# Routing con Angular 12 Router

En este documento, aprenderemos sobre el enrutador de Angular construyendo un ejemplo de Angular 12 y te enseñará todo lo que necesitas para empezar a usar el enrutamiento para construir aplicaciones de una sola página con navegación, guardias, resolvedores y animaciones.

Aprenderemos a utilizar múltiples salidas, redirigir a los usuarios desde la ruta vacía, utilizar rutas comodín para implementar páginas de error 404 y módulos de carga perezosa utilizando el método loadChildren().

## Navegación y enrutamiento usando el router de Angular 12 con un ejemplo

Ahora vamos a empezar con el enrutamiento de Angular. En esta sección, aprenderemos los conceptos básicos detrás del enrutamiento en Angular 12. Presentaremos el router de Angular y luego procederemos a crear una sencilla aplicación de una sola página con Angular 12 que demuestre las características más utilizadas del router.

## Presentación del router de Angular

El router de Angular es un elemento esencial de la plataforma Angular. Permite a los desarrolladores construir aplicaciones de una sola página con múltiples estados y vistas utilizando rutas y componentes y permite la navegación del lado del cliente y el enrutamiento entre los diversos componentes. Está construido y mantenido por el equipo central detrás del desarrollo de Angular y está contenido en el paquete @angular/router.

Puedes utilizar la URL del navegador para navegar entre los componentes de Angular de la misma manera que puedes utilizar la navegación habitual del lado del servidor.

Angular Router tiene una plétora de características como:

- El soporte para múltiples salidas de Router que te ayuda a añadir fácilmente un escenario de enrutamiento complejo como el enrutamiento anidado,
- Varias estrategias de coincidencia de rutas (prefijo y completa) para indicar al router cómo hacer coincidir una ruta específica con un componente,
- Fácil acceso a los parámetros de ruta y a los parámetros de consulta,
- Resolvers,
- Carga perezosa de módulos,
- Route guards para añadir protección en el lado del cliente y permitir o no el acceso a componentes o módulos, etc.

Angular 12 proporciona un potente router que permite mapear las rutas del navegador a los componentes. Así que vamos a ver cómo podemos añadir el enrutamiento a las aplicaciones construidas con Angular 12.

En esta sección, aprenderás varios conceptos relacionados con el enrutamiento de Angular como:

- Los componentes, las rutas y los caminos,
- La salida del enrutador,
- Las estrategias de coincidencia de rutas,
- Los parámetros de la ruta,
- Parámetros de consulta,
- Los guardianes de rutas,
- Los resolvedores de rutas,
- La directiva routerLink (sustituye al atributo href),
- Rutas auxiliares,
- Salidas de enrutadores primarios y secundarios.

Las aplicaciones de Angular se construyen como una jerarquía de componentes (o un árbol de componentes) que se comunican entre sí mediante entradas y salidas. Un componente controla una parte de la pantalla que se renderiza utilizando la plantilla del componente especificada como meta información en el decorador @Component.

- Un decorador @Component marca una clase como un componente de Angular y proporciona metadatos de configuración que determinan cómo el componente debe ser procesado, instanciado y utilizado en tiempo de ejecución.
- En las aplicaciones basadas en componentes, como las de Angular, una vista de pantalla se implementa utilizando uno o más componentes.
- El enrutamiento en Angular también se denomina enrutamiento de componentes porque el enrutador asigna un único componente o una jerarquía de componentes a una URL específica.

## Rutas y caminos

En Angular, una ruta es un objeto (instancia de Route) que proporciona información sobre qué componente se asigna a una ruta específica. Una ruta es el fragmento de una URL que determina dónde se encuentra exactamente el recurso (o página) al que se quiere acceder. Puedes obtener la ruta quitando el nombre del dominio de la URL.

- En Angular puedes definir una ruta utilizando configuraciones de rutas o instancias de la interfaz Route.
- Una colección de rutas define la configuración de rutas que es una instancia de Rutas.
- Cada ruta puede tener las siguientes propiedades:
  - path es una cadena que especifica el camino de la ruta.

  - pathMatch es una cadena que especifica la estrategia de coincidencia. Puede tomar prefijo (por defecto) o completo.

  - component es un tipo de componente que especifica el componente que debe asignarse a la ruta.

- redirectTo es el fragmento de URL al que será redirigido si la ruta coincide.

Estas son las propiedades más utilizadas de las rutas, pero hay muchas otras.

