





Introducción a Angular





Rutas

Enrutamiento

- ◆ Las **aplicaciones Angular 6** son **conjuntos de páginas enrutadas** en el propio navegador
- ◆ Al crear la aplicación podemos hacer uso del flag **--routing** en el comando de generación del CLI. Esto causa la aparición de dos módulos en la raíz de la aplicación. Tenemos el **AppModule** que es el verdadero módulo raíz, y ahora tendremos el módulo de enrutado **AppRoutingModule**.
- ◆ Para comprender bien el enrutado vamos a crear una nueva app: `ng new cursoRutas -S --routing true`

RouterModule

- ◆ El módulo **AppRoutingModule** cumple dos funciones.
 - ◆ Por un lado importa al **RouterModule** de Angular que contiene toda la lógica necesaria para enrutar en el navegador.
 - ◆ Por otro lado, permite la definición de rutas en el array **Routes[]**. Este array recibe **objetos ruta** con propiedades de configuración.

Routes[]

```
{
  path: '**',
  redirectTo: 'not-found'
}
{
  path: '',
  loadChildren: './info/info.module#InfoModule'
}
```

◆ El primero es path: en donde se especifica la dirección que resuelve, en el último caso la ruta vacía o raíz del árbol de rutas.

◆ El segundo puede variar... vamos a ver las distintas opciones

path: '**'

◆ Es un detector de rutas no contempladas, y una ruta a dónde redirigir a los usuarios perdidos

```
{  
  path: '**',  
  redirectTo: 'not-found'  
}
```

◆ ¿Quién es **not-found**?

Práctica

- ◆ Vamos a añadir un nuevo módulo con soporte de rutas:
 - ◆ `ng g m feature --routing true`
- ◆ Debemos importar este nuevo módulo en el AppModule:
 - ◆ `imports: [`
 - ◆ `BrowserModule,`
 - ◆ `AppRoutingModule,`
 - ◆ **`FeatureModule`**
 - ◆ `],`

- ◆ Añadamos un nuevo componente en este modulo:
- ◆ `ng g c feature/enrutable`
- ◆ Configuremos la gestión de rutas parametrizadas:

```
export class EnrutableComponent implements OnInit {  
  
  public id:string;  
  constructor(private route: ActivatedRoute) { }  
  ngOnInit() {  
    this.id = this.route.snapshot.params['id']  
  }  
}
```

◆ Modificamos el router FeatureRoutingModule del nuevo módulo para usar **parámetros**:

```
const routes: Routes = [  
  {  
    path: 'enrutable/:id',  
    component: EnrutableComponent  
  }  
];  
  
@NgModule({  
  imports: [RouterModule.forChild(routes)],  
  exports: [RouterModule]  
})
```

```
import { NgModule } from '@angular/core';  
import { Routes, RouterModule } from '@angular/router';  
import { EnrutableComponent } from '../feature/enrutable/enrutable.component';
```

```
const routes: Routes = [  
  {  
    path: 'enrutable',  
    component: EnrutableComponent  
  }  
];
```

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule { }
```

AppRoutingModule...

- ◆ Añadamos un link en el app.componente.html

- ◆ `<a [routerLink]="['/enrutable']">Componente - rutas`

- ◆ Nuestro FeatureModule importa el FeatureRoutingModule y exporta el EnrutableComponent

