




DESARROLLO DE APLICACIONES WEB II

2.-Angular router

- 
- Las aplicaciones angular siguen la arquitectura de aplicación de página única (SPA), donde se pueden activar diferentes vistas de la página web de acuerdo con la URL del navegador. Cualquier cambio en esa URL puede ser interceptado por el enrutador Angular y traducido a rutas que pueden activar un componente Angular en particular. Scully es un popular generador de sitios web estáticos que se basa en la arquitectura Jamstack.
 - Puede cooperar muy bien con el enrutador angular para preprocesar el contenido de una aplicación angular de acuerdo con cada ruta.




Vamos a combinar Angular y Scully para crear un blog personal. Se cubrirán los siguientes temas:


- Configuración de enrutamiento en una aplicación Angular
- Creando el diseño básico de nuestro blog
- Configurar el enrutamiento para nuestra aplicación
- Agregar capacidades de blog con Scully
- Mostrar publicaciones de blog en la página de inicio

TEORÍA Y CONTEXTO ESENCIALES

- En los viejos tiempos del desarrollo web, las aplicaciones del lado del cliente estaban altamente acopladas con la infraestructura del servidor subyacente. Cuando queríamos visitar la página de un sitio web usando una URL se hacían muchas cosas internamente.
- El navegador enviaría la URL solicitada al servidor, y el servidor debería responder con un archivo HTML coincidente para esa URL. Este era un proceso complicado que daría lugar a retrasos y tiempos variables de ida y vuelta.

- 
- Las aplicaciones web modernas eliminan estos problemas utilizando la arquitectura SPA. Un cliente necesita solicitar un único archivo HTML solo una vez al servidor. Cualquier cambio posterior en la URL del navegador se maneja internamente desde la infraestructura del cliente. En Angular, el enrutador es responsable de interceptar las solicitudes de URL en la aplicación y manejarlas de acuerdo con una configuración de ruta definida.

- Jamstack es una tecnología emergente que nos permite crear aplicaciones web rápidas y seguras. Se puede utilizar para cualquier tipo de aplicación, desde un sitio web de comercio electrónico hasta una aplicación web de software como servicio (SaaS) o incluso un blog personal. La arquitectura de Jamstack se basa en los siguientes pilares:
 - Rendimiento: las páginas se generan y renderizan previamente durante la producción, lo que elimina la necesidad de esperar a que se cargue el contenido.
 - Escalado: el contenido son archivos estáticos que se pueden servir desde cualquier lugar, incluso desde un proveedor de Content Delivery Network (CDN) que mejora el rendimiento de la aplicación.
 - Seguridad: la naturaleza sin servidor de los procesos del lado del servidor y el hecho de que el contenido ya es estático elimina los posibles ataques dirigidos a las infraestructuras del servidor.

- 
- Scully es el primer generador de sitios web estáticos para Angular que adopta el enfoque Jamstack. Básicamente, genera páginas de la aplicación Angular durante el tiempo de compilación para que estén disponibles de inmediato cuando se soliciten.

DESCRIPCIÓN DEL PROYECTO

- En este proyecto, crearemos un blog personal usando Angular y lo mejoraremos con características Jamstack usando el generador de sitios de Scully. Inicialmente, crearemos una nueva aplicación angular y la habilitaremos para el enrutamiento. Luego crearemos el diseño básico de nuestra aplicación agregando algunos componentes básicos. Tan pronto como tengamos una aplicación Angular en funcionamiento, le agregaremos soporte para blogs usando Scully. Luego crearemos algunas publicaciones de blog usando archivos Markdown y las mostraremos en la página de inicio de nuestra aplicación.
- Tiempo de construcción: 1 hora.

EMPEZAMOS

- Iniciaremos nuestro proyecto creando una nueva aplicación Angular desde cero. Ejecute el siguiente comando de Angular CLI en una ventana de terminal para crear una nueva aplicación Angular:

```
C:\Users\Geovany\Desktop\Angular> ng new miBlog --routing --style=scss
```

- Usamos el comando `ng new` para crear una nueva aplicación Angular, pasando las siguientes opciones:
 - `miBlog`: El nombre de la aplicación Angular que queremos crear. Angular CLI creará una carpeta `miBlog` en la ruta donde ejecutamos el comando.
 - Nota IMPORTANTE
 - Cada comando que ejecutamos en la ventana del terminal debe ejecutarse dentro de esta carpeta.
 - `--routing`: habilita el enrutamiento en la aplicación Angular.
 - `--style = scss`: Configura la aplicación Angular para usar el formato de hoja de estilo SCSS cuando se trabaja con estilos CSS.

- Cuando habilitamos el enrutamiento en una aplicación Angular, Angular CLI importa varios elementos del paquete @angular/router npm en nuestra aplicación:

```
app-routing.module.ts X
src > app > app-routing.module.ts
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3
4  const routes: Routes = [];
5
6  @NgModule({
7    imports: [RouterModule.forRoot(routes)],
8    exports: [RouterModule]
9  })
10 export class AppRoutingModule { }
11
```

- Importa AppRoutingModuleModule al módulo principal de nuestra aplicación, app.module.ts:

```
app.module.ts x
src > app > app.module.ts > ...
You, 3 minutes ago | 1 author (You)
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppRoutingModuleModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6
You, 3 minutes ago | 1 author (You)
7 @NgModule({
8   declarations: [
9     AppComponent
10  ],
11  imports: [
12    BrowserModule,
13    AppRoutingModule
14  ],
15  providers: [],
16  bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

- Configuramos nuestra aplicación para usar el formato de hoja de estilo SCSS. En lugar de crear los estilos de nuestra aplicación manualmente, usaremos la biblioteca Bootstrap CSS:
- Ejecute el siguiente comando en una ventana de terminal para instalar Bootstrap:

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> npm install bootstrap
```

- Usamos el ejecutable npm para instalar el paquete bootstrap desde el registro npm en el comando anterior.
- Agregue la siguiente declaración de importación en la parte superior del archivo styles.scss que existe en la carpeta src de nuestra aplicación Angular:

```
styles.scss ●
src > styles.scss
...
1  /* You can add global styles to this file, and also import other style files */
2  @import "~bootstrap/scss/bootstrap";
```

- El archivo `styles.scss` contiene estilos CSS que se aplican globalmente en nuestra aplicación. En el fragmento anterior, importamos todos los estilos de la biblioteca Bootstrap a nuestra aplicación. La regla `@import` CSS acepta la ruta absoluta del archivo `bootstrap.scss` como una opción, sin agregar la extensión `.scss`.
- El carácter `~` representa la carpeta `node_modules` de nuestra aplicación Angular.
- La carpeta `node_modules` contiene todos los paquetes y bibliotecas npm que nuestra aplicación necesita, ya sea durante el desarrollo o en tiempo de ejecución.
- Ahora aprenderemos cómo crear el diseño básico de nuestro blog mediante la creación de componentes, como el encabezado y el pie de página.

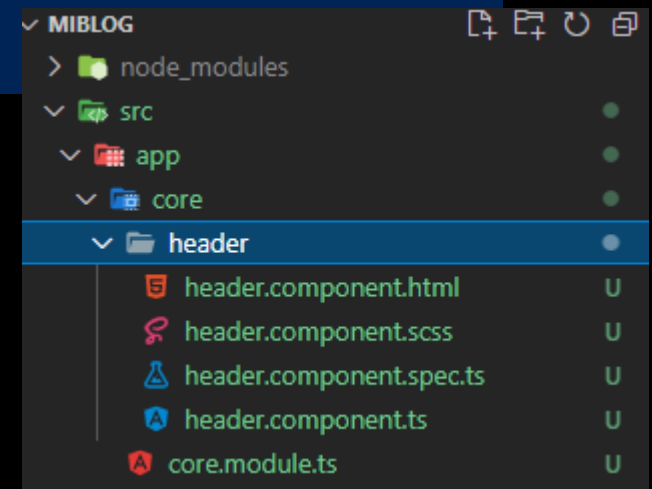
CREANDO EL DISEÑO BÁSICO DE NUESTRO BLOG

