IES Enrique Tierno Galván

Angular 5

Andrei García Cuadra

Contenido

1. BIS

Cabe destacar que vamos a utilizar Angular CLI para generar el proyecto, pero existen muchísimas otras alternativas para crearlo, como pueden ser WebPack, Ionic, etc.

La decisión de cuál usar varía, entre otras cosas, en el sistema donde vamos a desplegar nuestro sistema, pudiendo mezclar los anteriores (ya que son interoperables).

2. Introducción

1.1. ¿Qué es NPM?

Es un control de librerías para Node.js en su base. También se utiliza para numerosos proyectos de JavaScript, jQuery e incluso CSS. Su uso es sencillo: Incluir versiones de cierta librería y controlar sus versiones para evitar conflictos entre todas las librerías del proyecto. Asimismo, también es beneficioso para el trabajo en grupo o en varios equipos, ya que con un simple comando instala todas las dependencias, facilitando así la portabilidad del proyecto.

NPM requiere tener instalado Node.js en el equipo. Node.js es un framework de javascript de lado de servidor con múltiples utilidades, las cuales no trataremos en este proyecto.

1.2. Puesta a punto de NetBeans

1.2.1. Instalando Node.js

En primer lugar nos descargamos Node.js desde su página oficial. Instalamos el paquete de modo normal.

Una vez finalizada su instalación, ya en NetBeans, nos dirigimos a **Tools**->**Options**->**HTML/JS**->**Node.js** y en el apartado "**Node path**" introducimos la ruta en la cual acabamos de instalar Node.js. El fichero a buscar se llama **node.exe**.

Después, deberemos descargar el código fuente. Para ello, presionamos "**Download**", el cual está ubicado debajo de la línea de "Node Path".

1.2.2. Instalando NPM

En la pantalla anterior, buscamos la línea donde se indica "**npm path**". Ahí deberemos ubicar un archivo llamado "**npm.cmd**" que se debería ubicar en la misma carpeta que el Node.js.

1.2.3. Personalizando la instalación

Es recomendable a su modo marcar la opción "**Apply code changes on save**" para que nuestro proyecto se muestre en tiempo real compilado en el navegador.

1.2.4. Instalación de plugin TypeScript

Para poder trabajar con TypeScript, crear sus ficheros y visualizar su sintaxis... Deberemos instalar un plugin adicional en nuestro netbeans. Para ello, descargamos este archivo y, en netbeans, nos dirigimos a **Plugins** -> **Downloaded** -> **Add plugins** y nos dirigimos al fichero anteriormente descargado. Después, clickamos en la parte baja del panel el botón "**Install**". En el diálogo de instalación nos pedirá aceptar los términos y condiciones, y tras varios siguiente, reiniciar NetBeans.

2. Creando el proyecto Angular5

Para crear nuestro primer proyecto de Angular5, primero debemos crearlo desde la terminal y luego añadirlo a nuestro NetBeans. Es decir, primero usaremos la consola de comandos nativa de Angular y NPM para poder crear nuestro proyecto base y, posteriormente, agregaremos este proyecto ya creado a NetBeans.

2.1. Agregando NPM y NG al path de Windows

Esto será útil para poder usar los comandos **NPM** y **NG** en cualquier ubicación del sistema.

Dependiendo de nuestro sistema operativo estos pasos pueden variar, ya que están realizados sobre Windows 10.

Buscaremos "**Path**" en nuestro buscador de Windows, y pincharemos en "**Editar las variables de entorno del sistema**". En la nueva pestaña, clickamos "**Variables de entorno**" y en el nuevo clickamos "**PATH**". Pinchamos sobre "**Nuevo**" y agregamos la ruta de instalación de Node.js.

Aceptamos todos los diálogos, hasta quedarnos en el de "**Propiedades del sistema**" (el cual no cerraremos).

