Qué es un componente web y cómo crear rutas con el router-outlet

Coding Potions leccion 2

Qué es un componente

Como vimos anteriormente, Angular se basa en componentes. Un componente se basa en la creación de fragmentos con vista, estilos y controladores de forma que puedan ser reutilizadas en distintas partes de la web. Por ejemplo, pongamos que creamos un componente que reciba una lista de elementos y los pinte en el html. Una vez creado podemos añadir este componente de lista que hemos creado en

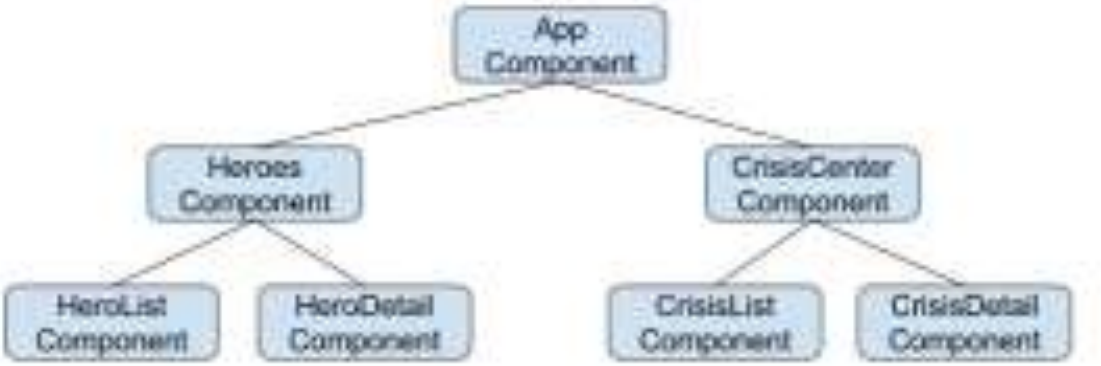
varios sitios de tal forma que le podemos pasar los elementos que queremos pintar para que los pinte. Esto ofrece la ventaja de tener el código modular, es decir, si tenemos que efectuar un cambio en la manera en la que visualizamos las listas, por ejemplo, solo lo tenemos que realizar una vez para todas las listas.

Un componente se puede componer de varios archivos: vista, estilos, controladores, servicios, etc.

La vista (html) y los estilos (css), definen qué y cómo queremos representar la web. En los controladores se encuentra la lógica de los componentes. Desde este archivo podemos inicializar las variables para la vista, actualizarlos, llamar a otros archivos, crear funciones, etc. Desde los servicios es donde se hacen las llamadas para gestionar los datos, por ejemplo guardar datos, es decir, desde los controladores es mejor no gestionar datos directamente, sino

que lo mejor es delegar estas funciones a los servicios.

Por ejemplo una estructura de componentes con componentes padres e hijos puede ser la siguiente:



Creando componentes en Angular

Para crear los componentes en Angular existen dos maneras de hacerlo: la método sencillo y el manual.

Método automático

Si estamos usando Angular cli, en nuestro proyecto, existe un comando para generar componentes:

Por ejemplo, imaginemos que queremos crear un componente para mostrar el navbar en todas las páginas:

```
ng generate component navbar
```

Si navegamos ahora a la carpeta app del proyecto veremos que Angular cli ha creado por nosotros una carpeta llamada navbar. Dentro de la carpeta navbar se ha creado un archivo css, un archivo html y un archivo .ts (controlador) junto con un .spec.ts (tests). Otro detalle a tener en cuenta es que Angular ha importado por nosotros el

nuevo componente en el archivo app.module.ts.

Método manual

Como podrás imaginar, este método consiste en crear los archivos que te hagan falta manualmente para el componente. Además, tendrás que importarlo manualmente en el archivo app.module.ts.

Angular recomienda separar los componentes en carpetas según su funcionalidad, de esta forma será más fácil

localizarlos y el código será más fácil de mantener.

Para crear un controlador vacío (archivo .ts) la estructura es la siguiente (por ejemplo para el componente de navbar):

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


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

```
export class NavbarComponent  {


  constructor() { }


}
```

Importamos ‘Component’ de @angular/core, llamamos a @Component y le pasamos tres cosas (de momento):

- selector: El selector es el nombre que va a tener la etiqueta html que se crea con el componente, para el navbar será <app-navbar></app-navbar>,

es decir desde el html de cualquier otro componente, poniendo esa etiqueta se pintará el navbar.

Angular tiene una convención de nombre para el selector, kebab-case (el nombre de los selectores tiene que ser una palabra seguida de un guión y otra palabra: app-ejemplo).

- templateUrl: La url de la vista html asociada al componente.
- styleUrl: La url del estilo CSS asociado al componente.

Por último hacemos export class y el nombre de la clase.

Cómo crear rutas en Angular. Sistema de routing

Con esto sabemos crear componentes sueltos (todavía sin funcionalidad) pero qué pasa si hemos creado un componente para una página entera, es decir, ¿cómo se crean páginas en Angular, y cómo asignarles una ruta en la página?

Para ello necesitamos hacer uso del routing de Angular, necesitamos el archivo con las rutas.