Por ejemplo, esta es la definición de una ruta que mapea la ruta /my/path/ al componente MyComponent:

{ path: 'my/path/', component: MyComponent }

La ruta puede ser una cadena vacía que normalmente se refiere a la URL principal de su aplicación o también puede ser una cadena comodín (**) que será igualada por el router si la URL visitada no coincide con ninguna ruta en la configuración del router. Esto se utiliza normalmente para mostrar un mensaje de página no existente o redirigir a los usuarios a una ruta existente.

## Estrategias de concordancia de rutas

El enrutador de Angular tiene un potente algoritmo de concordancia con varias estrategias de concordancia incorporadas y personalizadas.

Las estrategias de coincidencia incorporadas son prefijo (la predeterminada) y completa.

Cuando la estrategia de coincidencia de una ruta es el prefijo, el enrutador simplemente comprobará si el inicio de la URL del navegador tiene el prefijo de la ruta. Si ese es el caso, se renderizará el componente relacionado.

Este no es siempre el comportamiento deseado. En algunos escenarios, usted quiere que el enrutador coincida con la ruta completa antes de renderizar un componente. Puede establecer la estrategia completa utilizando la propiedad pathMatch de una ruta. Por ejemplo:

```
{ path: 'products', pathMatch: 'full', component: ProductListComponent}
```

Una estrategia completa asegura que el segmento de la ruta de la URL del navegador es igual a la ruta de la ruta.

Un caso especial de usar la propiedad completa es cuando se quiere hacer coincidir la ruta vacía. Porque usando la estrategia de prefijo coincidirá con todas las rutas ya que la ruta vacía prefija todas las rutas.

Por ejemplo, queremos redirigir al usuario a la ruta /products cuando visite nuestra aplicación. Así es como debería ser la configuración de nuestra ruta:

```
{ path: '',  redirectTo: '/products', pathMatch: 'full' }
```

También puede utilizar un emparejador personalizado si la combinación de la propiedad de ruta y la estrategia de emparejamiento no le ayuda a emparejar su componente con una URL específica.

Puede proporcionar un matcher personalizado utilizando la propiedad matcher de una definición de ruta. Para ver un ejemplo, consulte UrlMatcher.

## Route Parameters

Las rutas dinámicas se utilizan a menudo en las aplicaciones web para pasar datos (parámetros) o estado a la aplicación o entre varios componentes y páginas. El router de Angular tiene soporte para rutas dinámicas y proporciona una API fácil de usar para acceder a los parámetros de la ruta.

Puedes definir un parámetro de ruta utilizando la sintaxis de dos puntos seguida del nombre del parámetro. Por ejemplo:

```
{path: 'product/:id' , component: ProductDetailComponent}
```

- En el ejemplo, id es el parámetro de la ruta. /producto/1, /producto/2, /producto/p1 ... son ejemplos de URLs que serán comparadas a través de esta definición de ruta.
- El último segmento de estas URLs son los valores del parámetro id que se pasarán a ProductDetailComponent.
- En sus componentes emparejados, puede acceder a los parámetros de la ruta utilizando varias API:
- - Utilizando el servicio ActivatedRoute,
- - Usando el ParamMap Observable disponible a partir de Angular 4.

## Angular Route Guards

Los guardias de ruta le permiten permitir o desautorizar el acceso a sus rutas de aplicación específicas en función de algunos criterios (por ejemplo, si el usuario está conectado o no).

También puedes utilizar una guardia para evitar que los usuarios salgan de un componente en función de algunas condiciones (por ejemplo, si un formulario no se ha enviado todavía y se pueden perder datos).

Puedes utilizar los guards de Angular para proteger componentes o módulos completos.

Para proteger una ruta, primero tienes que crear una guardia subclasificando la interfaz CanActivate y sobrescribiendo el método canActivate() que debe devolver un valor booleano (true significa que el acceso está permitido) y añadirlo a la definición de la ruta mediante el atributo canActivate. Por ejemplo:

```
{path:    'product/:id,  canActivate:[ExampleGuard],  component:
ProductDetailComponent}
```

Este es un ejemplo de implementacion de ExampleGuard:

```
class MyGuard implements CanActivate {

  canActivate() {

    return true;

  }

}
```

Dado que el método canActivate() siempre devolverá true, esta protección siempre permitirá el acceso a ProductDetailComponent.