- Un blog suele tener un encabezado que contiene todos los enlaces del sitio web principal y un pie de página que contiene información sobre derechos de autor y otros enlaces útiles. En el mundo de Angular, ambos se pueden representar como componentes separados.
- El componente de encabezado se usa solo una vez, ya que se agrega cuando se inicia nuestra aplicación, y siempre se representa como el menú principal del sitio web. En Angular, normalmente creamos un módulo, denominado núcleo por convención, para mantener dichos componentes o servicios en el centro de nuestra aplicación. Para crear el módulo, usamos el comando generate de Angular CLI:

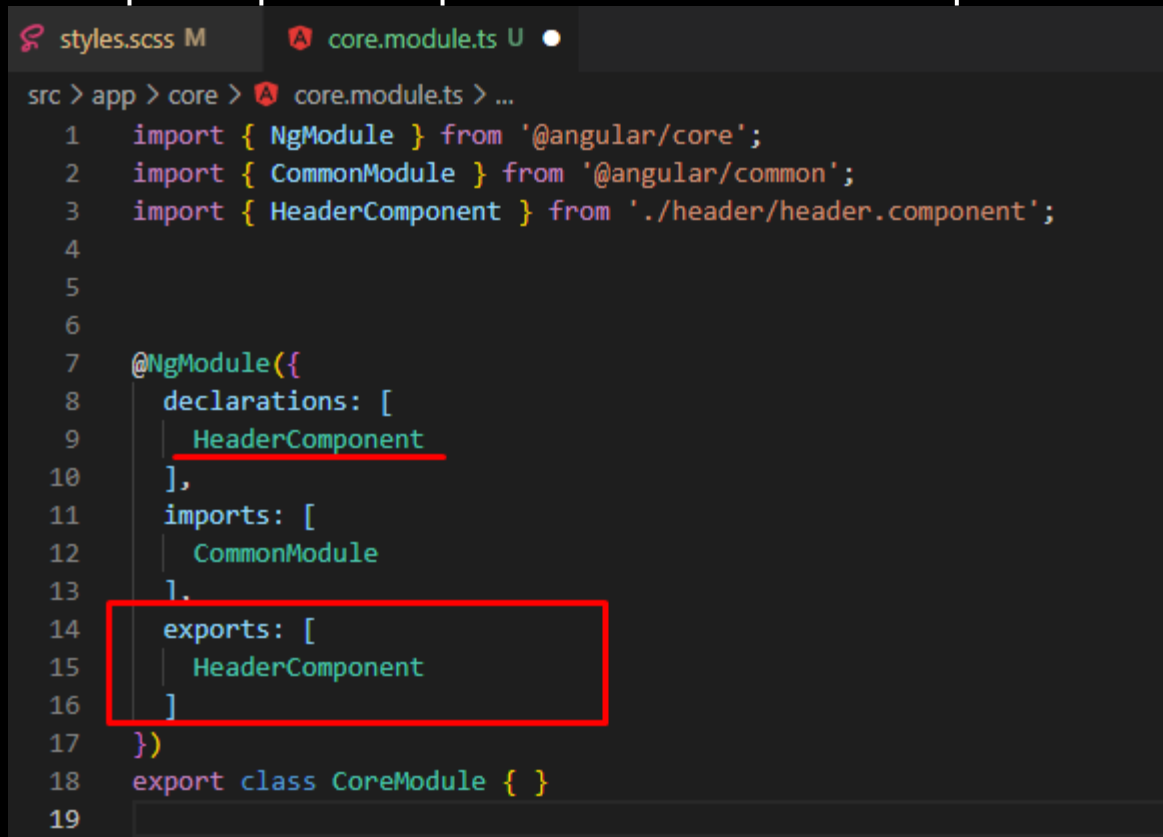
```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate module core
```

- El comando anterior creará el módulo en la carpeta src\app\core de nuestra aplicación. Para crear el componente de encabezado, usaremos el mismo comando, pasando un conjunto diferente de opciones:

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate component header --path=src/app/core --module=core
CREATE src/app/core/header/header.component.html (21 bytes)
CREATE src/app/core/header/header.component.spec.ts (626 bytes)
CREATE src/app/core/header/header.component.ts (276 bytes)
CREATE src/app/core/header/header.component.scss (0 bytes)
UPDATE src/app/core/core.module.ts (274 bytes)
```



- También declarará HeaderComponent en el módulo principal y lo agregará a su propiedad de export para que otros módulos puedan usarlo:



```
src > app > core > core.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { HeaderComponent } from '../header/header.component';
4
5
6
7  @NgModule({
8    declarations: [
9      HeaderComponent
10   ],
11   imports: [
12     CommonModule
13   ],
14   exports: [
15     HeaderComponent
16   ]
17 })
18 export class CoreModule { }
19
```

- El componente de encabezado debe mostrar los enlaces principales de nuestro blog. Abra el archivo de plantilla header.component.html del componente de encabezado y reemplace su contenido con el siguiente fragmento:

```
header.component.html U X
src > app > core > header > header.component.html > ...
Go to component
1  <nav class="navbar navbar-expand navbar-light bg-light">
2    <div class="container-fluid">
3      <a class="navbar-brand">Mi blog</a>
4      <ul class="navbar-nav me-auto">
5        <li class="nav-item">
6          <a class="nav-link">Artículos</a>
7        </li>
8        <li class="nav-item">
9          <a class="nav-link">Contacto</a>
10       </li>
11     </ul>
12   </div>
13 </nav>
```

- El componente de pie de página se puede utilizar más de una vez en una aplicación Angular. Actualmente, queremos mostrarlo en la página principal de nuestra aplicación. En el futuro, es posible que queramos tenerlo también en una página de inicio de sesión que estará disponible para los visitantes del blog. En tal caso, el componente de pie de página debería ser reutilizable.
- Cuando queremos agrupar componentes que se reutilizarán en toda nuestra aplicación, normalmente creamos un módulo denominado compartido por convención. Use el comando de generación de CLI de Angular para crear el módulo compartido:


```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate module shared  
CREATE src/app/shared/shared.module.ts (192 bytes)
```

- El comando anterior creará el módulo compartido en la carpeta `src\app\shared`. El componente de pie de página ahora se puede crear usando el siguiente comando:

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate component footer --path=src/app/shared --module=shared --export
CREATE src/app/shared/footer/footer.component.html (21 bytes)
CREATE src/app/shared/footer/footer.component.spec.ts (626 bytes)
CREATE src/app/shared/footer/footer.component.ts (276 bytes)
CREATE src/app/shared/footer/footer.component.scss (0 bytes)
UPDATE src/app/shared/shared.module.ts (314 bytes)
```

- El comando anterior creará todos los archivos necesarios del componente de pie de página dentro de la carpeta `src\app\shared\footer`. También agregará `FooterComponent` en las declaraciones y exporta propiedades del módulo compartido:

```
header.component.html U  shared.module.ts U x
src > app > shared > shared.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { FooterComponent } from '../footer/footer.component';
4
5
6
7  @NgModule({
8    declarations: [
9      FooterComponent
10   ],
11   imports: [
12     CommonModule
13   ],
14   exports: [
15     FooterComponent
16   ]
17 })
18 export class SharedModule { }
19
```



- 
- El contenido del componente de pie de página debe contener información de derechos de autor sobre nuestro blog. Veamos cómo agregar esta información a nuestro componente:
 - 1. Abra el archivo de clase de TypeScript `footer.component.ts` del componente de pie de página.
 - Agregue una propiedad `currentDate` en la clase `FooterComponent` e inicialícela en un nuevo objeto `Date`:


footer.component.ts U ●

src > app > shared > footer > footer.component.ts > ...

