Angular 2

Ángel Villalba Fdez-Paniagua

Table of Contents

1. Routing

Las rutas permiten cambiar entre las distintas secciones de la aplicación, y dependiendo de la ruta se cargan unos componentes u otros. Angular Router parsea la URL e intenta identificar el componente que tiene que cargar teniendo en cuenta los datos que van en la ruta en caso de que los tenga.

Para poder usar las rutas hay que configurar el proyecto para poder cambiar de sección desde cualquier parte de la aplicación.

Para empezar hay que crear un archivo de rutas `app.routing.ts` donde se van a configurar las rutas. Hay que importar `Routes` desde `@angular/router`. Las rutas se guardan en un array (de tipo Routes) de objetos javascript donde cada objeto representa una ruta. Cada objeto va a contener como mínimo:

- **path:** forma de la URL. En el caso de la ruta `home` se deja como un string vacío.
- **component:** el componente que se va a mostrar cuando estemos en esa URL. Los componentes hay que importarlos.

Para que las rutas estén disponibles en toda la aplicación hay que exportarlas para ponerlas en el archivo `app.modules.ts`.

Para el ejemplo siguiente, necesitamos crear un componente `home` en la carpeta app, un componente `usuario` y dentro de la carpeta usuario, dos componentes mas, `usuario-info` y `editar-usuario`.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';

const APP_ROUTES: Routes = [
  {path: 'usuario', component: UsuarioComponent},
  {path: '', component: HomeComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

Hay que importar las rutas en el `app.module.ts`:

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

import { AppComponent } from './app.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { EditarUsuarioComponent } from './usuario/editar-usuario.component';
import { UsuarioInfoComponent } from './usuario/usuario-info.component';
import { HomeComponent } from './home.component';
import { routing } from './app.routing';

@NgModule({
  declarations: [
    AppComponent,
    UsuarioComponent,
    EditarUsuarioComponent,
    UsuarioInfoComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    routing
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Ahora mismo debería de fallar porque faltaría decirle donde se tiene que renderizar el componente. Para ello hay que usar la etiqueta `<router-outlet></router-outlet>` en el sitio donde se quiere mostrar el componente correspondiente a la ruta.

```
<div class="container">
  <div class="row">
    <div class="col-md-10">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

Ya no tiene que fallar, sino que debería de mostrar el componente.

Añadimos unos links para poder navegar entre secciones.

```
<div class="container">
  <div class="row">
    <div class="col-md-10">
      <h1>Hello World!</h1>
      <hr>
      <a href="/">Home</a> |
      <a href="/usuario">Usuario</a>
      <hr>
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

Al hacer click sobre cualquiera de los dos enlaces cambia de componente, pero también recarga toda la página, y eso es algo feo y poco usable. En lugar de recargar la página debería cambiar de componente. Por supuesto, Angular es capaz de hacerlo. En lugar de `href` hay que usar la directiva `routerLink`.

```
<div class="container">
  <div class="row">
    <div class="col-md-10">
      <h1>Hello World!</h1>
      <hr>
      <a [routerLink]="['']">Home</a> |
      <a [routerLink]="['usuario']">Usuario</a>
      <hr>
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

Entre los corchetes se encuentran los segmentos de la URL, donde cada segmento es aquello que va despues de cada `/`. Por ejemplo en una ruta como `localhost:4200/usuario/10` habría dos segmentos, ["usuario", "10"].

Ahora al navegar por los links debería de cambiar el componente sin recargar la página.

Se pueden usar rutas absolutas (con `/`) o rutas relativas (sin `/`). Si estamos en la ruta de inicio (`localhost:4200/home`) y vamos a la ruta del usuario:

- **/usuario** va a `localhost:4200/usuario`.
- **usuario** va a `localhost:4200/usuario`.

Pero si estamos en la ruta `localhost:4200/usuario`:

- **/usuario** va a `localhost:4200/usuario`, es decir, que se queda donde está.
- **usuario** va a `localhost:4200/usuario/usuario`.

Cuando ponemos en el segmento `/` indica que la ruta comienza después del dominio.

Hay algunos casos que se necesita navegar a otra ruta sin pinchar en un link, sino que se necesita hacer la navegación a través del código, por ejemplo cuando te registras en una página te redirige a la ruta de inicio. Para ello hay que usar el método `navigate` del router. Este método tiene como argumento un array de segmentos, por lo que actúa de la misma forma que el `routerLink`.

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

@Component({
  selector: 'app-usuario',
  template: `
    <h3>Usuario</h3>
    <button (click)="onNavigate()">Ir a inicio</button>
    <hr>
    {{id}}
    <hr>
  `,
  styles: []
})
export class UsuarioComponent {
  constructor(private router: Router) { }

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

1.1. Parámetros

Las rutas se pueden personalizar un poco mas. De momento al ir a la página de `/usuario` nos muestra los datos de un solo usuario, pero en todas las aplicaciones hay mas de un usuario por lo que tenemos que indicar en la URL la página del usuario que queremos ver. Esto se indica con los parámetros de las rutas que pueden contener un valor. Estos párametros se definen de la siguiente forma `{ path: 'usuario/:id', component: UsuarioComponent }`.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';

const APP_ROUTES: Routes = [
  {path: 'usuario/:id', component: UsuarioComponent},
  {path: '', component: HomeComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

Y ahora en los links donde se llama al usuario hay que añadir un segundo segmento que es el id de ese usuario.