## Router Outlet

El Router-Outlet es una directiva exportada por el RouterModule y actúa como un marcador de posición que indica al enrutador dónde tiene que insertar el componente(s) correspondiente(s). El componente que tiene la salida del enrutador se denomina shell de la aplicación:

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

El enrutador angular admite más de una salida en la misma aplicación. La salida principal (o de nivel superior) se denomina salida principal. Otras salidas se llaman salidas secundarias.

Puede especificar una salida de destino para una definición de ruta utilizando el atributo de salida.

## Las Directivas de Navegación

Angular Router proporciona dos directivas para la navegación: la directiva routerLink que reemplaza el atributo href en las etiquetas <a> para crear enlaces y routerLinkActive para marcar el enlace activo. Por ejemplo:

```
<a [routerLink]="'/products'">Products</a>
```

***Step 1: Creando un proyecto Angular 12***

Para mostrarle cómo usar el enrutamiento Angular para crear una aplicación front-end con múltiples vistas de pantalla, crearemos un proyecto Angular 12 desde cero usando Angular CLI 12.

Nota: asegúrese de tener Angular CLI 12 instalado en su máquina de desarrollo para generar proyectos de Angular 12.

*Angular CLI requiere que tenga Node 10+ con NPM instalado en su máquina, por lo que sin estas dependencias no podrá instalar CLI en su máquina. Puede dirigirse fácilmente al sitio web oficial y descargar los archivos binarios correctos para su sistema operativo o seguir la documentación adecuada sobre cómo instalar una versión reciente de Node.js en su sistema*.

En el momento de realizar este docomento, Angular CLI 13.1.1 está instalado (npm install -g @angular/cli) usando npm 16.13.1

Abra una nueva terminal en su sistema, navegue hasta donde desee crear su proyecto y ejecute este comando:

```
$ ng new angular-routing-demo
```

Antes de proceder a generar el proyecto, la CLI le preguntará si:

***¿Le gustaría agregar enrutamiento angular?*** La respuesta predeterminada es No, así que escriba y para decirle a la CLI que instale el paquete @angular/router en el proyecto y genere un archivo src/app/app-routing.module.ts y también agregará un <router-outlet> en el src/app/app.component.html que será el shell de nuestra aplicación Angular. En versiones anteriores de Angular CLI, necesitaría crear este archivo manualmente.

***¿Qué formato de hoja de estilo le gustaría usar?*** (Utilice las teclas de flecha) CSS, Sass, Less o Stylus. Elija CSS y presione Entrar.

CLI generará la estructura del directorio y los archivos necesarios y también instalará las dependencias del proyecto desde npm y luego le devolverá el control.

*Nota: También puede pasar una opción de enrutamiento al comando ng new angular-routing-demo para indicarle que agregue enrutamiento en su proyecto sin preguntarle. Esta opción también es útil si está creando aplicaciones con ng new app o módulos con ng new module y desea configurar automáticamente el enrutamiento e incluir un archivo de módulo de enrutamiento.*

***Step 2: Comprender lo que CLI hizo automáticamente por usted***

Angular CLI ha configurado el enrutamiento en su proyecto y todo lo que tiene que agregar es definir asignaciones de componentes de ruta después de crear los componentes de su aplicación, pero ayuda a comprender los pasos que ha realizado la CLI para configurar el enrutamiento.

Si desea agregar enrutamiento manualmente en su aplicación o módulo, estos son los pasos necesarios que debe seguir:

***Step 2.1: Adicionando <base href>***

Primero, necesitaría abrir el archivo src/index.html y agregar una etiqueta <base> como elemento secundario de la etiqueta <head> **** que le permite al enrutador descubrir cómo componer rutas de navegación. Así es como se ve el index.html:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular Routing Demo</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

La etiqueta <base href> no es específica de Angular Router. Es una etiqueta HTML que especifica la URL base para todas las URL relativas en la página.