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-footer',
5    templateUrl: './footer.component.html',
6    styleUrls: ['./footer.component.scss']
7  })
8  export class FooterComponent implements OnInit {
9
10     currentDate = new Date();
11
12     constructor() { }
13
14     ngOnInit(): void {
15     }
16
17 }
```

- Abra el archivo de plantilla footer.component.html del componente de pie de página y reemplace su contenido con lo siguiente:

```
src > app > shared > footer >  footer.component.html > ...  
Go to component  
1 <nav class="navbar fixed-bottom navbar-light bg-light">  
2   <div class="container-fluid">  
3     <p>Copyright @{{currentDate | date: 'y'}}. Todos los derechos reservados xD</p>  
4   </div>  
5 </nav>
```

- 
- El código anterior utiliza la interpolación para mostrar el valor de la propiedad `currentDate` en la pantalla. También utiliza la barra de fecha incorporada para mostrar solo el año de la fecha actual.
 - Los pipes son una característica incorporada de angular que aplica transformaciones en la representación de la vista de una propiedad de componente. El valor subyacente de la propiedad permanece intacto.




Ya hemos creado los componentes esenciales de nuestro blog. Ahora es el momento de mostrarlos en la pantalla:

- Abra el módulo principal de la aplicación, el archivo `app.module.ts`, y agregue `CoreModule` y `SharedModule` en la propiedad de importaciones del decorador `@NgModule`:

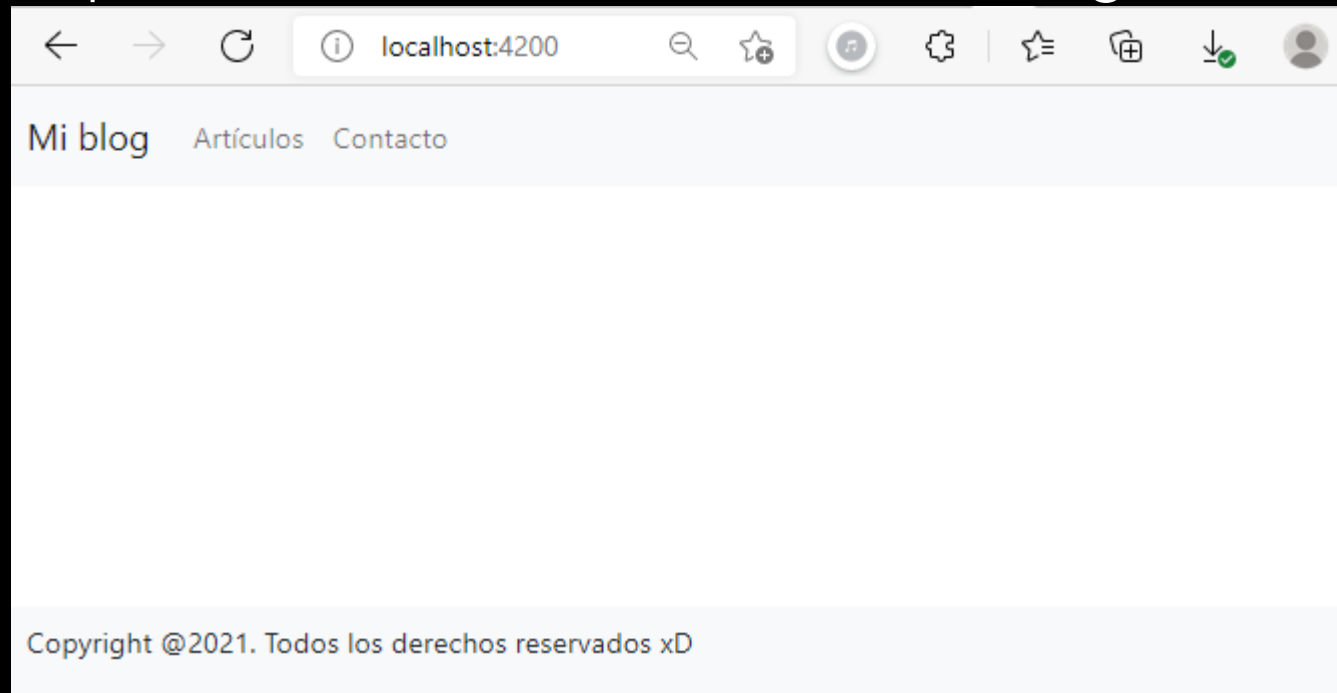
src > app > app.module.ts > AppModule


```
...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { CoreModule } from './core/core.module';
7  import { SharedModule } from './shared/shared.module';
...
8  @NgModule({
9    declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule,
15     CoreModule,
16     SharedModule
17   ],
18   providers: [],
19   bootstrap: [AppComponent]
20 })
21 export class AppModule { }
```

- Abra el archivo de plantilla app.component.html del componente principal y reemplace su contenido con el siguiente fragmento de HTML:

```
src > app >  app.component.html > ...  
Go to component | You, seconds ago | 1 author (You)  
1 | <app-header></app-header>  
2 | <app-footer></app-footer>  
3 |  
.
```


- Agregamos el componente de encabezado y pie de página en el fragmento anterior usando sus selectores de CSS.
- Si ejecutamos el comando serve de Angular CLI para obtener una vista previa de la aplicación, deberíamos obtener lo siguiente:



- 
- Ya hemos completado el diseño básico de nuestra aplicación de blog, ¡y se ve genial! Pero el encabezado contiene dos enlaces adicionales que aún no hemos cubierto. Ahora aprenderemos cómo usar el enrutamiento para activar esos enlaces..

CONFIGURANDO EL ENRUTAMIENTO PARA NUESTRA APLICACIÓN

- El componente de encabezado que creamos en la sección anterior contiene dos enlaces:
 - Artículos: muestra una lista de artículos de blog.
 - Contacto: muestra información personal sobre el propietario del blog.
- Los enlaces anteriores también se convertirán en las principales características de nuestra aplicación. Entonces, necesitamos crear un módulo angular para cada uno.
- Cuando diseñe su sitio web y necesite decidir sobre los módulos que utilizará, consulte el menú principal del sitio web. Cada enlace del menú debe ser una característica diferente y, por lo tanto, un módulo angular diferente.
- Por convención, los módulos que contienen funcionalidad para una característica específica se denominan módulos de características (feature modules).

CREANDO LA PÁGINA DE CONTACTO


- Comencemos creando nuestra función de contacto primero:

1. Cree un módulo para la función de contacto:


```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate module contact  
CREATE src/app/contact/contact.module.ts (193 bytes)
```

2. Cree un componente que será el componente principal del módulo de contacto:

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate component contact --path=src/app/contact --module=contact --export --flat  
CREATE src/app/contact/contact.component.html (22 bytes)  
CREATE src/app/contact/contact.component.spec.ts (633 bytes)  
CREATE src/app/contact/contact.component.ts (280 bytes)  
CREATE src/app/contact/contact.component.scss (0 bytes)  
UPDATE src/app/contact/contact.module.ts (312 bytes)
```

- 
- Pasamos la opción `--flat` al comando `generate` para que Angular CLI no cree una carpeta separada para nuestro componente, como en casos anteriores. El componente de contacto será el único componente de nuestro módulo, por lo que no tiene sentido tenerlo por separado.


- 3. Abra el archivo contact.component.html y agregue el siguiente contenido HTML:

```
src > app > contact >  contact.component.html > ...  
Go to component  
1 <div class="card mx-auto text-center border-light" style="width: 18rem;">  
2     
3   <div class="card-body">  
4     <h5 class="card-title">Mi blog</h5>  
5     <p class="card-text">  
6       Un blog personal creado con Angular y el generador de sitios estáticos de Scully  
7     </p>  
8     <a href="https://angular.io/" target="_blank" class="card-link">Angular</a>  
9     <a href="https://scully.io/" target="_blank" class="card-link">Scully</a>  
10  </div>  
11 </div>
```



- Ya hemos creado nuestra función de contacto. El siguiente paso es agregarlo a la página principal de nuestra aplicación Angular:
1. Abra el archivo `app-routing.module.ts` y agregue un nuevo objeto de configuración de ruta en la propiedad de rutas:


```
src > app > app-routing.module.ts > ...  
You, seconds ago | 1 author (You)  
1 import { NgModule } from '@angular/core';  
2 import { RouterModule, Routes } from '@angular/router';  
3 import { ContactComponent } from '../contact/contact.component';  
4 const routes: Routes = [  
5   { path: 'contact', component: ContactComponent }  
6 ];  
7  
You, 20 hours ago | 1 author (You)  
8 @NgModule({  
9   imports: [RouterModule.forRoot(routes)],  
10  exports: [RouterModule]  
11 })  
12 export class AppRoutingModule { }
```


- 
- El código anterior indica que cuando la URL del navegador apunta a la ruta del contacto, nuestra aplicación se activará y mostrará ContactComponent en la pantalla.
 - La propiedad de rutas de un módulo de enrutamiento contiene la configuración de enrutamiento del módulo de funciones respectivo.
 - Es un arreglo de objetos de configuración de ruta donde cada uno define la clase de componente y la ruta URL que lo activa.