```
@NgModule({  
  imports: [  
    CommonModule,  
    FeatureRoutingModule  
  ],  
  exports: [EnrutableComponent],  
  declarations: [EnrutableComponent]  
})
```

- ◆ La idea general de una **SPA** es tener una **única página que cargue dinámicamente otras vistas**.
- ◆ la página contenedora mantiene el menú de navegación, el pie de página y otras áreas comunes. Y deja un espacio para la carga dinámica. Para ello necesitamos saber **qué componente cargar y dónde mostrarlo**.

Router Outlet

- ◆ De esto último se ocupa el *router outlet* mediante la etiqueta `<router-outlet></router-outlet>`.
- ◆ Vamos a `app.component.html` para ver **`<router-outlet></router-outlet>`**
- ◆ Este elemento del framework inyectará dinámicamente el componente que le corresponda según la ruta activa

Práctica de redirección 1

- ◆ Vamos a crear un nuevo componente llamado `NotFoundComponent` que muestre una página amigable al usuario. Debe usar el selector **not-found**
- ◆ De esta forma cuando se escriba la ruta `/not-found` se mostrará un componente, el **NotFoundComponent**

Lazy Loading

- ◆ Consiste en diferir la carga de la lógica asociada a una dirección hasta el momento en que sea activada dicha ruta.
- ◆ De esa forma, **una página no visitada es una página que no pesa**. Y la carga inicial se hace mucho más liviana.
- ◆ En Angular 6 el *lazy loading* es sencillo y se recomienda implementarlo por defecto.
- ◆ Sin embargo, necesitamos conocer otros componentes...

- ◆ *Angular CLI* usa internamente la herramienta de empaquetado *webpack*.
- ◆ Webpack recorre el código TypeScript buscando imports y empaquetando su contenido en sacos o bundles.
- ◆ Estos Bundles suelen tener todo el código js de nuestro frontend y ser pesados por lo que webpack puede ayudarnos, junto con una buena configuración de rutas, a mejorar el tamaño pero no solo eso...

- ◆ Podemos configurar las rutas de forma que no sea necesario importar los componentes a mostrar.
- ◆ Si definimos todas las rutas tal como nuestro `NotFoundComponent`, *webpack* empaquetaría esos componentes como algo necesario... y por tanto serían enviados al navegador en el *bundle* principal sin que sea seguro su uso.
- ◆ Esa es una estrategia para componentes poco pesados y muy utilizados

Webpack & Angular-CLI

- ◆ La solución que ofrecen el *cli* y *webpack* consiste en **delegar la asignación del componente a otro módulo, pero sin importarlo** hasta que su ruta principal se active.
- ◆ Para usar esta estrategia **no debemos exportar** los elementos que queremos que se carguen bajo demanda
- ◆ Vamos a generar un nuevo módulo y componente para información general de nuestra SPA:
 - ◆ `ng g m info --routing`
 - ◆ `ng g c info/info`

Webpack & Angular-CLI

- ◆ Ahora debemos usar su ruta relativa en el módulo de enrutado `AppRoutingModule` como un valor especial.
- ◆ Para lograr esto usamos una nueva propiedad del objeto `Route`: **loadChildren**
- ◆ En nuestro `AppRoutingModule` añadimos la nueva ruta del módulo `Info`:

```
{  
  path: 'info',  
  loadChildren: './info/info.module#InfoModule'  
},
```
- ◆ No se está produciendo ninguna importación en TypeScript como ocurre con el componente `NotFoundComponent`.

Webpack & Angular-CLI

- ◆ Con esta información webpack va a generar un bundle específico para cada módulo.
- ◆ Si durante la ejecución se activa la ruta / (muy probable porque es la ruta raíz) o la ruta /info entonces se descarga ese paquete concreto y se ejecuta su contenido. Mientras tanto, se queda almacenado en el servidor.
- ◆ Esto hace que la aplicación de Angular pese menos y responda antes, mejorando el tiempo de renderización inicial.

Webpack & Angular-CLI

- ◆ Al usar el flag `--routing` se genera un segundo módulo de enrutado que es un **enrutador subordinado** al primero
- ◆ Si añadimos más módulos que queremos que se carguen usando Lazy Load, debemos usar este flag a la hora de generarlos.
- ◆ En el caso principal se usa imports:
`[RouterModule.forRoot(routes)]`
- ◆ En todos los demás imports: `[RouterModule.forChild(routes)]`

Webpack & Angular-CLI

◆ Tendríamos algo como esto:

```
import { InfoComponent } from './info/info.component';
const routes: Routes = [
  {
    path: '',
    component: InfoComponent
  }
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class InfoRoutingModule {}
```

nota:

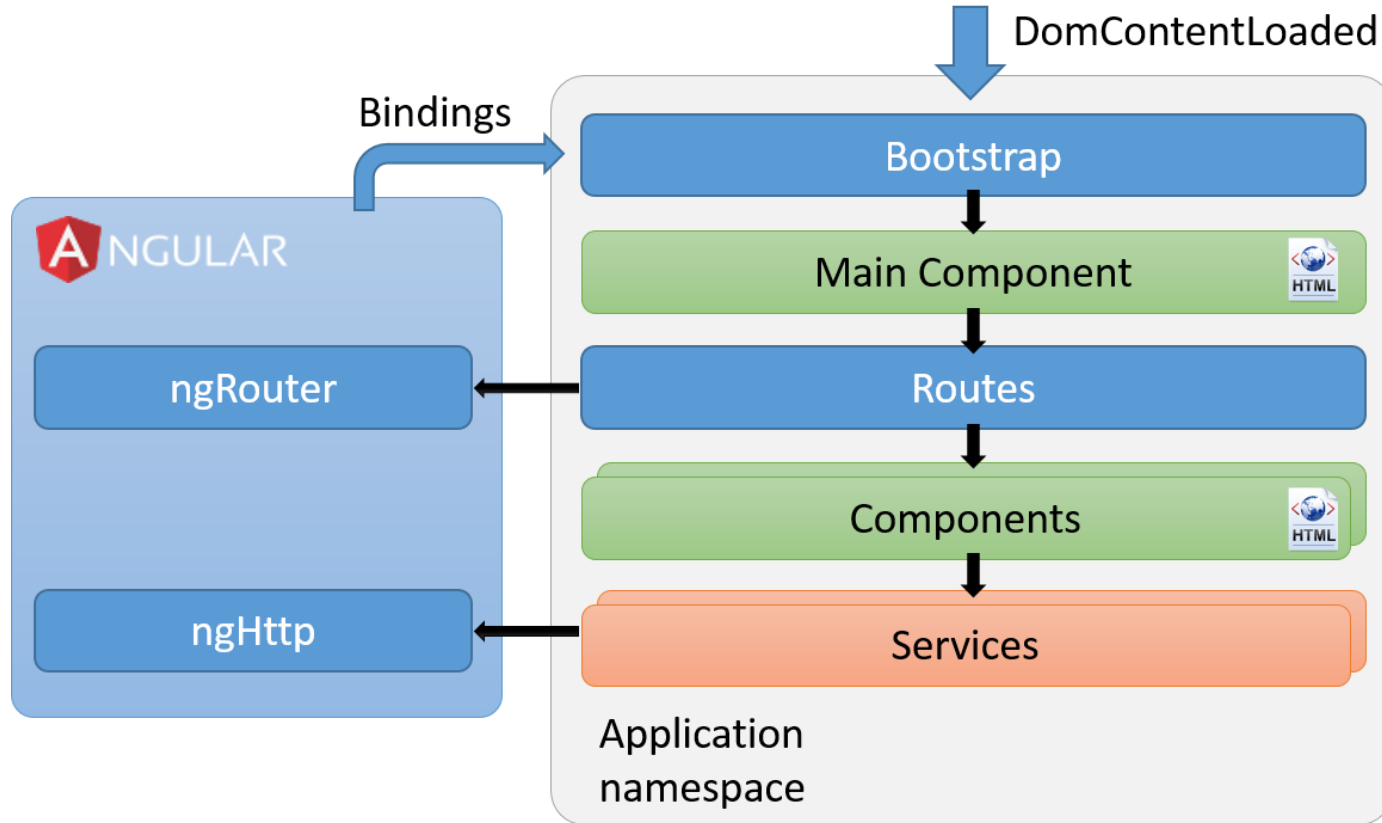
En el módulo *root* lleva

path: "info"

y en el *child* solamente

path: ""

Routing



Guards

- ◆ En ocasiones queremos que determinadas áreas de nuestra aplicación web estén protegidas y solo puedan ser accedidas si el usuario ésta logueado (un panel de control por ejemplo) o incluso que solo puedan ser accedidas por determinados tipos de usuarios.
- ◆ Para conseguir esto con Angular se usan los **guards**. Los guards pueden ser extensibles para que permitan acceder bajo las condiciones que nosotros queramos, podemos incluso hacer peticiones a un backend antes de que el usuario entre en la página.

Dentro de los guards hay 4 tipos principales:

- ◆ **CanActivate:** Mira si el usuario puede acceder a una página determinada.
- ◆ **CanActivateChild:** Mira si el usuario puede acceder a las páginas hijas de una determinada ruta.
- ◆ **CanDeactivate:** Mira si el usuario puede salir de una página, es decir, podemos hacer que aparezca un mensaje, por ejemplo, de confirmación, si el usuario tiene cambios sin guardar.
- ◆ **CanLoad:** Sirve para evitar que la aplicación cargue los módulos perezosamente si el usuario no está autorizado a hacerlo.

El primer guard

- ◆ Los guards se implementan para ser inyectados por lo tanto tenemos que usar la etiqueta `@Injectable`, como si fuera un servicio.
- ◆ Para que funcione, tenemos que importar el guard en el **`app.module.ts`**, en la sección **`providers`**
- ◆ Los guards devuelven **`true`** o **`false`** para permitir el paso o no de un usuario a la ruta. También pueden devolver un `Observable` o una `Promise` si el guard no puede responder inmediatamente y tiene que esperar.