```
<div class="container">
  <div class="row">
    <div class="col-md-10">
      <h1>Hello World!</h1>
      <hr>
      <a [routerLink]="['']">Home</a> |
      <input type="text" #id (input)="0" />
      <a [routerLink]="['usuario', id.value]">Usuario</a>
      <hr>
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

Para obtener los parámetros de una URL, hay que importar `ActivatedRoute` de `@angular/router` e inyectarlo en el constructor. Y podemos acceder a ellos como viene a continuación:

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

@Component({
  selector: 'app-usuario',
  template: `
    <h3>Usuario</h3>
    <button (click)="onNavigate()">Ir a inicio</button>
    <hr>
    {{id}}
    <hr>
  `,
  styles: []
})
export class UsuarioComponent {
  id: string;

  constructor(private router: Router, private activatedRoute: ActivatedRoute) {
    this.id = activatedRoute.snapshot.params['id'];
  }

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

El método `snapshot.params['id']` nos da un objeto de clave-valor en el que el id es una clave y lo usamos para obtener su valor. Pero si le cambiamos el id en el input para que nos mande al

componente correspondiente con ese id se puede observar que en la URL se ha cambiado el id, pero el componente no se ha cargado. Angular es muy eficiente y si detecta que queremos ir al mismo componente, no lo destruye para volverlo a crear, sino que realiza los cambios que detecta. Pero como no esta creando el componente otra vez, no detecta el cambio del id y no lanza el constructor otra vez para actualizar el componente. La alternativa que hay que usar a snapshot es `params`, que es un observable.

Un observable envuelve un objeto y escucha los cambios que se producen en dicho objeto para poder reaccionar a ellos.

```
import { Component, OnDestroy } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs/Rx';

@Component({
  selector: 'app-usuario',
  template: `
    <h3>Usuario</h3>
    <button (click)="onNavigate()">Ir a inicio</button>
    <hr>
    {{id}}
    <hr>
  `,
  styles: []
})
export class UsuarioComponent implements OnDestroy {
  private subscription: Subscription;
  id: string;

  constructor(private router: Router, private activatedRoute: ActivatedRoute) {
    this.subscription = activatedRoute.params.subscribe(
      (param: any) => this.id = param['id']
    );
  }

  onNavigate() {
    this.router.navigate(['/']);
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

Ahora se subscribe a cualquier cambio en los parámetros, y en el momento en que se produzca, ejecuta el callback pasandole como parámetro el param del que se puede obtener el valor del id. La subscripcion la guardamos en una variable de tipo `Subscription` que se importa desde `rxjs/Rx` y de la que vamos a desubscribirnos una vez que se elimine el componente para evitar perdidas de memoria.

1.2. Redireccionar rutas

Se puede usar `redirectTo` al definir una ruta, de esta forma al visitar una URL que coincide con el path indicado en la definición, nos redireccionará a la ruta indicada.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';

const APP_ROUTES: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'usuario/:id', component: UsuarioComponent},
  {path: 'home', component: HomeComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

1.3. Rutas hijas

Las rutas hijas son aquellas que dependen de los parámetros de una ruta, hasta ahora teniamos la ruta `usuario/:id`, pero ahora queremos una ruta que permita editar ese usuario por lo que necesitaremos una ruta del estilo `usuario/:id/editar`. Esta es una child route porque cuelga de la ruta del usuario y depende del id.

Para crear estas rutas hay que crearse otro archivo de rutas en este caso en la carpeta de **usuario**. Y las rutas definidas en este archivo hay que exportarlas para importarlas y usarlas en el archivo de rutas principal.

