos comprobando que la estructura de una aplicación en Angular es bastante compleja. Es normal, si tenemos en cuenta todas las funcionalidades que implementa el framework, y que se necesita practicar mucho para controlarlo. No obstante, una vez entendido la estructura que sigue y a pesar de esa complejidad, siempre se mantiene el mismo principio para cualquier tipo de aplicación.

A continuación se muestran dos esquemas de cómo se interrelacionan los elementos en una aplicación Angular, uno más global y otro un poco más específico.

Esquema 1



Esquema 2



Una vez visto resumidamente los elementos que intervienen en una aplicación Angular, vamos a seguir el flujo de ejecución de nuestra aplicación de ejemplo identificando cada elemento.

Flujo de ejecución :

1. En el index.html tenemos un componente raíz de la aplicación "**app-root**".
2. Al servir la aplicación `ng serve` , o al llevarla a producción `ng build` la herramienta **vite/esbuild** genera los paquetes (bundles) de código del proyecto y coloca los correspondientes scripts (JS) en el **index.html**, para que todo funcione.
3. El archivo por el que Webpack comienza a producir los bundles es el **main.ts**
4. Dentro del main.ts encontramos un import del módulo principal (**AppModule**) y la llamada al sistema de arranque (**bootstrapModule**) en la que pasamos por parámetro el módulo principal de la aplicación.
5. En el módulo principal se importa el componente raíz (**AppComponent**) y en el decorador **@NgModule** se indica que este componente forma parte del *bootstrap*.
6. El código del componente raíz tiene el selector "**app-root**", que es la etiqueta que aparecía en el index.html.
7. El template del componente raíz, contiene el HTML que se visualiza al poner en marcha la aplicación en el navegador relacionado con su CSS.

Resumen de conceptos en Angular

Módulo (NgModule): Agrupa componentes o servicios, existe un módulo principal denominado **AppModule**, un grupo de módulos forman una aplicación Angular.
Componente: Cada aplicación angular tiene al menos un componente, el componente raíz que conecta una jerarquía de componentes con el modelo de objeto de documento de página (DOM) con su nombre de selector (etiqueta-personalizada). Cada componente define una clase (TS) que contiene datos y lógica de la aplicación, y está asociada con una plantilla HTML y su CSS que forman la vista.
Servicio: Son unidades funcionales (clases) para ser inyectadas a otras clases por ejemplo de componentes, pero no ligadas a ninguna plantilla como éstos.
Plantilla: Combina HTML con lenguaje de marcado propio de Angular mediante **directivas** y **enlace de datos**.
Directivas: Enriquecen el lenguaje HTML mediante palabras claves que serán desplegadas en el proceso de compilación, por ejemplo: `<li *ngFor="let x of alumnos"></li>` , usa una instrucción iterativa para listar los nombres del array **alumnos**.

Enlace de datos: Enriquecen el lenguaje HTML a través de una conexión unidireccional o bidireccional entre los datos (valores de atributos en las clases) y elementos del DOM. (**Data Binding**)

- **enlace bidireccional:** al modificar el valor del **input** automáticamente se modifica el valor del atributo y viceversa. Ej: `<input [(ngModel)]="alumno.nombre">`
- **interpolación**: Despliega el valor en el elemento html Ej: `<li>{{coche.color}}</li>`
- **enlace de propiedad (Property bindind)**: Enlaza el valor de una propiedad de un componente al atributo de un elemento html , Ej: `<img [src]="propiedad_img_URL">`
- **enlace de evento (Event binding)**: Enlaza con un determinado controlador de evento ejecutando el método asociado. Ej: `<button (click)="nombreMetodo()">Pulsar!</button>`

Tubería (Pipe): Toman la salida de una expresión y la transforman. En el siguiente ejemplo la tubería definida como **date** da formato al valor de la variable **today** :

```
<p>La fecha es {{today | date}}</p>
```

Observables: Los observables brindan soporte para pasar mensajes entre editores y suscriptores en su aplicación (creados a partir de la librería **RxJs)**. Los observables ofrecen beneficios ->

significativos sobre otras técnicas para el manejo de eventos, la programación asincrónica y el manejo de múltiples valores. << *Aunque los enlaces de datos o eventos facilitan la actualización de datos en ambos sentidos, consumen demasiados recursos, en algunos casos pueden ser sustituidos por los* ***observables*** >>.
Los observables son declarativos, es decir, se define una función para publicar valores, pero no se ejecuta hasta que un consumidor se suscribe. El consumidor suscrito recibe notificaciones hasta que se completa la función, o hasta que se da de baja.

En este apartado sólo hemos visto una introducción a Angular, para ir ampliando conocimientos veremos distintos apartados de ampliación, e iremos haciendo ejemplos y prácticas con este framework.

Bibliografía

- https://angular.io/
- https://material.angular.io/
- https://desarrolloweb.com/manuales/manual-angular-2.html