- 2. Agregue ContactModule en la matriz de importaciones del decorador @NgModule de AppModule para poder usarlo:


```
src > app > app.module.ts > AppModule
...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { ContactModule } from './contact/contact.module';
7  import { CoreModule } from './core/core.module';
8  import { SharedModule } from './shared/shared.module';
9
10 @NgModule({
11   declarations: [
12     AppComponent
13   ],
14   imports: [
15     BrowserModule,
16     AppRoutingModule,
17     CoreModule,
18     SharedModule,
19     ContactModule
20   ],
21   providers: [],
22   bootstrap: [AppComponent]
23 })
24 export class AppModule { }
```

3. Los componentes enrutados, al igual que ContactComponent, necesitan un lugar donde puedan cargarse. Abra el archivo app.component.html y agregue la directiva router-outlet:

```
src > app >  app.component.html > ...  
Go to component | ...  
1 | <app-header></app-header>  
2 | <div class="container">  
3 | | <router-outlet></router-outlet>  
4 | </div>  
5 | <app-footer></app-footer>  
6 |
```

- Ahora, necesitamos conectar la configuración de ruta que creamos con el enlace real en el componente de encabezado:
1. Abra el archivo header.component.html y agregue la directiva routerLink al elemento HTML de anclaje respectivo:

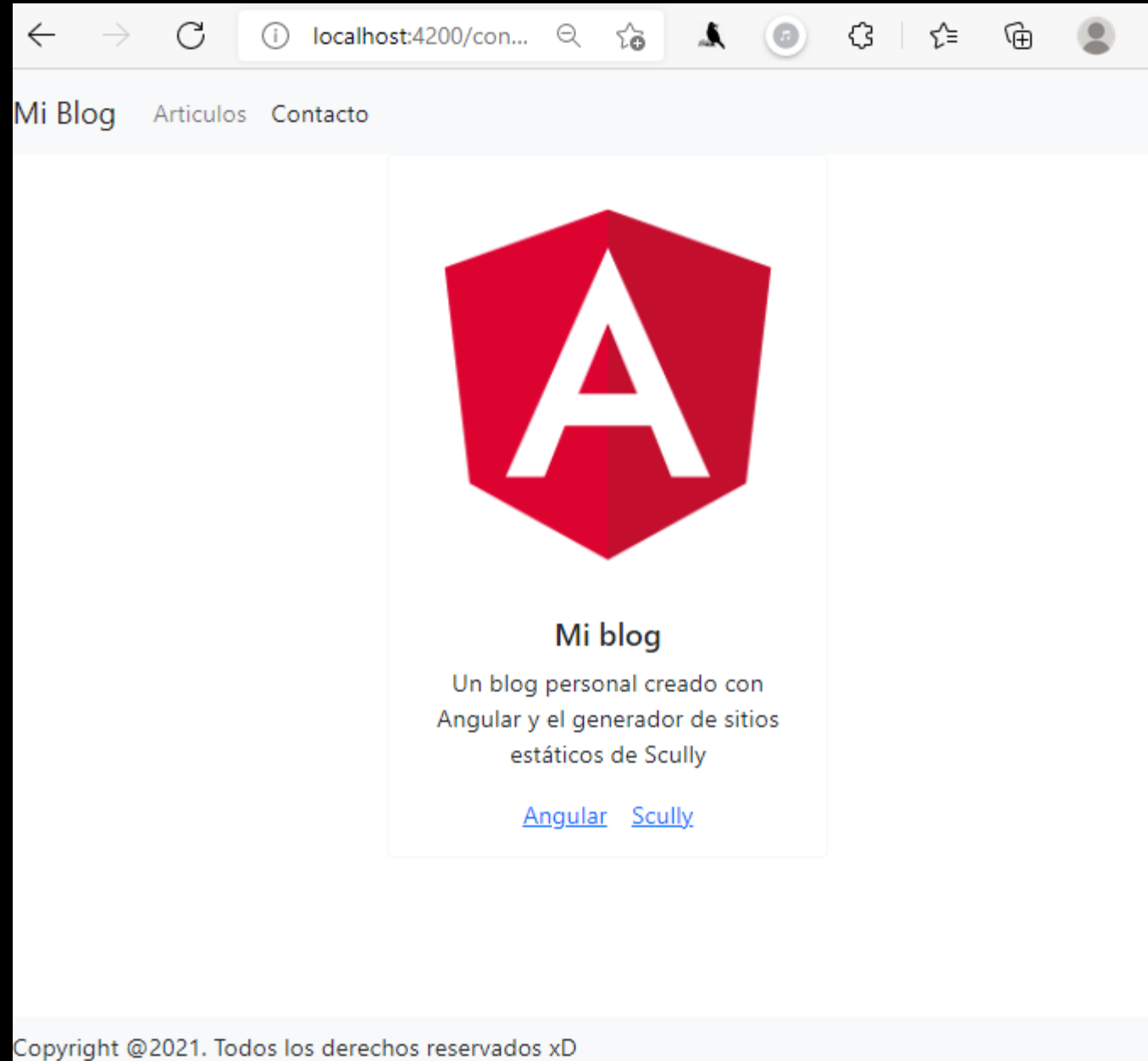
```
src > app > core > header > header.component.html > ...  
Go to component  
1 <nav class="navbar navbar-expand navbar-light bg-light">  
2   <div class="container-fluid">  
3     <a class="navbar-brand">Mi Blog</a>  
4     <ul class="navbar-nav me-auto">  
5       <li class="nav-item">  
6         <a routerLink="/articles" routerLinkActive="active" class="nav-link">Articulos</a>  
7       </li>  
8       <li class="nav-item">  
9         <a routerLink="/contact" routerLinkActive="active" class="nav-link">Contacto</a>  
10      </li>  
11    </ul>  
12  </div>  
13 </nav>
```

- 
- En el fragmento anterior, la directiva `routerLink` apunta a la propiedad de ruta del objeto de configuración de ruta.
 - También hemos agregado la directiva `routerLinkActive`, que establece la clase activa en el elemento de anclaje cuando se activa la ruta específica.
 - Observe que el valor de la directiva `routerLink` contiene un `/`, mientras que la propiedad de ruta del objeto de configuración de ruta que definimos no lo hace. Según el caso, omitir el `/` daría un significado diferente a la ruta

- 2. Las directivas routerLink y routerLinkActive son parte del paquete del enrutador angular. Necesitamos importar RouterModule en el módulo principal para usarlos

```
src > app > core > core.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { HeaderComponent } from './header/header.component';
4  import { RouterModule } from '@angular/router';
5
6  @NgModule({
7    declarations: [
8      HeaderComponent
9    ],
10   imports: [
11     CommonModule,
12     RouterModule
13   ],
14   exports: [
15     HeaderComponent
16   ]
17 })
18 export class CoreModule { }
19
```

- ¡Ahora estamos listos para obtener una vista previa de nuestra nueva página de contacto! Si ejecutamos la aplicación usando ng serve y hacemos clic en el enlace de Contacto, deberíamos ver el siguiente resultado



AGREGAR LA PÁGINA DE ARTÍCULOS


- La característica que se encarga de mostrar los artículos en nuestro blog será el módulo de artículos. También será el módulo que conecta los puntos entre Angular y Scully. Usaremos el comando generate de Angular CLI para crear ese módulo:

- En el comando anterior, pasamos algunas opciones de enrutamiento adicionales:
 - `--route`: define la ruta URL de nuestra función
 - `--module`: Indica el módulo de enrutamiento que definirá el objeto de configuración de ruta que activa nuestra característica

