

Routing

En el desarrollo web, routing significa dividir una aplicación en diferentes áreas, usualmente basadas en determinadas reglas y derivadas a través de su dirección URL.

Por Qué necesitamos el ruteo de páginas

Definir rutas en nuestra aplicación es útil por los siguientes motivos:

- Separar diferentes áreas de nuestra aplicación
- Mantener el estado dentro de la aplicación
- Proteger áreas de la aplicación en base a determinadas reglas.

El routing nos permite definir un string para la URL que especifique en donde el usuario debe estar dentro de la aplicación.

Cómo trabaja el ruteo en el Front-end

Cuando hacemos un ruteo desde el lado del servidor, al venir el requerimiento HTTP el servidor redirigirá a diferentes controladores dependiendo de la URL entrante.

El patrón puede variar dependiendo si utilizamos un framework o no, y de qué framework utilicemos, pero en reglas generales tenemos un servidor que acepta un requerimiento y este redirige a un controlador, y este a su vez, corre una acción específica, dependiendo de la ruta y de los parámetros.

El ruteo del lado del cliente es similar pero de diferente implementación. En el lado del cliente no es necesario hacer un requerimiento al servidor cada vez que cambie la URL. En las aplicaciones Angular, nos referimos a este tipo de ruteo como "*Simple Page Apps*", porque el servidor solo provee una única página (simple page) y es JavaScript el que renderiza a las diferentes páginas.

Angular Routing

En Angular se configura las rutas mapeando los path en el componente que los va a manejar. El router de Angular puede interpretar una URL como una instrucción para navegar a una vista generada por el cliente. Se le puede pasar parámetro que lo ayuda a decidir qué contenido específico presentar.

Componentes de Angular Routing.

Hay tres componentes fundamentales que usamos para configurar el ruteo en Angular:

- *Routes*: describe las rutas que nuestra aplicación va a soportar.
- *RouterOutlet*: Es el componente de posición que le muestra a Angular donde colocar el contenido de cada ruta.

- *RouterLink*: es la directiva que utilizamos para linkear las rutas.

Para utilizar el router de Angular, debemos importar constantes del paquete *@angular/router*.

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

A partir de ahora podemos definir nuestra configuración de ruteo.

Routes

Para definir las rutas de nuestra aplicación, creamos una configuración de *Routes* y luego utilizamos el *RouterModule.forRoot(routes)* para proporcionar a nuestra aplicación las dependencias necesarias para usar el router.

```
const routes: Routes = [  
  {path: '', redirectTo: 'home', pathMatch: 'full'},  
  {path: 'home', component: HomeComponent},  
  {path: 'contacto', component: ContactoComponent},  
  {path: 'about', component: AboutComponent},  
  {path: 'aboutus', redirectTo: 'contacto'}  
];
```

El *path* especifica la URL de la ruta que va a manejar. El *component* es el lazo que une el path de la ruta con el componente que la va a manejar. El *redirectTo* es opcional, y lo usamos para redireccionar el path declarado a una ruta existente.

Redirecciones

Cuando usamos el *redirectTo* en la definición de una ruta, le estamos diciendo al router que cuando visitemos el path declarado pretendemos que el browser lo redirija a otra ruta.

```
{path: 'aboutus', redirectTo: 'contacto'}
```

En este caso si la url ingresada al browser es *http://localhost:4200/#/aboutus* nos va a redirigir a la url *http://localhost:4200/#/contacto*.

<base href>

La mayoría de las aplicaciones de ruteo deben agregar un elemento *<base>* como elemento hijo del tag *<head>* del *index.html* para decirle al router como componer las URLs de navegación.

```
<head>
```

```
  <meta charset="utf-8">
```

```
  <title>Routing</title>
```

```
  <base href="/">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
  <link rel="icon" type="image/x-icon" href="favicon.ico">
```

```
</head>
```

Router Outlet

La directiva *RouterOutlet* de la librería *router* se utiliza como un componente. Este actúa como marcador de posición que establece el lugar en el que el template de los componentes se mostrará.

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

Router Links

Ya tenemos configurado el ruteo, y también tenemos el lugar donde vamos a mostrar el contenido de los componentes, pero cómo hacemos para navegar? La URL puede ser accedida directamente desde la barra de navegación del navegador. Pero la mayoría de las veces navegamos como resultado de alguna acción como hacer click en un botón del menú.