```
import { Injectable } from '@angular/core';  
import { Router, CanActivate } from '@angular/router';
```

```
@Injectable()
```

```
export class CanActivateViaAuthGuard implements CanActivate {
```

```
  logueado:boolean = true;
```

```
  constructor(private router: Router) { }
```

```
  canActivate() {
```

```
    if (!this.logueado) {
```

```
      console.log('No estás logueado');
```

```
      this.router.navigate(['/']);
```

```
      return false;
```

```
    }
```

```
    return true;
```

```
  }
```

```
}
```

◆ Para usar este guard en una ruta, lo importamos en el archivo de rutas y añadimos un campo a la ruta llamado **canActivate** con el guard que acabamos de crear:

```
import { CanActivateViaAuthGuard } from '../guards/seguridad.guard';

const routes: Routes = [
  {
    path: 'enrutable/:id',
    component: EnrutableComponent,
    canActivate: [CanActivateViaAuthGuard]
  }
];
```



Testing y Depuración

Depuración

- ◆ Trabajaremos con Augury, una herramienta de depuración de aplicaciones Angular que corre en el navegador.
- ◆ Actualmente solo se encuentra disponible para Chrome, en forma de extensión al panel de herramientas de desarrollo
- ◆ Está respaldada por parte del equipo de Angular como una herramienta oficial.

Depuración

- ◆ Nos ayudará a la hora de explorar las relaciones entre los componentes individuales de nuestra aplicación que son visibles en la página.
- ◆ Por ejemplo, podemos ver el estado de un componente y modificarlo, podemos lanzar eventos manualmente, ver el código fuente del componente, etc.
- ◆ La instalamos desde: <https://augury.rangle.io/>

La interfaz de Augury

- ◆ Se compone, principalmente, de las siguientes secciones:
- ◆ **Component Tree:** es una lista de todos los componentes Angular que son visibles en la página. Si realizamos alguna acción que altera esos componentes o su estado, los cambios se resaltan actualizados en el panel de Augury. Cada uno de los componentes podrá tener información útil como: propiedades, providers, estado, inputs, dependencias, etc.

La interfaz de Augury

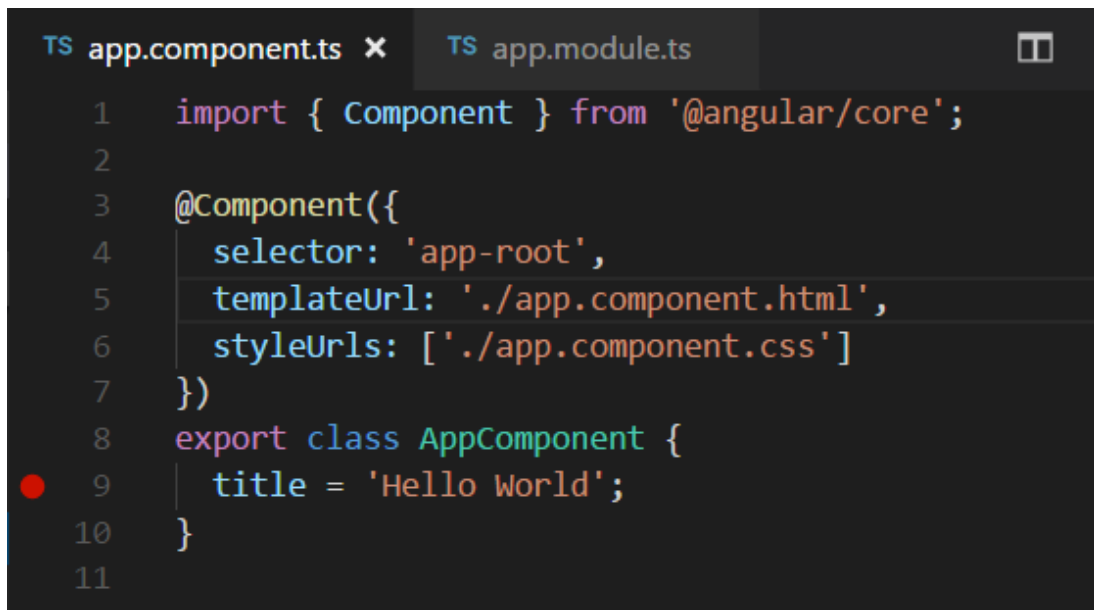
- ◆ **Injector Graph:** para cada componente podemos ver un diagrama de las dependencias inyectadas en él.