```
import { Routes } from '@angular/router';
import { UsuarioInfoComponent } from './usuario-info.component';
import { EditarUsuarioComponent } from './editar-usuario.component';

export const USUARIO_ROUTES: Routes = [
  { path: 'info', component: UsuarioInfoComponent },
  { path: 'editar', component: EditarUsuarioComponent }
];
```

En el archivo `app.routing.ts` además de importar las rutas, hay que asignarselas a la ruta de la que van a depender.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { USUARIO_ROUTES } from './usuario/usuario.routing';

const APP_ROUTES: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'usuario/:id', component: UsuarioComponent, children: USUARIO_ROUTES },
  {path: 'home', component: HomeComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

Para que funcione, hay que indicar donde se quiere renderizar el componente hijo con la etiqueta `router-outlet`. Y también tendremos que añadir los enlaces a los componente hijos.

```
import { Component, OnDestroy } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs/Rx';

@Component({
  selector: 'app-usuario',
  template: `
    <h3>Usuario</h3>
    <button (click)="onNavigate()">Ir a inicio</button>
    <hr>
    {{id}}
    <hr>
    <a [routerLink]="['editar']">Editar usuario</a>
    <a [routerLink]="['info']">Info de usuario</a>
    <router-outlet></router-outlet>
  `,
  styles: []
})
export class UsuarioComponent implements OnDestroy {
  private subscription: Subscription;
  id: string;

  constructor(private router: Router, private activatedRoute: ActivatedRoute) {
    this.subscription = activatedRoute.params.subscribe(
      (param: any) => this.id = param['id']
    );
  }

  onNavigate() {
    this.router.navigate(['/']);
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

En el caso de que no se inserte un id, la URL no sería correcta, porque siempre tiene que recibir un id. En este caso, cuando no lo reciba hay que redireccionarla a una URL válida.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { USUARIO_ROUTES } from './usuario/usuario.routing';

const APP_ROUTES: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'usuario/', redirectTo: 'usuario/1'},
  {path: 'usuario/:id', component: UsuarioComponent, children: USUARIO_ROUTES },
  {path: '', component: HomeComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

1.4. Guards

Las guards se usan para controlar a que partes de la aplicación puede navegar un usuario. Por ejemplo, es posible que algunas de las rutas de la aplicación las queramos restringir solo a usuarios logueados, y podemos usar estas guards para controlar las condiciones en las que pueden o no entrar o salir de una ruta.

1.4.1. CanActivate

Este guard se ejecuta al entrar en una ruta.

El guard que se ha creado implementa la interface CanActivate la cual tiene dos parámetros, donde el primero es la ruta que está activada, y el segundo es el estado de esa ruta. Este método devolverá un Observable, una Promise o un boolean. Si el valor devuelto es `true` permite pasar a la ruta, mientras que si es `false` no. El observable se usa cuando el valor depende de que algo se ejecute asíncronamente y tengamos que esperar a la respuesta.

usuario-info.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from
'@angular/router';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class UsuarioInfoGuard implements CanActivate {
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> | Promise<boolean> | boolean {
    return confirm('Estas seguro??');
  }
}
```

Para asignar el **guard** a una ruta, hay que poner en la ruta la clave `canActivate` cuyo valor será un array con todas las guards que tiene que pasar esa ruta.

usuario.routing.ts