La directiva *RouterLink* en las etiquetas *<a>* nos da el control de navegación sobre esos elementos. Las rutas de navegación son fijas, por lo tanto podemos asignar un string a cada *routerLink* ("one-time" binding).

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <a class="navbar-brand" href="#">Angular Routing</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
```

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav mr-auto">
    <li class="nav-item active">
      <a class="nav-link" routerLink="/home">Home</a>
    </li>
    <li class="nav-item active">
      <a class="nav-link" routerLink="/contacto">Contacto</a>
    </li>
    <li class="nav-item active">
      <a class="nav-link" routerLink="/about">Acerca de</a>
    </li>
    <li class="nav-item active">
      <a class="nav-link" routerLink="/aboutus">Nosotros</a>
```

```

        </li>
    </ul>
</div>
</nav>

<router-outlet></router-outlet>

```

Router Links activos

La directiva *RouterLinkActive* alterna clases css de acuerdo al RouterLink activo basado en su *RouterState* actual.

La expresión tendrá a la derecha del igual(=) una cadena de clases CSS delimitada por espacios que el Router agregará cuando este enlace este activo (y se eliminará cuando el enlace esté inactivo).

En el siguiente código evaluamos qué URL está activa y en base a eso agregamos la clase *active* de bootstrap al tag *<a>*.

```

<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
        <li class="nav-item">
            <a class="nav-link" routerLinkActive='active'
routerLink="/home">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" routerLinkActive='active'
routerLink="/contacto">Contacto</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" routerLinkActive='active'
routerLink="/about">Acerca de</a>
        </li>
    </ul>
</div>

```

Parámetros de navegación

En nuestras aplicaciones muchas veces tenemos que navegar a un recurso específico. Digamos que tenemos un sitio de noticias al cual le agregamos numerosos artículos. Cada uno de estos artículos tiene un ID, y si por ejemplo tenemos un artículo con un ID 3 entonces tenemos que navegar a ese artículo visitando la URL:

`/articulos/3`

Y si tenemos un artículo con un ID 4 deberíamos acceder de esta manera:

`/articulos/4`

y así sucesivamente.

Desde ya no vamos a escribir una ruta por cada uno de los artículos, por lo tanto queremos usar una *variable de navegación*. Podemos especificar que esa ruta toma un parámetro poniendo dos puntos(:)

`/route/:param`

```
const routes: Routes = [  
  {path: 'noticias/:id', component: NoticiasComponent},  
];
```

Cuando visitemos la ruta `/noticias/3`, la parte del 3 va a ser pasada como un parámetro de navegación llamado *id*.

Pero cómo hacemos para recuperar ese parámetro desde el componente? Para eso usamos los parámetros de navegación.

ActivatedRoute.

Para poder usar los parámetros de navegación, tenemos que importar *ActivatedRoute*.

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

```
@Component({  
  selector: 'app-noticias',  
  templateUrl: './noticias.component.html',  
  styleUrls: ['./noticias.component.css']  
})  
export class NoticiasComponent implements OnInit {  
  
  constructor(activatedRoute : ActivatedRoute) { }  
  
  ngOnInit() {  
  }  
  
}
```

Como vemos en el código anterior, después de importar el *ActivatedRoute* lo inyectamos en el constructor.

```
export class NoticiasComponent {  
  id:string;  
  constructor(activatedRoute : ActivatedRoute) {
```

```
    activatedRoute.params.subscribe(params => {this.id =  
params['id'];});  
  }  
}
```

Notemos que el *activatedRoute.params* es un *observable*. Podemos extraer el valor del parámetro usando *.subscribe*. En este caso se lo asignamos a una instancia de la clase declarada en el componente.

Ahora podemos usar esta instancia en nuestra vista:

```
<div class="container">  
  <h1>Noticia nro. {{id}}</h1>  
</div>
```

NgModules.

NgModules configura el inyector y el compilador y ayuda a organizar elementos relacionados en un determinado conjunto.