- ◆ **Router Tree:** diagrama de todas las rutas definidas en la aplicación (sólo funcionará si tenemos inyectado un Router en el componente root de la aplicación).
- ◆ **NgModules:** una vista con lo que tenemos en nuestro “app.module.ts”.
- ◆ Vamos al lío!

Augury y Debugger for Chrome

- ◆ Augury puede ser una herramienta muy útil a la hora de visualizar y depurar de manera concisa y resumida los componentes que forman nuestra aplicación Angular.
- ◆ Sin embargo, la mejor forma de depurar es con la extensión [Debugger for Chrome](#)
- ◆ La instalamos desde la vista extensiones en VSCode

Debugger for Chrome

◆ Una vez instalada podemos añadir break points en nuestros componentes:



```
TS app.component.ts x TS app.module.ts
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'Hello World';
10 }
11
```

Debugger for Chrome

- ◆ Necesitamos configurar nuestra extensión para que use la instancia de angular que tenemos en ejecución, para esto presionamos:
- ◆ Ctrl+Shift+D o vamos a la vista de depuración y presionamos el botón en Configuración (engranage)
- ◆ Ahora elegimos Chrome como Entorno
- ◆ Esto nos creará un archivo .vscode con la configuración a ajustar

Debugger for Chrome

◆ Necesitamos tener algo como esto:

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "chrome",  
      "request": "launch",  
      "name": "Launch Chrome against localhost",  
      "url": "http://localhost:4200",  
      "webRoot": "${workspaceFolder}"  
    }  
  ]  
}
```

Debugger for Chrome

◆ Ahora presionamos F5 para arrancar y podremos tener depuración en los puntos de interrupción definidos

Testing

- ◆ Angular es un framework que ya viene con la integración para que podamos ejecutar y escribir nuestros tests sin tener que hacer mucho (bueno si, los tests los vamos a tener que escribir).
- ◆ El testing en Angular está compuesto de varios elementos que vamos a conocer



◆ Jasmine es un behavior-driven development framework para probar código JavaScript. No depende de ningún otro framework de JavaScript. No requiere un DOM. Y tiene una sintaxis clara y obvia para que pueda escribir fácilmente pruebas.