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate module articles --route=articles --module=app-routing
CREATE src/app/articles/articles-routing.module.ts (351 bytes)
CREATE src/app/articles/articles.module.ts (371 bytes)
CREATE src/app/articles/articles.component.html (23 bytes)
CREATE src/app/articles/articles.component.spec.ts (640 bytes)
CREATE src/app/articles/articles.component.ts (284 bytes)
CREATE src/app/articles/articles.component.scss (0 bytes)
UPDATE src/app/app-routing.module.ts (475 bytes)
```

- Angular CLI realiza acciones adicionales, en lugar de simplemente crear el módulo, al ejecutar el comando:
 - Crea un componente enrutado en la carpeta `src\app\articles` que se activará por defecto desde un objeto de navegación de ruta. Es la página de inicio de nuestra función y mostrará una lista de publicaciones de blog.
 - Crea un módulo de enrutamiento, llamado `articles-routing.module.ts`, que contiene la configuración de enrutamiento del módulo de artículos.
 - Agrega un nuevo objeto de configuración de ruta en la configuración de ruta del módulo principal de la aplicación que activa el módulo de artículos.


- El archivo `articles-routing.module.ts` contiene la configuración de enrutamiento para el módulo de artículos:

```
src > app > articles >  articles.module.ts > ...  
1  import { NgModule } from '@angular/core';  
2  import { CommonModule } from '@angular/common';  
3  
4  import { ArticlesRoutingModule } from './articles-routing.module';  
5  import { ArticlesComponent } from './articles.component';  
6  import { RouterModule, Routes } from '@angular/router';  
7  
8  const routes: Routes = [{ path: '', component:  
9  ArticlesComponent }];  
10 @NgModule({  
11   declarations: [  
12     ArticlesComponent  
13   ],  
14   imports: [  
15     CommonModule,  
16     ArticlesRoutingModule,  
17     RouterModule.forChild(routes)  
18   ],  
19   exports: [RouterModule]  
20 })  
21 export class ArticlesModule { }
```

- Importa RouterModule usando el método forChild para pasar la configuración de enrutamiento al enrutador angular. Si echamos un vistazo al módulo de enrutamiento principal de la aplicación, veremos que sigue un enfoque ligeramente diferente:


```
src > app > app-routing.module.ts > ...  
  
1  import { NgModule } from '@angular/core';  
2  import { RouterModule, Routes } from '@angular/router';  
3  import { ContactComponent } from '../contact/contact.component';  
4  const routes: Routes = [  
5    { path: 'contact', component: ContactComponent },  
6    { path: 'articles', loadChildren: () => import('../articles/articles.module').then(m => m.ArticlesModule) }  
7  ];  
8  
9  @NgModule({  
10    imports: [RouterModule.forRoot(routes)],  
11    exports: [RouterModule]  
12  })  
13  export class AppRoutingModule { }
```









- El método `forChild` se usa en módulos de características, mientras que el método `forRoot` debe usarse solo en el módulo principal de la aplicación.
- La configuración de ruta del módulo de artículos contiene solo una ruta que activa `ArticlesComponent`. La ruta de la ruta (xD) se establece en una cadena vacía para indicar que es la ruta predeterminada del módulo de enrutamiento. Básicamente significa que `ArticlesComponent` se activará siempre que se cargue ese módulo. Pero, ¿cómo se carga el módulo de artículos en nuestra aplicación?
- La segunda ruta del módulo de enrutamiento principal contiene un objeto de configuración de ruta que no activa un componente sino un módulo. Utiliza el método `loadChildren` para cargar `ArticlesModule` dinámicamente cuando la navegación activa la ruta de los artículos.

- 
- El enfoque anterior se llama carga diferida y mejora el inicio y el rendimiento general de una aplicación Angular. Crea un paquete separado para cada módulo de carga diferida, que se carga a pedido, lo que reduce el tamaño del paquete final y el consumo de memoria de su aplicación. Conectemos la nueva ruta a nuestro componente de encabezado:

- 1. Abra el archivo header.component.html y agregue las siguientes directivas routerLink y routerLinkActive al elemento HTML del ancla de artículos:

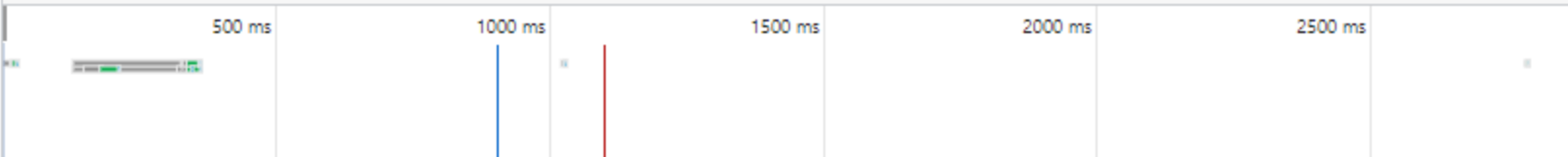
```
header.component.html U ●
src > app > core > header > header.component.html > nav.navbar.navbar-expand.navbar-light.bg-light > div.container-flu
Go to component
1 <nav class="navbar navbar-expand navbar-light bg-light">
2   <div class="container-fluid">
3     <a class="navbar-brand">Mi Blog</a>
4     <ul class="navbar-nav me-auto">
5       <li class="nav-item">
6         <li class="nav-item">
7           <a routerLink="/articles" routerLinkActive="active" class="nav-link">Articulos</a>
8         </li>
9         <li class="nav-item">
10          <a routerLink="/contact" routerLinkActive="active" class="nav-link">Contacto</a>
11        </li>
12      </ul>
13    </div>
14  </nav>
```

- 
2. Ejecute ng serve y use su navegador favorito para obtener una vista previa de su aplicación.
 3. Abra las herramientas de desarrollo de su navegador, haga clic en el enlace Artículos e inspeccione la pestaña Red:






☐ Conservar el registro
 ☐ Deshabilitar la memoria caché
 Sin limitación
 




☐ Ocultar las URL de datos

Fetch/XHR JS CSS IMG Multimedia Fuente Documento WS Wasm Manifiesto Otros ☐ Ha bloqueado las cookies

☐ Ha bloqueado las cookies

Nombre	Estado	Tipo	Iniciador	Tamaño	Hora	Cascada
localhost	304	document	Otros	210 B	13 ms	
runtime.js	304	script	(índice)	211 B	63 ms	
polyfills.js	304	script	(índice)	212 B	27 ms	
styles.js	304	script	(índice)	212 B	32 ms	
vendor.js	304	script	(índice)	213 B	32 ms	
main.js	304	script	(índice)	211 B	32 ms	
info?t=1628864512560	200	xhr	zone.js:2863	368 B	2 ms	
websocket	101	websocket	sockjs.js:1687	0 B	Pendiente	
src_app_articles_articles_module_ts.js	304	script	load script:41	211 B	3 ms	

- 
- Entre otras solicitudes, debería ver uno llamado `articles-articles-module.js`. Es el paquete del módulo de artículos de carga diferida que se cargó cuando hizo clic en el enlace Artículos.

- Ahora estamos listos para convertir nuestra aplicación Angular en un sitio web de blog profesional. Antes de continuar, agreguemos algunas rutas adicionales al archivo app-routing.module.ts:

