

Curso de Angular Router: Lazy Loading y Programación Modular



Vistas anidadas 12/25

RECURSOS

APUNTES

Trabajar con Angular en una aplicación modularizada da la posibilidad de que, cada módulo, tenga a su vez N cantidad de páginas hijas. Veamos cómo es posible estructurar un proyecto para este propósito.

Proceso para modularizar una aplicación

El CLI de Angular te ayudará a crear rápidamente un nuevo módulo. Para esto, utiliza el comando ng generate module <module-name> --routing , o en su forma corta ng g m <module-name> --routing .

Presta atención al flag --routing que le indica a Angular que tiene que crear, además del módulo, el archivo base para agregarle rutas a este nuevo módulo.

1. Crear módulos necesarios

Comienza creando algunos módulos para tu aplicación. Por ejemplo, separaremos la app en un módulo para todo el sitio público y otro para el administrador.

- ng generate module modules/website --routing
- ng generate module modules/cms --routing

2. Preparación del routing

A continuación, prepara tu app-routing.module.ts para *Lazy Loading* y *CodeSplitting* importando los módulos de la siguiente manera:

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { NotFoundComponent } from './components/not-found/not-found.component';
const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./modules/website/website.module').then(m =>
m.WebsiteModule)
  },
    path: 'cms',
    loadChildren: () => import('./modules/cms/cms.module').then(m => m.CmsModule)
  },
    path: '**',
    component: NotFoundComponent
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
export class AppRoutingModule { }
```

Observa que, a excepción del módulo 404, solo estamos importando los módulos de una manera especial. Con loadChildren, cada módulo será enviado bajo demanda, ya que ahora se cargan de manera asíncrona.

3. Renderizar módulos

El componente principal de tu aplicación será el encargado de renderizar cada módulo. Para esto, asegúrate de que solo posea el <router-outlet> , porque es todo lo que necesitas para lograrlo.

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
    selector: 'app-root',
    template: '<router-outlet></router-outlet>',
    styleUrls: ['./app.component.scss']
})
export class AppComponent { }
```

Incluso puedes borrar el archivo app.components.html y colocar el <router-outlet> dentro de la propiedad template en el decorador @Component() para simplificar tu código.

4. Componentes base

Cada uno de tú módulos, también deberá tener un componente base al cual denominaremos Layout.

- ng g c modules/website/layout
- ng g c modules/cms/layout

Para el caso del layout del módulo website, agrégale su propio < router-outlet> además del componente para la barra de navegación.

```
<!-- modules/website/layout/layout.component.html -->
<app-nav-bar></app-nav-bar>
<router-outlet></router-outlet>
```

Mientras que el layout del módulo CMS, solo necesitará del <router-outlet> , ya que, de momento, no posee <header> o <footer> exclusivo.

```
<!-- modules/cms/layout/layout.component.html -->
<router-outlet></router-outlet>
```

5. Routing de cada módulo

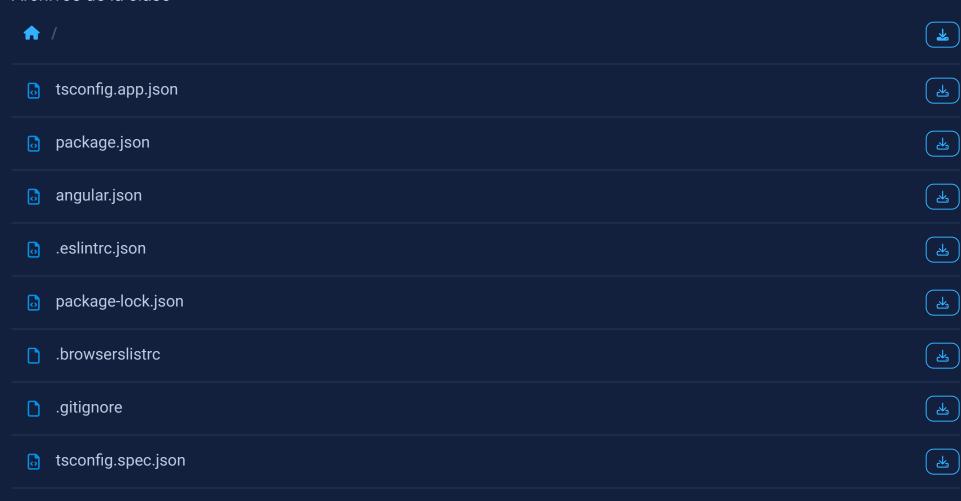
```
children: [
       path: '',
       redirectTo: '/home',
       pathMatch: 'full'
      },
       path: 'home',
       component: HomeComponent
      },
       path: 'about',
       component: AboutComponent
      },
       path: 'catalogo',
       component: CatalogoComponent
      },
       path: 'catalogo/:categoryId',
       component: CatalogoComponent
      },
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
 exports: [RouterModule]
export class WebsiteRoutingModule { }
```

Presta atención a la propiedad children que construye las nuevas reglas para las rutas.

De esta manera, puedes tener un <router-outlet> dentro de otro <router-outlet> para renderizar páginas hijas de cada módulo y tener un layout personalizado por nada uno de ellos. Además de estar optimizado el rendimiento de tu aplicación gracias al *Lazy Loading* y *CodeSplitting*.

Contribución creada por: Kevin Fiorentino.

Archivos de la clase



ō	tsconfig.json	(F)
۵	README.md	F
٥	.editorconfig	4
ō	karma.conf.js	4
	src	