```
import { Routes } from '@angular/router';
import { UsuarioInfoComponent } from './usuario-info.component';
import { EditarUsuarioComponent } from './editar-usuario.component';
import { UsuarioInfoGuard } from './usuario-info.guard';

export const USUARIO_ROUTES: Routes = [
  { path: 'info', component: UsuarioInfoComponent, canActivate: [UsuarioInfoGuard] },
  { path: 'editar', component: EditarUsuarioComponent }
];
```

Ahora solo faltaría añadir el guard en el array de **providers** del `app.module.ts` para que funcione.

app.module.ts

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

import { AppComponent } from './app.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { EditarUsuarioComponent } from './usuario/editar-usuario.component';
import { UsuarioInfoComponent } from './usuario/usuario-info.component';
import { HomeComponent } from './home.component';
import { routing } from './app.routing';
import { UsuarioInfoGuard } from './usuario/usuario-info.guard';

@NgModule({
  declarations: [
    AppComponent,
    UsuarioComponent,
    EditarUsuarioComponent,
    UsuarioInfoComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    routing
  ],
  providers: [UsuarioInfoGuard],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

1.4.2. CanDeactivate

Este guard se lanza cuando te vas a salir de una cierta ruta.

Es parecido al anterior, pero hay que tener en cuenta algunas cosas. Para empezar al crear el guard, hay que importar el CanDeactivate e implementarlo en la clase. Este CanDeactivate tiene como parámetro un interface que se va a implementar en los componentes que necesiten usar esta guard. Hay que definir el interface en este mismo archivo, el cual no tiene ningún parámetro y devuelve al igual que el guard, un Observable o un boolean.

editar-usuario.guard.ts

```
import { CanDeactivate } from '@angular/router';
import { Observable } from 'rxjs/Observable';

export interface ComponentCanDeactivate {
  canDeactivate: () => Observable<boolean> | boolean;
}

export class EditarUsuarioGuard implements CanDeactivate<ComponentCanDeactivate> {
  canDeactivate(component: ComponentCanDeactivate): Observable<boolean> | boolean {
    return component.canDeactivate ? component.canDeactivate() : true;
  }
}
```

En el canDeactivate de la clase del guard, se le indica que si el componente tiene implementado el interface de arriba nos deje salir según el valor que devuelva ese método, y si no lo tiene implementado entonces permite salir siempre.

Una vez definido el interface, hay que implementarlo en el componente.

editar-usuario.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { Observable } from 'rxjs/Rx';
import { ComponentCanDeactivate } from './editar-usuario.guard';

@Component({
  selector: 'app-editar-usuario',
  template: `
    <h3>Editar Usuario</h3>
    <button (click)="hecho=true">Hecho</button>
    <button (click)="onNavigate()">Ir a inicio</button>
  `,
  styles: []
})
export class EditarUsuarioComponent {
  hecho: boolean = false;
  constructor(private router: Router) { }

  onNavigate() {
    this.router.navigate(['/']);
  }

  canDeactivate(): Observable<boolean> | boolean {
    if(!this.hecho) {
      return confirm('Seguro que quieres salir??');
    }
    return true;
  }
}
```

No hay que olvidar añadir el guard en el array de providers del archivo `app.module.ts`.

app.module.ts

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

import { AppComponent } from './app.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { EditarUsuarioComponent } from './usuario/editar-usuario.component';
import { UsuarioInfoComponent } from './usuario/usuario-info.component';
import { HomeComponent } from './home.component';
import { routing } from './app.routing';
import { UsuarioInfoGuard } from './usuario/usuario-info.guard';
import { EditarUsuarioGuard } from './usuario/editar-usuario.guard';

@NgModule({
  declarations: [
    AppComponent,
    UsuarioComponent,
    EditarUsuarioComponent,
    UsuarioInfoComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    routing
  ],
  providers: [UsuarioInfoGuard, EditarUsuarioGuard],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Y por último se añade a la ruta correspondiente con la clave canDeactivate que tiene por valor un array de guards en el que hay que incluir esta que se ha creado.

usuario.routing.ts

```
import { Routes } from '@angular/router';
import { UsuarioInfoComponent } from './usuario-info.component';
import { EditarUsuarioComponent } from './editar-usuario.component';
import { UsuarioInfoGuard } from './usuario-info.guard';
import { EditarUsuarioGuard } from './editar-usuario.guard';

export const USUARIO_ROUTES: Routes = [
  { path: 'info', component: UsuarioInfoComponent, canActivate: [UsuarioInfoGuard] },
  { path: 'editar', component: EditarUsuarioComponent, canDeactivate:
[EditarUsuarioGuard] }
];
```

1.5. Query params

A veces necesitamos incluir en la URL unos parámetros opcionales para proporcionar más información, como por ejemplo para indicar a la paginación en que página nos encontramos. Estos parámetros son los **query params** que están compuestos por una clave y un valor separados por un igual.