◆ karma js es un *test runner*, desarrollado por el equipo de angular, que nos permite automatizar algunas tareas de los frames de tests, como jasmine.

Testing

- ◆ Vamos a generar una nueva app, vamos a añadir un componente y en vez de arrancar, ejecutamos:
 - ◆ `npm test`
- ◆ Esto ejecutará las pruebas y levantará un servidor para consultar el reporte de las mismas, estará disponible en:
 - ◆ `http://localhost:9876/`

Testing

- ◆ Cuando lanzamos el test, la configuración lo que hace es mirar las carpetas y revisar si tiene *****.spec.ts***, para poder ejecutar los tests...
- ◆ Vamos al: ***app.component.spec.ts***

Testing

- ◆ Importamos TestBed y Async para poder ejecutar nuestros tests y el componente que vamos a testear.
- ◆ **TestBed** es la primera y más importante de las utilidades para poder hacer pruebas en Angular.
- ◆ Crea un módulo de prueba angular que se configura con el método `configureTestingModule` para producir el entorno del módulo para la clase que desea probar.
- ◆ Separa el componente “*a probar*” de su propio módulo de aplicación y lo vuelve a conectar a un módulo de prueba Angular de construcción dinámica adaptado específicamente para estas pruebas.

Testing

- ◆ Luego vemos que se utiliza una función “it”, que casualmente en su primer parámetro recibe el texto que imprime nuestro test,

```
it('should create', () => {  
  expect(component).toBeTruthy();  
});
```

- ◆ Luego vemos validaciones de igualdad o de que contenga cierto contenido

Testing

◆ Vamos a añadir test para nuestra tabla de usuarios para asegurarnos que tiene la longitud esperada



Deployment

Despliegue

- ◆ Para poder desplegar nuestra aplicación necesitamos usar el comando `ng build`
- ◆ Al lanzarlo sin parámetros se nos crea una carpeta **dist** con lo compilado de nuestra aplicación.
- ◆ `ng build` también nos permite hacer más cosas enfocadas a desplegar en producción:
 - ◆ Minificar
 - ◆ Uglyficar

- ◆ Hace el minify uglify, es decir, reduce el tamaño del código, quitando los espacios en blanco y acortando el nombre de las variables.
- ◆ Realiza Tree Shaking que es una técnica para eliminar el código inaccesible
- ◆ precompila el código de forma que el cliente no tenga que bajarse el compilador cuando visita la web:

ng build—prod

- ◆ veremos que en los archivos se crean unos identificadores:

```
$ ng build --prod  
  
Date: 2018-10-03T08:58:12.156Z  
Hash: 9e951b50386b64d9c93c  
Time: 34416ms  
chunk {0} runtime.ec2944dd8b20ec099bf3.js (runtime) 1.44 kB [entry] [rendered]  
chunk {1} main.29ca1af28427658a0eca.js (main) 729 kB [initial] [rendered]  
chunk {2} polyfills.f6ae3e8b63939c618130.js (polyfills) 59.6 kB [initial] [rendered]  
chunk {3} styles.3bb2a9d4949b7dc120a9.css (styles) 0 bytes [initial] [rendered]
```

- ◆ La idea es que cada vez que desplegamos se generan nuevos ids para cada fichero de forma que se desplieguen los más recientes.

◆ `ng build --prod --build-optimizer` es una variante que añade aun más mejoras de optimización

◆ Para probarlo, vamos a descargar:

<http://nginx.org/en/download.html>

◆ Copiamos en la carpeta **html** de nginx el contenido de nuestra carpeta **dist**



Gracias!

Nos vemos en el próximo!

Pueden contactarnos mediante info@impactotecnologico.net y apuntarse a algunos de nuestros cursos