Abrimos un nuevo terminal de Windows e introducimos el siguiente comando:

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

Tras varios segundos de instalación, deberemos agregar al Path de Windows el terminal de Angular. Para ello, cerramos la ventana actual de terminal; Volvemos sobre el diálogo de "**Propiedades del sistema**", clickamos sobre "**Variables de entorno**" de nuevo, otra vez sobre "**Path**", pero esta vez agregaremos dos nuevas rutas:

IMPORTANTE: Reemplaza {TU_USUARIO} por tu nombre de usuario actual de Windows. De lo contrario, no funcionará.

```
C:\Users\{TU_USUARIO}\AppData\Roaming\npm\ng
```

```
C:\Users\{TU_USUARIO}\AppData\Roaming\npm
```

Tras esto, ya tendremos disponibles los comandos **NPM** y **NG** desde nuestros terminales de Windows (Nota: Se deben reiniciar los terminales que ya estaban abiertos).

2.2. Creando el primer proyecto de Angular5

Para crear nuestro primer proyecto usaremos el terminal. El comando ng es muy útil, y recordemos que lo hemos añadido usando el gestor de paquetes de **NPM**, por lo que nuestro comando se mantendrá actualizado siempre que hagamos un **npm install** global.

Crearemos nuestro primer proyecto en **C:** para evitar tener problemas de rutas. Para ello, abrimos un terminal de Windows e introducimos el comando

cd C:

Una vez en esta ruta, utilizaremos el siguiente comando para crear la aplicación (NOTA: my-app es el nombre de la app. Puede ser reemplazado, pero no se responsabiliza de que no puedas seguir correctamente el tutorial).

```
ng new my-app
```

Como podremos observar, nuestro proyecto de Angular está creando todos sus archivos. Cuando acabe, deberá salir algo parecido a:

```
added 1420 packages in 31.301s
Installed packages for tooling via npm.
Project 'my-app' successfully created.
```

¡Y ya tenemos nuestro proyecto creado!

2.3. Entendiendo los ficheros

Podemos explorar la carpeta **C:\my-app**, la cual contendrá varios archivos y directorios, entre ellos:

- **e2e**: Directorio generado por las pruebas unitarias.
- **node_modules**: Ubicación de nuestros módulos de NPM.
- **src**: Ubicación real de nuestro proyecto de Angular.
- **Ficheros karma/protractor**: Son ficheros relacionados con tests unitarios.
- **tsconfig.json**: Fichero de configuración para ES y JavaScript en general. Se recomienda poner el valor *compileOnSave* a *true* para cerciorarnos de que compilamos el proyecto cada vez que guardemos cualquier cambio.
- **.angular-cli.json**: Fichero de configuración propio de angular para nuestro proyecto.

2.4. Agregando el proyecto a NetBeans

¿Qué sería de tanto código sin un IDE? Para poder agregar nuestro proyecto a NetBeans, simplemente nos dirigimos al menú de NetBeans. Clickamos sobre **File** -> **New Project** -> **HTML5/JavaScript** -> **HTML5/JavaScript with existing sources**. Tras esto, al clickar **next**, nos solicitará la ubicación de nuestro proyecto, así que en **Site root** le indicamos que nuestro proyecto está sobre **C:\my-app**. Tras esto, clickamos finish, y ya se habrá añadido el proyecto a nuestra barra lateral izquierda de NetBeans.

Tras esto, debemos configurar nuestro proyecto para que se vea más uniforme en nuestro IDE. Daremos **click derecho al proyecto** -> **Properties**. En la pestaña "**Sources**" en el apartado "**Source Folder**" pincharemos el botón **browse**, y seleccionaremos la carpeta **src**.

En la pestaña "**npm**" podremos visualizar el listado de librerías utilizadas en el proyecto, con su versión, entorno (prod/dev) y si está correctamente instalada o no. De no ser así, se debería correr un **npm install**.

Volviendo al menú lateral, en la sección JavaScript, al desplegarla, nos encontraremos con la pestaña TypeScript. Al abrirla, marcamos la opción "**Compile on save**".

2.5. Personalizando comandos NPM

Al realizar un click derecho sobre nuestro proyecto, podemos observar dos apartados en la parte superior.

NPM install: Instala y renueva las dependencias del proyecto.

NPM Scripts > (Listado): Muestra una lista de comandos de NPM.