```
4  const routes: Routes = [  
5    { path: 'contact', component: ContactComponent },  
6    {  
7      path: 'articles',  
8      loadChildren: () =>  
9        import('./articles/articles.module').then(m => m.ArticlesModule),  
10   },  
11   { path: '', pathMatch: 'full', redirectTo: 'articles' },  
12   { path: '**', redirectTo: 'articles' },  
13 ];  
14
```

- Agregamos una ruta predeterminada para redirigir automáticamente a los usuarios de nuestro blog a la ruta de los artículos al visitar el blog. Además, creamos un nuevo objeto de configuración de ruta con su ruta establecida en ** que también navega a la ruta de los artículos. La sintaxis ** se denomina ruta comodín y se activa cuando el enrutador no puede hacer coincidir una URL solicitada con una ruta definida.
- Siempre defina primero las rutas más específicas y luego agregue las genéricas, como las rutas predeterminadas y las rutas comodín. El enrutador angular analiza la configuración de la ruta en el orden que definimos.
- Ya hemos habilitado y configurado el enrutamiento en nuestra aplicación Angular. Ahora vamos a agregar capacidades de blogs a nuestra aplicación.

AÑADIENDO CAPACIDADES DE BLOG CON SCULLY

- Nuestra aplicación actualmente no tiene ninguna lógica específica con respecto a las publicaciones de blog. Es una aplicación Angular típica que usa enrutamiento. Sin embargo, al agregar una configuración de enrutamiento, hemos establecido las bases para agregar soporte para blogs usando Scully.
- Scully necesita al menos una ruta definida en una aplicación Angular para funcionar correctamente.
- Primero, necesitamos instalar Scully en nuestra aplicación.

INSTALACIÓN DE LA BIBLIOTECA DE SCULLY

- Usaremos el comando add de Angular CLI para instalar Scully en nuestra aplicación Angular:

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng add @scullyio/init
i Using package manager: npm
✓ Found compatible package version: @scullyio/init@1.1.4.
✓ Package information loaded.

The package @scullyio/init@1.1.4 will be installed and executed.
Would you like to proceed? Yes
```

- Importa ScullyLibModule en nuestro módulo de aplicación principal:

```
9  import { ScullyLibModule } from '@scullyio/ng-lib';
10  @NgModule({
11    declarations: [
12      AppComponent
13    ],
14    imports: [
15      BrowserModule,
16      AppRoutingModule,
17      CoreModule,
18      SharedModule,
19      ContactModule,
20      ScullyLibModule
21    ],
22    providers: [],
23    bootstrap: [AppComponent]
24  })
25  export class AppModule { }
26
```

- ScullyLibModule es el módulo principal de la biblioteca de Scully que contiene varios servicios y directivas angular que Scully necesitará.
- También crea un archivo de configuración para la biblioteca Scully en la carpeta raíz del espacio de trabajo de Angular CLI:

```
scully.miBlog.config.ts > ...  
1  import { ScullyConfig } from '@scullyio/scully';  
2  export const config: ScullyConfig = {  
3    projectRoot: "./src",  
4    projectName: "miBlog",  
5    outDir: './dist/static',  
6    routes: {  
7    }  
8  };
```

- El archivo de configuración contiene información sobre nuestra aplicación Angular que Scully necesitará en el camino:
 - projectRoot: la ruta que contiene el código fuente de la aplicación Angular
 - projectName: el nombre de la aplicación Angular
 - outDir: la ruta de salida de los archivos generados por Scully
 - routes: contiene la configuración de la ruta que se utilizará para acceder a las publicaciones de nuestro blog. Scully lo completará automáticamente, como veremos en la siguiente sección.

La ruta de salida de Scully debe ser diferente de la ruta en la que la CLI de Angular genera el paquete de su aplicación Angular. Este último se puede configurar desde el archivo angular.json.

INICIALIZANDO NUESTRA PÁGINA DE BLOG

- Scully proporciona un esquema Angular CLI específico para inicializar una aplicación Angular, como un blog, mediante el uso de archivos Markdown (.md):

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate @scullyio/init:markdown
```

- El comando anterior iniciará el proceso de configuración de nuestro blog pasando por una lista de preguntas (los valores predeterminados se muestran entre paréntesis):
- 1. Le pondremos posts

```
? What name do you want to use for the module? (blog) _
```

- 2.- El slug es un identificador único para cada publicación y se define en el objeto de configuración de ruta del módulo. Default

```
? What slug do you want for the markdown file? id
```

- 3.- Ingrese mdfiles como la ruta que Scully usará para almacenar nuestros archivos de publicación de blog reales. Esto creará una carpeta mdfiles dentro de la ruta raíz de nuestro proyecto Angular CLI. De forma predeterminada, también creará una publicación de blog para nuestra conveniencia. Aprenderemos a crear uno propio en la sección Visualización de datos del blog en la página de inicio. Default


```
? Where do you want to store your markdown files?
```


- 4.- Escriba publicaciones como el nombre de la ruta para acceder a las publicaciones de nuestro blog. Default
- Nombre de la ruta del blog. El nombre de la ruta es la propiedad de ruta del objeto de configuración de ruta que se creará

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> ng generate @scullyio/init:markdown
"@scullyio/init:markdown" schema is using the keyword "id" which its support is deprecated. Use "$id" for schema ID.
"@scullyio/init:markdown" schema is using the keyword "id" which its support is deprecated. Use "$id" for schema ID.
? What name do you want to use for the module? posts
? What slug do you want for the markdown file? id
? Where do you want to store your markdown files?
? Under which route do you want your files to be requested?
  @@ Update scully.miBlog.config.ts
CREATE src/app/posts/posts-routing.module.ts (409 bytes)
CREATE src/app/posts/posts.component.html (153 bytes)
CREATE src/app/posts/posts.component.spec.ts (621 bytes)
CREATE src/app/posts/posts.component.ts (494 bytes)
CREATE src/app/posts/posts.component.scss (131 bytes)
CREATE src/app/posts/posts.module.ts (387 bytes)
UPDATE scully.miBlog.config.ts (283 bytes)
UPDATE src/app/app-routing.module.ts (689 bytes)
"@scullyio/init:post" schema is using the keyword "id" which its support is deprecated. Use "$id" for schema ID.
  @@ Blog ./posts/2021-08-13-posts.md file created
CREATE posts/2021-08-13-posts.md (99 bytes)
PS C:\Users\Geovany\Desktop\Angular\miBlog>
```

- scully realiza varias acciones al ejecutar los comandos anteriores, incluida la creación de la configuración de enrutamiento del módulo de publicaciones:

```
posts-routing.module.ts U x
src > app > posts > posts-routing.module.ts > ...
1  import {NgModule} from '@angular/core';
2  import {Routes, RouterModule} from '@angular/router';
3
4  import {PostsComponent} from './posts.component';
5
6  const routes: Routes = [
7    {
8      path: ':id',
9      component: PostsComponent,
10   },
11   {
12     path: '**',
13     component: PostsComponent,
14   }
15 ];
16
17 @NgModule({
18   imports: [RouterModule.forChild(routes)],
19   exports: [RouterModule],
20 })
21 export class PostsRoutingModule {}
```

- 
- En la primera ruta se establece en: id y activa PostsComponent. El carácter de dos puntos indica que id es un parámetro de ruta. El parámetro id está relacionado con la propiedad slug definida anteriormente en la configuración de Scully.
 - Scully trabaja creando una ruta para cada publicación de blog que creamos. Utiliza la configuración de ruta del módulo de publicaciones y el módulo de aplicación principal para construir la propiedad de rutas en el archivo de configuración de Scully:


scully.miBlog.config.ts U X

scully.miBlog.config.ts > ...

```
1 import { ScullyConfig } from '@scullyio/scully';
2 export const config: ScullyConfig = {
3   projectRoot: "./src",
4   projectName: "miBlog",
5   outDir: './dist/static',
6   routes: {
7     '/posts/:id': {
8       type: 'contentFolder',
9       id: {
10        folder: "./posts"
11      }
12    },
13  },
14 };
```

- PostsComponent es el componente angular que se utiliza para representar los detalles de cada publicación de blog. El archivo de plantilla del componente se puede personalizar aún más según sus necesidades:


```
posts.component.html U x
src > app > posts > posts.component.html > h3
1  <h3>ScullyIo content</h3>
2  <hr>
3
4  <!-- This is where Scully will inject the static HTML -->
5  <scully-content></scully-content>
6  <hr>
7  <h4>End of content</h4>
8
```

- 
- Puede personalizar todo el contenido del archivo de plantilla anterior, excepto la línea `<scully content> </scully-content>`, que Scully utiliza internamente.
 - En este punto, hemos completado la instalación y configuración de Scully en nuestra aplicación Angular. Ahora pondremos Angular y Scully para cooperar y mostrar publicaciones de blog en nuestra aplicación Angular.


VISUALIZACIÓN DE PUBLICACIONES DE BLOG EN LA PÁGINA DE INICIO

- Nos gustaría que nuestros usuarios vean la lista de publicaciones de blog disponibles tan pronto como lleguen a nuestro sitio web de blogs. Según la ruta de ruta predeterminada que hemos definido, ArticlesComponent es la página de destino de nuestro blog.
- Scully proporciona ScullyRoutesService, un servicio que podemos usar en nuestros componentes para obtener información sobre las rutas que creará de acuerdo con las publicaciones del blog.