Creamos un archivo al mismo nivel que el app.module.ts y lo llamamos app.routing.ts. Una vez creado importamos las rutas de Angular:

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

Ahora, creamos una variable con las rutas:

```
const appRoutes = [
```

```
    { path: '', component: ItemListComponent,
pathMatch: 'full'}
];
```

- path: La ruta a que queremos configurar
- component: Componente asociado a esa ruta. Para que funcione tienes que importar el componente en la parte de arriba, por ejemplo:
- import { ItemListComponent } from './item-list/item-list.component';
-

- pathMatch: Esto es opcional, significa que toda la ruta URL tiene que coincidir.

Ahora, imaginemos que queremos crear una página para mostrar en detalle los items de la lista, entonces necesitamos que Angular genere una ruta para cada item, eso se puede hacer de la siguiente manera:

```
{ path: 'item/:id', component: ItemDetailComponent }
```

:id indica que se generarán rutas con distintos id, luego dentro del controlador del

detalle de item, recogeremos este valor y mostraremos el item correspondiente.

Para recoger este valor, dentro del componente de detalle del item, tenemos que incluir en el constructor:

```
this.myId = activatedRoute.snapshot.params['id'];
```

this.myId es una variable que he creado dentro de el componente.

myId tendrá el valor que aparece en la ruta, es decir, si navegamos a la ruta /item/2, myId tendrá valor 2.

Y si queremos iuna página 404 que aparezca cuando una ruta no coincide con alguna de las anteriores:

```
{ path: '**', component: PageNotFoundComponent }
```

Una vez creadas todas las rutas, al final del fichero de routing introducimos:

```
export const routing =

RouterModule.forRoot(appRoutes);
```

El fichero completo, sin la ruta a la página 404, quedaría de la siguiente manera:

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

import { AppComponent } from './app.component';
```

```
import { ItemListComponent } from './item-list/item-
list.component';
import { ItemDetailComponent } from './item-
detail/item-detail.component';


const appRoutes = [
    { path: '', component: ItemListComponent,
pathMatch: 'full'},
    { path: 'item/2', component: ItemDetailComponent,
},
];
export const routing =
RouterModule.forRoot(appRoutes);
```

Ahora, tenemos que importar las rutas en el archivo app.module.ts, para ello importamos la ruta y lo añadimos, esta vez en la parte de imports:

```
import { routing } from './app.routing';


...


  imports: [
    BrowserModule,
    routing
  ],
```

Router outlet

Si pruebas las páginas con estos cambios te darás cuenta de que todavía no se muestran las nuevas rutas, esto pasa porque en el archivo app.component.html que es el primer componente que se carga, tenemos que quitar el html que viene por defecto para poner una etiqueta especial:

<router-outlet></router-outlet>

router-outlet es una etiqueta especial en Angular que sirve para mostrar los

componentes hijos de un componente. Por defecto todos los componentes son hijos del componente AppComponent, por lo que si incluímos esta etiqueta dentro de la vista de AppComponent, se renderizará cada uno de los componentes del routing dependiendo de la página en la que nos encontremos. Si nos encontramos en la ruta, por ejemplo, /item/2 se renderizará en el lugar de router-outlet el componente de ItemDetailComponent como definimos en el routing.+

Para incluir un componente dentro de otro se usa el selector que definas dentro del componente que quieres renderizar, por ejemplo para el navbar:

```
<app-navbar></app-navbar>
```

Como hemos dicho antes, como todos los componentes son hijos del AppComponent, si incluyes el navbar en la vista del AppComponent (además del router-outlet), el navbar se visualizará en todas las páginas de la web.

Componentes hijos

Para que un componente tenga componentes hijos asociados, lo tenemos que especificar en el routing:

```
{
        path: 'admin', component: AdminComponent,
        children: [
            { path: '', component: MainComponent, },
            { path: 'settings', component:
SettingsComponent },
        ]
    },
```

En este ejempllo, la página de admin tiene dos componentes hijos. En la ruta /admin/ se cargará el componente MainComponent y en la ruta /admin/settings, el componente SettingsComponent

Los componentes hijos se dibujarán en el router outlet que coloquemos en el padre, es decir para que se renderizen estos componentes, tenemos que poner en la vista del AdminComponent un router-outlet. Todo lo que incluyamos en la vista del

AdminComponent se visualizará también en las dos páginas hijas.

¿Y en el navbar cómo podemos poner links a páginas de nuestra web?

Ya no podemos usar el atributo href, de no ser que queramos navegar a una página fuera de la web.

Para poner un link ahora tenemos que usar:

```
  <a routerLink="/list"
routerLinkActive="active">Link</a>
```

De esta manera, por ejemplo, navegaremos a la url list, y si la tenemos configurada en el app.routing.ts se cargará su vista dentro del router-outlet visto anteriormente.

Conclusiones

Con esto ya sabemos cómo crear componentes y como asignarlos a rutas, aunque ésto es solo una parte de todo lo que se puede hacer con rutas y componentes.

Lo que yo recomiendo es separar cada cosa en varios componentes, por ejemplo, un componente para mostrar la cabecera de una página, otro para mostrar toda la lista de elementos (en verde), otro componente por cada categoría de la lista (en rojo) y otro por cada item de la lista (azul oscuro). Para páginas pequeñas esta estructura es más tediosa de programar pero para páginas más grandes es la mejor estructura ya que al estar todo dividido en

componentes es más fácil de mantener ya que cada cosa funciona independiente.