Haremos un NPM install para comprobar su funcionamiento.

Tras esto, viendo que nos funciona correctamente, modificaremos los comandos por defecto de NPM. Así, tendremos un menú sencillo, agradable y útil. Para ello, editamos el fichero **package.json** que se ubica dentro de **Site root**.

Dentro del fichero, buscaremos la sección **scripts**, y dentro de los corchetes, reemplazaremos todo el contenido por el siguiente:

```
"Iniciar servidor": "ng serve",
"Compilar": "ng build --prod"
```

De este modo, tras guardar el archivo, observaremos que la lista desplegable del menú NPM Scripts ha cambiado y nos muestra "**Iniciar servidor**" y "**Compilar**", que hacen lo propio.

2.6. Iniciando el servidor

Para iniciar nuestro servidor de angular, debemos clickar en "**Iniciar servidor**", que se ubicará en el menú anteriormente dicho. Una vez nos muestre el mensaje "**webpack: Compiled successfully**.", podremos acceder a nuestra web de angular desde el navegador utilizando la url: *http://localhost:4200*.

2.7. Usar comandos desde el terminal de Windows

Si no queremos agregar determinados comandos a NetBeans y deseamos ejecutarlos desde el terminal de Windows, simplemente debemos abrir un nuevo terminal, mover la ubicación actual a la carpeta principal del proyecto de Angular (en este caso **cd C:\my-app**) y allí podremos usar los comandos de angular y npm sin problema. Para probar, podemos simplemente escribir **npm install** y ejecutar el comando, y veremos cómo instala dependencias si fuera necesario. Tienes una lista completa de los comandos de NPM y NG en sus respectivas páginas. También puedes escribir los comandos **npm** o **ng** sin argumentos y te devoverá todas las opciones disponibles (que no son pocas).

3. Terminología de Angular

Para poder entender qué estamos haciendo, primero debemos entender qué es cada cosa y por qué se llama así.

3.1. Componentes

Un Component controla una zona de espacio de la pantalla que podríamos denominar vista. Un componente es una clase estándar de ESx decorada con **@Component**.

3.2. Plantillas

El Template es lo que te permite definir la vista de un Componente.

3.3. Servicios

Funciones que estarán disponibles en toda la aplicación y se podrán ***inyectar*** en cualquier constructor.

3.4. Modelos

Es la declaración de un objeto con sus atributos, setters y getters.

3.5. Módulos

Un módulo de Angular, es un conjunto de código dedicado a un ámbito concreto de la aplicación, o una funcionalidad específica y se define mediante una clase decorada con **@NgModule**.

3.6. Anotaciones (o Metadatos)

Son comentarios que TypeScript interpreta como órdenes, y en Angular, hacen que se interprete ese fichero de manera distinta (por ejemplo, como un Componente o un módulo).

4. Componentes

Para entender más a fondo los componentes, vamos a trabajar sobre ellos. El resultado final será una página dinámica en la cual cambiaremos un número con un click. Accederemos al archivo, desde NetBeans, que se ubica en **Sources** > **app** > **app.component.ts**.

Nota: Los componentes tienen la extensión de archivo **.component.ts**.

4.1. Crear atributo, setters y getters.

Dentro del archivo podremos observar la anteriormente mencionada etiqueta @Component, por ahora la ignoraremos y nos centraremos en el atributo.

Dentro de la clase AppComponent observaremos una variable llamada title. Cambiaremos su valor de "**app**" a "**Hola, me llamo Eusebio**".

Además, en la línea siguiente, añaderemos:

```
public miNumero:number = 0;
```

Esto significa que hemos creado un atributo público de la clase, que además es de tipo number y por defecto es 0.

Ahora crearemos sus setter y getter. Para ello, creamos las siguientes dos funciones inmediatamente después de lo anterior:

```
public getMiNumero():number
 {
    return this.miNumero;
 }
 public setMiNumero()
 {
    this.miNumero++;
 }
```

La explicación es simple: el método **getMiNumero()** devolverá un número, y **setMiNumero()** sumará 1 al atributo **miNumero**. Usaremos estas funciones más adelante (véase Plantillas).