- Pongamos este servicio en acción en nuestra página de destino:
 1. Navegue hasta el archivo de clase de TypeScript `articles.component.ts`.
 2. Importe `ScullyRoute` y `ScullyRoutesService` desde el paquete `@scullyio/ng-lib`:

```
src > app > articles >  articles.component.ts > ...  
1  import { Component, OnInit } from '@angular/core';  
2  import { ScullyRoute, ScullyRoutesService } from '@scullyio/ng-lib';  
3  |
```


3. Inyecte `ScullyRoutesService` en el constructor de la clase `ArticlesComponent`:
4. Cree una propiedad de componente del tipo de matriz `ScullyRoute`:


```
9   export class ArticlesComponent implements OnInit {  
10    posts: ScullyRoute[] = [];  
11    constructor(private scullyService: ScullyRoutesService) {  
12    }  
13  }
```



5. Edite el método `ngOnInit` del componente y agregue el siguiente código:

```
ngOnInit(): void {  
  this.scullyService.available$.subscribe((posts) => {  
    this.posts = posts.filter((post) => post.title);  
  });  
}
```

6. Abra el archivo `articles.component.html` y agregue el siguiente código HTML:

```
src > app > articles >  articles.component.html > ...  
Go to component  
1 <div class="list-group mt-3">  
2   <a *ngFor="let post of posts" [routerLink]="post.route" class="list-group-item  
3   list-group-item-action">  
4     <div class="d-flex w-100 justify-content-between">  
5       <h5 class="mb-1">{{post.title}}</h5>  
6     </div>  
7     <p class="mb-1">{{post.description}}</p>  
8   </a>  
9 </div>
```

- 
- Hay muchas técnicas involucradas en los pasos anteriores, así que vamos a desglosarlas pieza por pieza.
 - Cuando queremos usar un servicio Angular en un componente, solo necesitamos referenciarlo. ¿Cómo? Agregándolo como una propiedad en el constructor del componente. El componente no necesita saber nada sobre cómo se implementa el servicio.
 - El método `ngOnInit` es parte de la interfaz `OnInit`, que es implementada por nuestro componente. Es llamado por el framework Angular cuando se inicializa un componente y nos proporciona un hook para agregar lógica personalizada que se ejecutará.

- 
- La propiedad `available$` de `ScullyRoutesService` se denomina observable.
 - Para recuperar su valor, debemos suscribirnos a él. La variable de publicaciones devueltas contendrá todas las rutas disponibles que se generaron desde Scully.
 - Scully se ejecuta en todas las rutas de nuestra aplicación Angular. Para evitar mostrar rutas distintas a las relacionadas con publicaciones de blog, como la ruta de contacto, filtramos los resultados de la propiedad `available$`.

- Cuando nos suscribimos a un observable, debemos cancelar la suscripción cuando nuestro componente ya no existe. De lo contrario, podemos experimentar pérdidas de memoria en nuestra aplicación. Veamos cómo podemos lograr esta tarea usando otro hook de ciclo de vida del componente llamado `ngOnDestroy`:
 1. Declare una propiedad `routeSub` privada del tipo `Subscription` en la clase `ArticlesComponent`. La suscripción se puede importar desde el paquete `rxjs npm`.
 2. Establezca el valor de retorno observable `available$` en la propiedad `routeSub`.
 3. Agregue la interfaz `OnDestroy` a la lista de interfaces implementadas del componente. `OnDestroy` se puede importar desde el paquete `@angular/core`. Se ejecuta cuando se destruye el componente y ya no se representa en la pantalla.
 4. Implemente el método `ngOnDestroy` y llame al método `unsubscribe` de la propiedad `routeSub` en el cuerpo del método.

src > app > articles >  articles.component.ts > ...

```
1  import { Component, OnInit, OnDestroy } from '@angular/core';
2  import { ScullyRoute, ScullyRoutesService } from '@scullyio/ng-lib';
3  import { Subscription } from 'rxjs';
4
5  @Component({
6    selector: 'app-articles',
7    templateUrl: './articles.component.html',
8    styleUrls: ['./articles.component.scss'],
9  })
10 export class ArticlesComponent implements OnInit, OnDestroy {
11   posts: ScullyRoute[] = [];
12   private routeSub: Subscription | undefined;
13   constructor(private scullyService: ScullyRoutesService) {}
14
15   ngOnInit(): void {
16     this.routeSub =
17     this.scullyService.available$.subscribe((posts) => {
18       this.posts = posts.filter((post) => post.title);
19     });
20   }
21
22   ngOnDestroy(): void {
23     this.routeSub?.unsubscribe();
24   }
25 }
```

- En la plantilla de nuestro componente, usamos la directiva incorporada `*ngFor` para iterar sobre la matriz de publicaciones dentro de HTML. Luego podemos acceder a cada elemento de la matriz usando la variable de referencia de la plantilla de publicación y usar la interpolación para mostrar el título y la descripción.
- Finalmente, agregamos una directiva `routerLink` a cada elemento de anclaje para navegar a la publicación de blog respectiva cuando se hace clic. Observe que `routerLink` está rodeado por []. La sintaxis [] se llama vinculación de propiedad y la usamos cuando queremos vincular la propiedad de un elemento HTML a una variable. En nuestro caso, vinculamos la directiva `routerLink` a la propiedad `route` de la variable de referencia de la plantilla de publicación.

- Ahora que finalmente hemos completado todas las piezas del rompecabezas, podemos ver el sitio web de nuestro blog en acción:
1. Ejecute el comando build de Angular CLI para construir nuestra aplicación Angular:
 - `ng build`
 2. Ejecute el siguiente comando npm para construir Scully y generar nuestras rutas de blog:
 - `npm run scully`
 - El comando anterior creará un archivo `scully-routes.json` dentro de la carpeta `src\assets`. Contiene las rutas de nuestra aplicación Angular y se necesita por Scully runtime.


3. Ejecute el siguiente comando npm para servir nuestro blog:

- `npm run scully:serve`
- El comando anterior iniciará dos servidores web: uno que contiene la versión pre-renderizada estática de nuestro sitio web creado con Scully y otro que es la versión en vivo de Angular de nuestra aplicación:

```
PS C:\Users\Geovany\Desktop\Angular\miBlog> npm run scully:serve

> mi-blog@0.0.0 scully:serve C:\Users\Geovany\Desktop\Angular\miBlog
> npx scully serve --

using plugins from folder "./scully"
starting static server
Scully static server started on "http://localhost:1668/"
Angular distribution server started on "http://localhost:1864/"
```


- 
- Si abrimos nuestro navegador y navegamos hasta `http://localhost:1668`, no veremos ninguna publicación en el blog. ¿Porqué es eso?
 - Una publicación de blog creada con Scully no se devuelve en la propiedad `available$` de `ScullyRoutesService` a menos que la publiquemos. Para publicar una publicación de blog, hacemos lo siguiente.