Para pasar estos parámetros opcionales a una ruta a través de la directiva `routerLink` se usa otra directiva junto a esa que es la `queryParams` cuyo valor tiene que ser un objeto con los parámetros opcionales.

```
<div class="container">
  <div class="row">
    <div class="col-md-10">
      <h1>Hello World!</h1>
      <hr>
      <a [routerLink]="['']">Home</a> |
      <input type="text" #id (input)="1" />
      <a [routerLink]="['usuario', id.value]">Usuario</a>
      <a [routerLink]="['usuario', id.value]" [queryParams]="{visible: false}">Usuario
con id no visible</a>
      <hr>
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

Al igual que con los parámetros normales, también nos podemos **subscribir a los query params** usando el observable `queryParams()` que nos proporciona ActivatedRoute.

```
import { Component, OnDestroy } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs/Rx';

@Component({
  selector: 'app-usuario',
  template: `
    <h3>Usuario</h3>
    <button (click)="onNavigate()">Ir a inicio</button>
    <hr>
    {{ visible ? '***' : id }}
    <hr>
    <a [routerLink]="['editar']">Editar usuario</a>
    <a [routerLink]="['info']">Info de usuario</a>
    <router-outlet></router-outlet>
  `,
  styles: []
})
export class UsuarioComponent implements OnDestroy {
  private subscription: Subscription;
  private subscritionQueryParams: Subscription;
  id: string;
  visible = true;

  constructor(private router: Router, private activatedRoute: ActivatedRoute) {
    this.subscription = activatedRoute.params.subscribe(
      (param: any) => this.id = param['id']
    );
    this.subscritionQueryParams = activatedRoute.queryParams.subscribe(
      (queryParam: any) => this.visible = queryParam['visible']
    );

  }

  onNavigate() {
    this.router.navigate(['/']);
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
    this.subscritionQueryParams.unsubscribe();
  }
}
```

Y en el caso de que se los vayamos a pasar a través del método `navigate`, tendremos que pasarle un objeto como segundo parámetro, donde la clave será `queryParams` y el valor otro objeto con los parámetros.

```
import { Component, OnDestroy } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs/Rx';

@Component({
  selector: 'app-usuario',
  template: `
    <h3>Usuario</h3>
    <button (click)="onNavigate()">Ir a inicio</button>
    <hr>
    {{ visible ? '***' : id }}
    <hr>
    <a [routerLink]="['editar']">Editar usuario</a>
    <a [routerLink]="['info']">Info de usuario</a>
    <button type="button" (click)="irAEditarUsuario()">Editar usuario con query
param</button>
    <router-outlet></router-outlet>
  `,
  styles: []
})
export class UsuarioComponent implements OnDestroy {
  private subscription: Subscription;
  private subscritionQueryParams: Subscription;
  id: string;
  visible = true;

  constructor(private router: Router, private activatedRoute: ActivatedRoute) {
    this.subscription = activatedRoute.params.subscribe(
      (param: any) => this.id = param['id']
    );
    this.subscritionQueryParams = activatedRoute.queryParams.subscribe(
      (queryParam: any) => this.visible = queryParam['visible']
    );

  }

  onNavigate() {
    this.router.navigate(['/']);
  }

  irAEditarUsuario() {
    this.router.navigate(['usuario', this.id, 'editar'], {queryParams: { hecho: true
}});
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
    this.subscritionQueryParams.unsubscribe();
  }
}
```

```
import { Component, OnDestroy } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { Observable, Subscription } from 'rxjs/Rx';
import { ComponentCanDeactivate } from './editar-usuario.guard';

@Component({
  selector: 'app-editar-usuario',
  template: `
    <h3>Editar Usuario</h3>
    <button (click)="hecho=true">Hecho</button>
    <button (click)="onNavigate()">Ir a inicio</button>
  `,
  styles: []
})
export class EditarUsuarioComponent implements OnDestroy {
  hecho = false;
  subscription: Subscription;

  constructor(private router: Router, private route: ActivatedRoute) {
    this.subscription = this.route.queryParams.subscribe(
      (queryParam: any) => this.hecho = queryParam['hecho']
    );
  }

  onNavigate() {
    this.router.navigate(['/']);
  }

  canDeactivate(): Observable<boolean> | boolean {
    if (!this.hecho) {
      return confirm('Seguro que quieres salir??');
    }
    return true;
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

1.6. Rutas **

El path ** es un comodín que sirve para mostrar un componente en caso de que ningún path de los que hay en las rutas coincida con la url. Se suele usar para mostrar un componente con la página *404* o para redireccionar a otro lugar. Esta ruta tiene que ir en la última posición del array de rutas para que funcione correctamente.