4.2. Modificando los metadatos

Como podremos observar casi al principio del archivo, tendremos las siguientes líneas:

```
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
```

La primera línea indica que se trata de un componente, e inicia un objeto con las siguientes propiedades:

- **selector**: Etiqueta en al cual el componente se ejecutará. La etiqueta principal de nuestra aplicación es **app-root**, y por ahora no ahondaremos más en este tema (véase plantillas).
- **templateUrl**: Ubicación relativa (NUNCA ABSOLUTA) de nuestra plantilla. También se puede sustituir por la propiedad **template**.
- **template**: Aunque no está en nuestro componente, especifica la plantilla directamente sin hacer uso de ningún fichero adicional. Se debe poner entre `` (no son comillas simples, es un apóstrofe inglés) para así especificar a JavaScript que se trata de una plantilla (y no de un valor simple).
- **styleUrls**: Este parámetro es opcional. Contiene un array de los ficheros de estilo requeridos por la vista del componente.

Eliminados la propiedad templateUrl y añadimos la siguiente línea en su lugar:

```
template: `Hola, soy la plantilla. `,
```

Ejecutamos la web en el navegador y observamos que nos muestra “Hola, soy la plantilla”.

Revertimos la acción anterior y dejamos templateUrl con el valor que tenía (eliminando template), esto puede llevarse a cabo presionando **ctrl + z** en netbeans.

4.3. Eventos nativos del componente

El componente tiene ciertos eventos especiales que se generan en función de su carga: al principio, al final, cuando se cambie de ruta, cuando se cargue la plantilla… El listado de éstos está disponible aquí, pero en este caso usaremos sólo **OnInit**, que se lanza cuando se inicia el componente.

Para usar cualquiera de éstas funciones debemos realizar los siguientes pasos (Variando OnInit por el evento nativo correspondiente):

1. Agregamos la dependencia correspondiente. En nuestro caso, nuestra dependencia es **OnInit** del paquete **@angular/core**. Así que modificamos la primera línea para que quede así:

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

2. Agregar la implementación a la clase, de modo que quedaría algo como **export class AppComponent implements OnInit {** en la primera línea.
3. Agregamos la función **ngOnInit() {}** dentro de nuestra clase AppComponent.
4. Hacemos lo que queramos dentro de la función. En este caso vamos a poner el contador a 0, cuando en el atributo le indicaremos que su valor inicial es 5.

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

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

```
  title = 'Hola, me llamo Eusebio';
  public miNumero:number = 5;
  ngOnInit()
  {
      this.miNumero = 0;
  }
  public getMiNumero():number
  {
      return this.miNumero;
  }
  public setMiNumero()
  {
      this.miNumero++;
  }

}
```

5. Plantillas

5.1. Modificar la plantilla de app.component

Comenzaremos a trabajar sobre una base vacía. Así que abrimos la plantilla llamada "**app.component.html**" que se ubica al lado de app.component.ts, y borramos todo su contenido.

Agregaremos el siguiente contenido:

```
<div>
  <div>
    Mensaje inicial:
  </div>
  <div>
    {{ title }}
  </div>
</div>
<div>
  <div>
    Contador:
  </div>
  <div>
    {{ miNumero }}
  </div>
</div>
```

Cabe explicar tan sólo los elementos {{ title }} y {{ miNumero }}. Éstos, si recordamos, son los nombres de los atributos del componente. Se han de poner entre corchetes para que la plantilla las interprete con el valor del atributo de la plantilla. Visualicemos el contenido en el navegador.

5.2. Sumatorio del contador

Tenemos la variable miNumero con un setter y un getter como explicamos anteriormente al crear el componente, ha llegado el momento de utilizarlos.

Utilizaremos los eventos de angular, que si bien son parecidos a los de JavaScript/jQuery, tienen sus diferencias, pero el funcionamiento es el mismo. Para agregar nuestra nueva funcionalidad, agregaremos el siguiente contenido después del texto "**Contador:**":

```
<input type="button" value="Incrementar" (click)="setMiNumero()"/>
```

Funcionamiento muy simple. **(click)** se refiere al evento click, y dentro hemos introducido la función que se haya en nuestro componente, **setMiNumero()**. Ahora cada vez que clickemos el botón Incrementar, el número aumentará en tiempo real con el valor del atributo. Asimismo, también habrá cambiado el valor del atributo si lo usamos desde la clase.