El NgModule es una clase marcada con el decorador *@NgModule*. *@NgModule* describe cómo compilar una vista de un componente y como crear un inyector en tiempo de ejecución. Este identifica los componentes y directivas que pertenecen al módulo, haciendo a algunos de ellos públicos, mediante la propiedad *export*, para que los componentes externos puedan utilizarlos. *@NgModule* también puede agregar proveedores de servicios a los inyectores de dependencias de la aplicación.

Modularización de Angular.

Los módulos son una buena forma de organizar una aplicación y extender sus capacidades a través de librerías externas.

Las librerías de Angular son NgModules, tales como *FormsModule*, *HttpClient* y *RouterModule*.

NgModule consolida componente y directivas en bloques cohesivos de funcionalidad, cada uno con el foco puesto en una determinada área de prestación, aplicación de dominio empresarial, flujo de trabajo o colección de utilidades comúnmente utilizadas.

Los módulos pueden agregar servicios a la aplicación. Estos pueden ser desarrollados internamente, o puede venir de fuentes externas, como el router de Angular o el cliente HTTP.

La metadata de NgModule hace lo siguiente:

- Declara cuáles componentes y directivas pertenecen al módulo.
- Transforma a algunos de los componentes y directivas como públicos, para que otros componentes de otros módulos pueden utilizarlos.
- Importa otros módulos con componentes y directivas que los componentes del módulo puedan necesitar.
- Provee servicios que otros componentes de la aplicación puedan usar.

Toda aplicación Angular tiene al menos un módulo, el *root module*. Este módulo es todo lo que necesitamos en una aplicación simple que tenga algunos componentes. Pero a medida que la app

crece refactorizamos este módulo raíz en módulos que representan funcionalidades relacionadas. Luego importamos esos módulos en el módulo raíz.

Veamos un ejemplo. En este caso vamos a agrupar los componentes AboutComponent y NosotrosComponent en un módulo llamado AboutModule

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AboutComponent } from '../about/about.component';
import { ContactoComponent } from '../contacto/contacto.component';
```

```
@NgModule({
  declarations: [
    AboutComponent,
    ContactoComponent
  ],
  imports: [
    CommonModule
  ],
  exports: [
    AboutComponent,
    ContactoComponent,
  ]
})
```

```
export class AboutModule { }
```

Importamos los dos componentes, lo agregamos a las declaraciones y luego los exportamos. De esta manera cualquier módulo que importe el *AboutModule* va a tener acceso a los dos componentes exportados.

Importamos el *AboutModule* en el módulo raíz:

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

```
import { AboutModule } from '../about.module';
```

```
import { AppRoutingModule } from '../app-routing.module';
import { AppComponent } from '../app.component';
import { HomeComponent } from '../home/home.component';
import { NoticiasComponent } from '../noticias/noticias.component';
```

```

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    NoticiasComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    AboutModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Deployment.

El acto de deployar una aplicación es subir nuestro código a un servidor, desde el cual puede ser accedido por otros usuarios.

Hablando en forma general, la idea es que vamos a:

- *compile*: compilar todo el código *Typescript* en código *JavaScript* (que el navegador puede leer).
- *bundle*: Poner todos los archivos de código *JavaScript* en un "manejo" de uno o dos archivos.
- *upload*: Subir los *bundles*, librerías e imágenes al servidor.

Construir nuestra aplicación para subirla a Producción.








La herramienta *Angular Cli* que utilizamos para generar la aplicación puede ser usada para construir nuestra aplicación para ser subida a producción.

```
ng build --prod --base-href /
```

Este comando le dice al comando *ng* que vamos a construir la aplicación para el entorno Producción.

El *--base-href /* describe cual es la URL raíz de la aplicación. Si por ejemplo queremos deployar la aplicación a una subcarpeta del servidor, por ejemplo */curso-angular/*, entonces nuestro *--base-href* sería */curso-angular/*.

El resultante se va a almacenar en una carpeta llamada *dist*.

 3rdpartylicenses.txt
 favicon.ico
 index.html
 main.f4aa6edb79622141a386.js
 polyfills.c6871e56cb80756a5498.js
 runtime.ec2944dd8b20ec099bf3.js
 styles.a4b0a4963e19ee73e88e.css

Documento de texto
Icono
Chrome HTML Docu...
Archivo JS
Archivo JS
Archivo JS
Archivo CSS