1. Navega hasta la carpeta post que creó Scully y abre el único archivo .md que encontrarás. El nombre y el contenido pueden diferir de su archivo porque se basa en la fecha de creación de Scully:

Scully ha definido un conjunto de propiedades entre las líneas ---- que representan los metadatos sobre la publicación del blog. También puede agregar los suyos propios como pares clave-valor.

```
posts > M+ 2021-08-13-posts.md > [abc] # 2021-08-13-posts
1  ---
2  title: 2021-08-13-posts
3  description: 'blog description'
4  published: false
5  slugs:
6  | | - ____UNPUBLISHED____kttf3tru_cyAiAcXnj4rN3t8qKmg9ZsbGeHRKgOnf
7  ---
8
9  # 2021-08-13-posts
```

2. Elimine la propiedad `slug` y establezca la propiedad `published` en `true`:

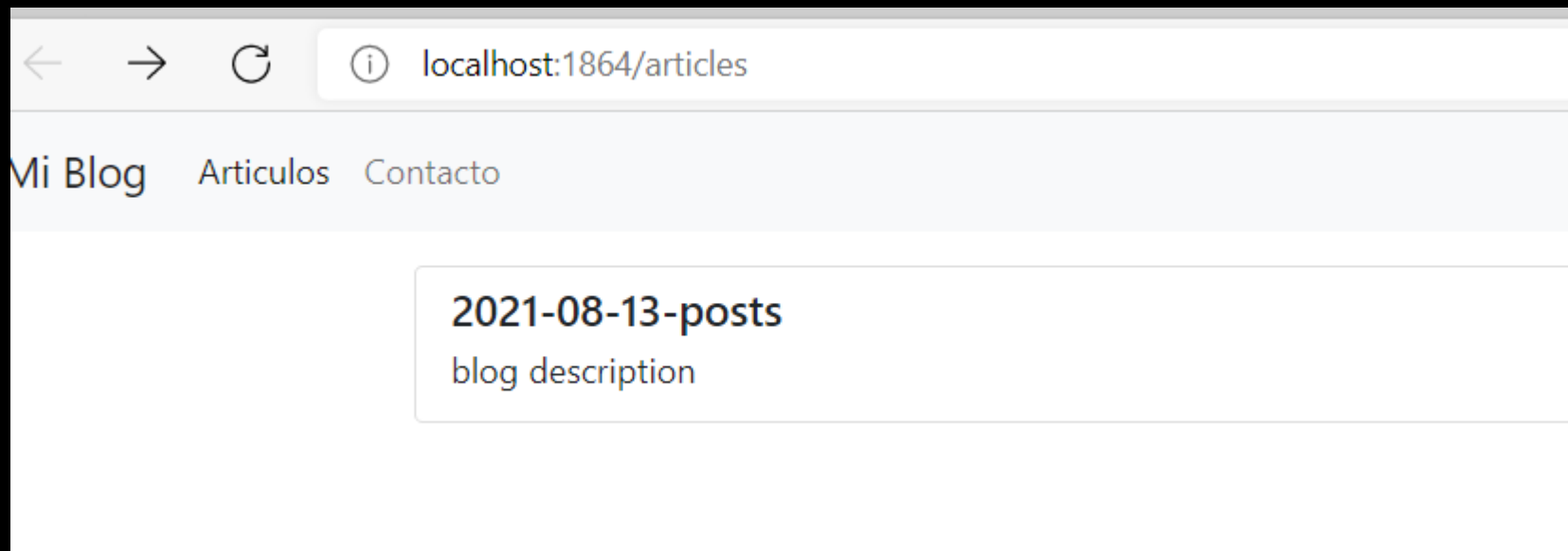
```
posts > M+ 2021-08-13-posts.md > [abc] # 2021-08-13-posts
1  ---
2  title: 2021-08-13-posts
3  description: 'blog description'
4  published: true
5  ---
6
7  # 2021-08-13-posts
```


3. Ejecute el siguiente comando para obligar a Scully a regenerar las rutas de nuestra aplicación:

3. `npm run scully`

Necesitamos ejecutar el comando anterior cada vez que hacemos un cambio en los archivos relacionados con nuestro blog.

4. Ejecute el comando `npm run scully:serve` y navegue para obtener una vista previa del sitio web generado.
4. `npm run scully:serve`



- 
- Ahora podemos ver una publicación de blog, la predeterminada que se creó cuando instalamos Scully. Creemos otro:
1. Ejecute el siguiente comando generate de Angular CLI:
 - `ng generate @scullyio/init:post --name="Angular y Scully"`
 2. Scully creará un archivo Markdown llamado `angular-y-scully.md` dentro de la carpeta especificada. Abra ese archivo y actualice su contenido para que sea el mismo que se muestra a continuación:



```
blog > M+ angular-y-scully.md > # Angular y Scully
```

```
1  ---
2  title: Angular y Scully
3  description: 'Cómo construir un blog con Angular y Scully'
4  published: true
5  ---
6
7  # Angular y Scully
8  Angular es un framework de JavaScript robusto que podemos usar para crear aplicaciones web excelentes y de alto rendimiento.
9  Scully es un popular generador de sitios web estáticos que potencia Angular con características Jamstack.
10 Puede encontrar más sobre ellos en los siguientes enlaces:
11 - https://angular.io
12 - https://scully.io
13 - https://www.jamstack.org
```

- Ejecute `npm run scully` para crear una ruta para la publicación de blog recién creada. Scully también actualizará el archivo `scully-routes.json` con la nueva ruta.



Scullylo content


Angular y Scully

Angular es un framework de JavaScript robusto que podemos usar para crear aplicaciones web excelentes y de alto rendimiento. Scully es un popular generador de sitios web estáticos que potencia Angular con características Jamstack. Puede encontrar más sobre ellos en los siguientes enlaces:

- <https://angular.io>
- <https://scully.io>
- <https://www.jamstack.org>

End of content

- Para verificar eso, navegue a la carpeta dist de su proyecto Angular, donde encontrará dos carpetas:
 - mi-blog: Contiene la versión en vivo de Angular de nuestra aplicación. Cuando ejecutamos el comando `ng build` Angular CLI, compila nuestra aplicación y genera archivos de paquete en esta carpeta.
 - static: contiene una versión renderizada previamente de nuestra aplicación Angular generada a partir de Scully cuando ejecutamos el comando `npm run scully`.

- 
- Si navegamos a la carpeta estática, veremos que Scully ha creado una carpeta para cada ruta de nuestra aplicación Angular. Cada carpeta contiene un archivo index.html, que representa el componente que se activa desde esa ruta. El contenido del índice html se genera automáticamente desde Scully, y es el resultado como si ejecutamos nuestra aplicación en vivo y navegamos hasta ese componente.
 - Ahora puedes tomar tu aplicación Angular, subirla al CDN o al servidor web de tu elección, ¡y tendrás tu blog listo en poco tiempo!

RESUMEN

- En este capítulo, aprendimos cómo combinar Angular con la biblioteca Scully para crear un blog personal.
- Vimos cómo Angular usa el paquete de enrutador incorporado para mejorar las aplicaciones web con navegación dentro de la aplicación. También aprendimos cómo organizar una aplicación Angular en módulos y cómo navegar a través de ellos.
- Introdujimos Jamstack en nuestra aplicación Angular usando la biblioteca Scully y vimos lo fácil que es convertir nuestra aplicación en un blog pre-renderizado. Usamos la interfaz de Scully para crear algunas publicaciones de blog y mostrarlas en la pantalla.
- En el siguiente capítulo, investigaremos otra característica interesante de Angular, las formas. Vamos a aprender a utilizarlas y a crear un sistema de seguimiento de problemas.

PREGUNTAS DE PRÁCTICA

¿HABRÁ EXAMEN?

- 1. ¿Qué biblioteca usamos para el enrutamiento en una aplicación Angular?
- 2. ¿Cómo agregamos capacidades de enrutamiento en un elemento de vínculo HTML?
- 3. ¿Qué pipe angular utilizamos para formatear la fecha?
- 4. ¿Cuál es el propósito de la carpeta de assets en una aplicación CLI de Angular?
- 5. ¿Qué propiedad de ruta utilizamos para cargar un módulo de forma retardada (lazy loading)?
- 6. ¿Qué comando CLI de Angular usamos para instalar Scully?
- 7. ¿Qué servicio utilizamos para buscar rutas de Scully?
- 8. ¿Qué es la propiedad vinculante?
- 9. ¿Qué directiva Angular usamos para iterar sobre una matriz en HTML?
- 10. ¿Cuál es la diferencia entre una aplicación Angular estándar y una Scully?

LECTURAS ADICIONALES

- Enrutamiento : <https://angular.io/guide/router>
- Módulos de funciones: <https://angular.io/guide/module-types>
- Módulos de carga diferida: <https://angular.io/guide/lazy-loadingngmodules>
- Pipes integradas : <https://angular.io/api?type=pipe>
- Bootstrap CSS: <https://getbootstrap.com/>
- Jamstack: <https://jamstack.org/>
- Scully: <https://scully.io/>
- Dominar Markdown: <https://guides.github.com/features/mastering-markdown/>