5.3. Usando comparadores

Ahora, haremos que se muestre un mensaje cuando el número sea mayor de 3. Para ello, añadiremos el siguiente contenido al final de nuestra plantilla:

```
<div *ngIf="miNumero > 3">LA VARIABLE ES MAYOR QUE 3!</div>
```

Nótese el ***ngIf**. Dentro de él no hemos puesto los corchetes para referenciar la variable, ya que se sobreentiende que estás comparando una variable del componente. Es decir, si se cumple la condición, se mostrará ese div y sus hijos, pero si no, no se mostrará ni ese div ni sus hijos. ¡Pruébalo!

Para usar un bucle, deberíamos tener una colección, ya que la plantilla tan sólo permite colecciones y objetos para iterar. Asimismo, tenemos un número, por lo cual el for de angular no nos permite iterarlo (ya que es similar a un foreach de JavaScript). Si aún así deseas ver su estructura, puedes hacerlo aquí.

Si nos quedamos con ganas de más, podemos usar más usos comunes de las plantillas. Aquí tenemos la lista completa.

5.4. Agregando estilos al componente

NOTA IMPORTANTE: Se puede agregar un estilo dinámicamente al componente como se va a mencionar, pero **NO** se pueden incluir scripts de esta forma.

Si queremos agregar un estilo al componente actual (y no a otros), podemos hacerlo con el anteriormente mencionado atributo del componente llamado **styleUrls**. Aprovecharemos la plantilla que ya nos incorpora nuestro componente y la editaremos directamente, así que abrimos el archivo **app.component.css**, y le introducimos estas líneas (ya que estará vacío inicialmente):

```
div {
    background-color: gray;
    color: white;
}
input {
    background-color: red;
    color: white
}
```

Estoy casi seguro de que no vas a ver nada más cutre hoy, pero al menos, estás aprendiendo bien si aún estás por aquí. Puedes ver la aplicación en el navegador, y observa cómo han cambiado los estilos del componente.

5.5. Agregando estilos y scripts globales

Como ocurre con CSS a secas, puede que incluyamos un fichero en una determinada página (componente en Angular), pero en otras no… Y queramos tener un estilo base para nuestra web. No hay problema, ya que **Angular CLI** nos permite agregar ficheros de Scripts y estilos globales. Para ello, vamos a crear un archivo llamado **main.css** en el directorio **assets**, ubicado en **sources**. Le introducimos el siguiente contenido:

```
.diferente {
   background-color: blue !important;
   color: white !important
}
```

Ya tenemos nuestro CSS, pero aún no está incluido en nuestro proyecto. Podemos comprobarlo visualizándolo en el navegador. Para incluirlo, abrimos el archivo **.angular-cli.json**, ubicado en **Site Root**. Modificamos el contenido de styles, y le agregamos "**assets/main.css**", inmediatamente después de "**styles.css**", debería quedar así (ojo a la coma):

```
"styles.css",
 "assets/main.css"
```

Ya que el fichero **.angular-cli.json** sólo se carga cuando iniciamos el servidor, lo reiniciamos.
Ya nos habría cargado los nuevos estilos, pero no hemos usado la clase **.diferente**. Para usarla, simplemente añadimos la clase a un elemento (sin el ., como hacemos siempre. **class="diferente"**, no ser burro). En nuestro caso, será al div de **contador**.

5.6. Más información sobre eventos

Cualquier evento de JavaScript está disponible en Angular. Para usarlo, podemos seguir esta referencia ubicada en W3.