Vamos a irnos al archivo de rutas donde pondremos una ruta nueva con el path que hemos dicho y que nos mostrar un nuevo componente que crearemos después, el cual nos mostrará una página de

error.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { USUARIO_ROUTES } from './usuario/usuario.routing';
import { ErrorComponent } from './error.component';

const APP_ROUTES: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'usuario/', redirectTo: 'usuario/1'},
  {path: 'usuario/:id', component: UsuarioComponent, children: USUARIO_ROUTES },
  {path: 'home', component: HomeComponent},
  {path: '**', component: ErrorComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

Y el componente quedaría de la siguiente forma:

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

@Component({
  selector: 'app-error',
  template: `
    <h2>
      Página no encontrada!
    </h2>
  `,
  styles: []
})
export class ErrorComponent { }
```

1.7. Rutas auxiliares

Angular tiene otro tipo de rutas, las **rutas auxiliares** que nos permiten tener múltiples rutas independientes que dividen una página en varias regiones. Los componentes pueden tener cero o mas rutas auxiliares.

Para crear una región para una ruta auxiliar hay que usar un `router-outlet` con una propiedad `name` única para diferenciar entre las distintas regiones que podemos crear, quedando de la siguiente forma `<router-outlet name="nombre-region"></router-outlet>`.

Y por último también es necesario poner enlaces que vayan a la ruta que nos mostrará los componentes en esa región. El enlace será como el siguiente `<a [routerLink]="[{ outlets: {'nombre-región': 'path-aux'} }]>Enlace</a>"`

```
<div class="container">
  <div class="row">
    <div class="col-md-10">
      <h1>Hello World!</h1>
      <hr>
      <a [routerLink]="['']">Home</a> |
      <input type="text" #id (input)="1" />
      <a [routerLink]="['usuario', id.value]">Usuario</a>
      <a [routerLink]="['usuario', id.value]" [queryParams]="{visible: false}">Usuario
con id no visible</a>
      <a [routerLink]="[{outlets: {'reg-aux': 'aux1'}}]>Aux 1</a>"
      <a [routerLink]="[{outlets: {'reg-aux': 'aux2'}}]>Aux 2</a>"
      <hr>
      <router-outlet></router-outlet>
      <router-outlet name="reg-aux"></router-outlet>
    </div>
  </div>
</div>
```

Y añadimos dos rutas auxiliares a nuestro archivo de rutas para indicar que componente se tiene que mostrar en la nueva región de la página. En las rutas auxiliares hay que ponerles la propiedad `outlet` que indica en que región hay que mostrar el componente, quedando de la siguiente forma `{ path: 'valor-path', component: 'nombre-componente', outlet: 'nombre-region-aux' }`.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component';
import { UsuarioComponent } from './usuario/usuario.component';
import { USUARIO_ROUTES } from './usuario/usuario.routing';
import { ErrorComponent } from './error.component';
import { Aux1Component } from './aux1.component';
import { Aux2Component } from './aux2.component';

const APP_ROUTES: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'usuario/', redirectTo: 'usuario/1'},
  {path: 'usuario/:id', component: UsuarioComponent, children: USUARIO_ROUTES },
  {path: 'home', component: HomeComponent},
  {path: 'aux1', component: Aux1Component, outlet: 'reg-aux'},
  {path: 'aux2', component: Aux2Component, outlet: 'reg-aux'},
  {path: '**', component: ErrorComponent}
];

export const routing = RouterModule.forRoot(APP_ROUTES);
```

Por último habría que crear los dos componentes que vamos a mostrar en la región auxiliar.

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

@Component({
  selector: 'app-aux1',
  template: `
    <h2>
      Aux1
    </h2>
  `,
  styles: []
})
export class Aux1Component { }
```

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

@Component({
  selector: 'app-aux2',
  template: `
    <h2>
      Aux2
    </h2>
  `,
  styles: []
})
export class Aux2Component { }
```

Y de esta forma al pulsar en alguno de los enlaces que acabamos de añadir, nos mostrará en la nueva región los componentes correspondientes.