Para usar un evento, por ejemplo, **onclick**, simplemente debemos **eliminarle el *on*** del principio y **ponerlo entre paréntesis** dentro de nuestra plantilla de angular. Por ejemplo, lo que en JavaScript sería:

```
<img src="…" ondrag="miFuncion()">
```

En angular2+ sería:

```
<img src="…" (drag)="miFuncion()">
```

6. Servicios

Los servicios se cargan en la fase Bootstrap de la aplicación. Por ello, para poder inyectarlo correctamente en nuestro componente, es necesario agregarlo a la fase de carga de nuestra aplicación y después usarlo correctamente.

6.1. Creando el servicio de geolocalización

Vamos a crear un servicio que consiga las coordenadas de ubicación del cliente. Para ello, vamos a crear un archivo llamado **geolocalización.service.ts** en netbeans. Va a estar ubicado en **Sources** > **app**. Para crear un fichero con extensión **.ts**, debemos buscar typescript en el cuadro de búsqueda que aparece al crear un archivo.

OJO: la extensión .ts se agrega automáticamente. Revisar que nuestro fichero no tiene el .ts duplicado al final.

Rellenamos el archivo con el siguiente contenido:

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

@Injectable()
export class GeolocalizacionService {
  getCoords() {

    if (navigator.geolocation) {
      return navigator.geolocation.getCurrentPosition(this.devolverPosicion);
    } else {
      return "ERROR DE GEOLOCALIZACIÓN";
    }
  }
  private devolverPosicion(position: any) {
    /* Devuelve un objeto con las propiedades lat y lng */
    return {
      "lat": position.coords.latitude,
      "lng": position.coords.longitude
    };
  }
}
```

El código es simple si se conoce la api de geolocalización de JavaScript. Devuelve un objeto con las propiedades **lat** y **lng**, que contiene respectivamente la latitud y la longitud. Cabe destacar la etiqueta **@Injectable()**, la cual permite inyectar el servicio en los componentes.
Nota: El nombre de la clase debe ser igual al nombre del archivo, respetando mayúsculas y minúsculas, para seguir buenas prácticas.

6.2. Inyectando el servicio al bootstrap

Para poder inyectar el servicio en nuestro componente, primero lo debemos agregar a nuestra fase de carga (o Bootstrap). Para ello, agregamos esta línea (la cual importará la clase que acabamos de generar) a nuestro fichero **app.module.ts**.

```
import { GeolocalizacionService } from './geolocalizacion.service';
```

Tras esto, buscamos la línea que pone **providers**, y entre los corchetes siguientes, añadimos **GeolocalizacionService**.

6.3. Inyectando el servicio al componente y usándolo

Para inyectar el servicio al componente, debemos lógicamente importar la clase (ya que sino no sería reconocida):

```
import { GeolocalizacionService } from './geolocalizacion.service';
```

Y después crear un constructor con el servicio inyectado:

```
constructor(private geo:GeolocalizacionService)
 {
    console.log(geo.getCoords());
 }
```

De este modo, tendremos acceso desde todo el componente a la clase **GeolocalizacionService**. Como se puede apreciar, nos mostrará las coordenadas por un console.log una vez se inicie el componente. Podemos hacer lo que queramos con esas coordenadas, pasarlas a atributos y mostrarlas a través del componente, pasarlas a un API Rest…

7. Módulos

En este tutorial no ahondaremos demasiado en los módulos ya que son ligeramente más complejos, y en nuestro tutorial no tendrían mucha utilidad. Tienen una elevada complejidad, ya que hay que añadirlo y saber su herencia, pero tienen unos usos muy buenos: Traducir la página a tiempo real, un módulo de carga dinámica…

7.1. El Router

En este caso usaremos el módulo para crear un **Router**. El router es la navegación de la página, es decir, definir el nombre de las páginas y a qué componente apuntan. Ésta es realmente una de las magias de Angular, y aunque es algo complejo, allá vamos.

7.2. Creando los componentes necesarios

En primer lugar agregaremos otro componente. Usaremos el CLI de Angular. Para ello, corremos el siguiente comando en la ruta base del proyecto: `ng generate component componente2`

Tras esto, modificaremos el selector para el módulo **app.component** y **componente2.component**. Debemos eliminar el atributo selector, para que coja la etiqueta por defecto (**router-outlet**) y no tenga conflictos por duplicidad.

Ahora generaremos un nuevo componente que se encargará de ser el menú y será cargado siempre. Los componentes **app** y **componente2** serán cargados en función de la ruta actual. Para ello, volvemos a usar el comando anterior pero con otro nombre de componente: `ng generate component menu`

Ahora debemos abrir el componente menu y cambiarle también el selector. En este caso le introduciremos el selector **app-root**.

Para que nuestra aplicación funcione correctamente a posteriori, debemos modificar la plantilla del menu para que sea igual que esta:

```
<div>
   <a routerLink="/inicio" routerLinkActive="active">Inicio</a> <a routerLink="/pagina2"
routerLinkActive="active">Página 2</a>
</div>
<router-outlet></router-outlet>
```

El atributo routerLink es interpretado por angular para que el router se dirija a esa ruta. Debe ser una ruta del proyecto y no externa, además de que debe estar declarada (siguiente punto).
La etiqueta **router-outlet** define dónde se pondrá el contenido cuando se cambie de ruta.

7.3. Puesta a punto del módulo principal

Trabajaremos sobre el archivo **app.module.ts**.

En primer lugar, reemplazamos el contenido entre corchetes de **bootstrap** por **MenuComponent**. De esta forma, estamos diciendo a la aplicación que por defecto nos cargue el menú, y que en función de la ruta, debe cargar otros elementos sobre la etiqueta **router-outlet** que se ubica en la plantilla del menú.

Agregamos el siguiente import, que incluye las librerías necesarias:

```
import { RouterModule, Routes } from '@angular/router';
```

7.3.1 Agregando rutas

Y después, antes de la anotación **@NgModule**, agregamos las siguientes líneas:

```
const appRoutes: Routes = [
 { path: 'inicio', component: AppComponent },
 { path: 'pagina2',     component: Componente2Component },
 { path: '',
   redirectTo: '/inicio',
   pathMatch: 'full'
 }
];
```

Con esto estamos definiendo el componente para cada ruta. Por ejemplo, para la ruta **/inicio**, el componente será **AppComponent**, y cargará su contenido.
Para que estas rutas funcionen, debemos agregar la línea **RouterModule.forRoot(appRoutes)** dentro de los imports.

7.4 Visualizando el Router

Nuestro proyecto ya no generará errores y podremos visualizarlo de nuevo. Si clickamos en los links superiores, cambiará la vista sin recargar la página y no afectará a los estilos precargados y demás, cambiando entre componentes.

8. Compilar proyecto y visualizarlo en servidor

Para poder utilizar Angular en cualquier tipo de servidor (Apache, Nginx, etc.) tan sólo necesitamos compilarlo. Para ello, utilizamos un **ng build** ó simplemente utilizar nuestra opción **NPM Scripts** > **Compilar** de nuestro menú propiedades del proyecto de NetBeans. Una vez el comando finalice, en la ubicación principal de nuestro proyecto (en nuestro caso, **C:\my-app**) se habrá generado (si es que no estaba ya) la carpeta dist. Esta carpeta contiene la compilación del proyecto, y se compone de los ficheros HTML y javascript generados y optimizados por nuestro compilador de Angular. Si movemos éstos archivos a nuestra carpeta raíz del servidor, se visualizarán sin problema.

9. Como trasladar el proyecto de equipo

Para trasladar el proyecto de ordenador, o simplemente sincronizarlo en un repositorio de GIT, es altamente recomendable excluir las carpetas **node_modules**, **dist**, **e2e**; debido a que tienen un tamaño considerable y simplemente son inútiles a la hora de trasladarlo de equipo, ya que igualmente usaremos los comandos **npm install** y **ng build**, los cuales descargan las dependencias necesarias y compilan el proyecto, respectivamente.

Para ello, borramos estas carpetas y copiamos el proyecto donde queramos. Lo abrimos en NetBeans tal y como se explicó en los pasos 1 y 2 y corremos los comandos **npm install** y **ng build**. Y ya tendríamos el proyecto listo en el nuevo equipo.

10. Valoración de la exposición

En nuestro escritorio de dudas se ha abierto una encuesta. ¡Valora la exposición!