***Step 2.2: Creando un Routing Module***

A continuación, deberá crear un módulo de enrutamiento dentro del módulo de la aplicación principal y en su propio archivo mediante un comando como este:

```
$ ng generate module app-routing --module app --flat
```

La opción --flat le dice a la CLI que genere un archivo plano sin una subcarpeta. De esta forma, el archivo app-routing.module.ts se creará en la carpeta src/app junto con el archivo app.module.ts. Este es el contenido de este módulo antes de configurar el enrutamiento:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
@NgModule({
  declarations: [],
  imports: [
    CommonModule
```

```
  ]
})
export class AppRoutingModule { }
```

Este es un módulo regular decorado por el decorador NgModule e importa CommonModule.

*Nota: CommonModule es un módulo Angular integrado que exporta todas las directivas y canalizaciones básicas de Angular, como NgIf, NgForOf, DecimalPipe, etc.*

***Step 2.2: Importando el Router y configurando Routing***

A continuación, deberá abrir el archivo src/app/app-routing.module.ts y actualizarlo de la siguiente manera:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Importamos los símbolos Routes y RouterModule del paquete @angular/router, que es el repositorio central que contiene todas las API del enrutador.

A continuación, declaramos una variable de rutas del tipo Rutas.

A continuación, importamos RouterModule a través de la matriz de importaciones de nuestro módulo de enrutamiento y pasamos la matriz de rutas a RouterModule a través del método forRoot().

Y finalmente, exportamos RouterModule desde nuestro módulo de enrutamiento al agregarlo a la matriz de exportaciones.

Como se puede ver, AppRoutingModule es simplemente un wrapper alrededor de una instancia de RouterModule (devuelto desde el método estático forRoot()) que lo alimenta con el objeto de configuración de rutas. Ahora está vacío, pero le agregará rutas una vez que cree los componentes de su aplicación.

*Nota: RouterModule es un módulo de enrutamiento integrado que exporta el servicio del enrutador y las directivas necesarias para el enrutamiento, como RouterLink, RouterLinkActive, RouterLinkWithHref y RouterOutlet.*

El método estático forRoot() crea un módulo que contiene todas las directivas, las rutas dadas y el propio servicio de enrutador.

En algunas situaciones (para submódulos y submódulos con carga diferida), necesitará usar el método estático forChild() en su lugar, que crea un módulo que contiene todas las directivas y las rutas dadas, pero no incluye el servicio de enrutador.

Para obtener más detalles sobre la diferencia entre los dos métodos, consulte RouterModule.forRoot(ROUTES) vs RouterModule.forChild(ROUTES)

***Step 2.3: Adicionando Router-Outlet***

Después de configurar el módulo de enrutamiento, a continuación, deberá agregar la salida del enrutador en el componente principal de su aplicación. Abra el src/app/app.component.html, así es como se ve:

```
<div style="text-align:center">

  <h1>

    Welcome to {{ title }}!

  </h1>

  <!-- [...] -->

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

Lo importante en lo que debe concentrarse es <router-outlet>.

RouterOutlet es una directiva Angular incorporada que se exporta desde el paquete @angular/router, precisamente RouterModule y es un marcador de posición que marca en qué parte de la plantilla, el enrutador puede representar los componentes que coinciden con la URL actual y la configuración de rutas pasada al Enrutador.

*Nota: El componente que contiene la salida del enrutador actúa como un caparazón de su aplicación.*

***Step 2.4: Importando Routing Module en el Main Application Module***

Finalmente, deberá importar AppRoutingModule en su módulo de aplicación principal que reside en el archivo src/app/app.module.ts. Si abres ese archivo, así es como se ve:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
```

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

Importamos AppRoutingModule desde ./app-routing.module y lo agregamos a la matriz de importaciones de AppModule.

Hemos visto todos los pasos que necesitaría hacerse si Angular CLI no configura automáticamente el enrutamiento cuando se generó el proyecto.

***Step 3: Configurando un Service para obtener datos***

Esto no es parte de cómo funciona el enrutamiento en Angular, pero para el propósito de nuestra aplicación de demostración, necesitaremos crear un servicio que pueda usarse para obtener algunos datos para mostrar en los componentes de nuestra aplicación. Como no tenemos un proyecto de back-end que nos proporcione datos, podemos usar una característica muy útil de Angular: la API web en memoria disponible en el paquete angular-in-memory-web-api.

Este módulo simula una aplicación web back-end al interceptar las solicitudes de HttpClient y las redirige a un almacén de memoria que necesita para crear y proporcionar algunos datos en él.

Más tarde, cuando tenga un backend real, simplemente puede eliminar el módulo API web en memoria y todas sus solicitudes irán al backend real.

Primero, comencemos instalando el paquete desde npm usando el siguiente comando en su terminal:

```
$ npm install --save angular-in-memory-web-api
```

A continuación, creemos el servicio que devolverá los datos simulados. En su terminal, ejecute el siguiente comando:

```
ng generate service data
```

Abra el archivo src/app/data.service.ts e importe InMemoryDbService desde el paquete angular-in-memory-web-api:

```
import { InMemoryDbService } from 'angular-in-memory-web-api';
```

DataService debe implementar InMemoryDbService y anular el método createDb():

```
import { Injectable } from '@angular/core';
import { InMemoryDbService } from 'angular-in-memory-web-api';


@Injectable({
  providedIn: 'root'
})
export class DataService implements InMemoryDbService{
  constructor() { }
  createDb(){

    let  products =  [
      {  id:  1,  name:  'Product 1' },
      {  id:  2,  name:  'Product 2' },
      {  id:  3,  name:  'Product 3' },
      {  id:  4,  name:  'Product 4' },
      {  id:  5,  name:  'Product 5' }
    ];

    return { products };


   }
}
```

El único requisito es que cada objeto en la matriz de datos debe tener una identificación única id

A continuación, debe importar InMemoryWebApiModule y DataService en el archivo src/app/app.module.ts y agregarlos en la matriz de importaciones.:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
```

```
import { HttpClientModule } from "@angular/common/http";
import { InMemoryWebApiModule } from "angular-in-memory-web-api";
import { DataService } from "./data.service";

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    InMemoryWebApiModule.forRoot(DataService),
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

También importamos HttpClientModule porque necesitaremos HttpClient para enviar solicitudes de API.

A continuación, cree otro servicio para trabajar con productos. En tu terminal, ejecuta:

```
$ ng generate service product
```

Next, open the src/app/product.service.ts file and update accordingly:

```
import { Injectable } from '@angular/core';
import { HttpClient } from "@angular/common/http";

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  API_URL: string = "api/";

  constructor(private httpClient: HttpClient) { }
```

```
  getProducts() {
    return this.httpClient.get(this.API_URL + 'products')
  }

  getProduct(productId) {
    return this.httpClient.get(`${this.API_URL + 'products'}/${productId}`)
  }
}
```

Agregamos la cadena API_URL que contiene la dirección del servidor API.

Luego, importamos e inyectamos HttpClient y finalmente definimos los dos métodos getProducts() y getProduct(productId).

***Step 4: Creating a Model***

A continuación, creemos una clase de Producto que actuará como un modelo de datos para el tipo de producto. En tu terminal, ejecuta:

```
$ ng generate class product
```

Abra el archivo src/app/product.ts y agregue el siguiente código:

```
export class Product {
  id: number;
  name: string;
  constructor(id: number, name: string) {
    this.id = id;
    this.name = name;
  }
}
```

***Step 5: Creando Componentes***

Ahora que tiene un proyecto con configuración de enrutamiento y servicios de datos creados, necesita crear los componentes de su aplicación. Puede generar fácilmente componentes utilizando la CLI de Angular.

Regrese a su terminal y ejecute los siguientes comandos:

```
$ ng generate component product-list

$ ng generate component product-detail
```

Creamos dos componentes. El componente de lista de productos que muestra una lista de productos. Cuando haga clic en un producto específico, accederá al componente de detalles del producto que muestra ese único producto.

***Step 6: Implementando el Componente Product List***

Ahora agreguemos una implementación para ProductListComponent.

Abra el archivo src/app/product-list/product-list.component.ts y actualícelo según corresponda.

```
import { Component, OnInit } from '@angular/core';
import { ProductService } from "../product.service";
import { Product } from "../product";
@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {
  products: Product[] = [];

  constructor(private productService: ProductService) { }
  ngOnInit() {
    this.productService.getProducts().subscribe((products: Product[])=>{
      this.products = products;
      console.log(products);
    })
  }
}
```

Importamos ProductService y Product desde sus respectivas rutas.

A continuación, definimos una variable de productos de tipo Product[] y la inicializamos con una array vacía.

A continuación, inyectamos ProductService como productService a través del constructor del componente.

Finalmente, en el evento del ciclo de vida ngOnInit() del componente, nos suscribimos al Observable devuelto al llamar al método getProducts() y asignamos los productos obtenidos a la matriz de productos.

Ahora mostremos la lista de productos en el archivo src/app/product-list/product-list.component.html usando el siguiente código:

```
<div>
  <h1>
    Products
  </h1>
  <ul>
    <li *ngFor="let product of products">
      {{ product.name }}
    </li>
  </ul>
</div>
```

Simplemente usamos una directiva Angular ngFor para iterar sobre el array de productos y mostrar el nombre de cada producto.

***Step 7: Implementando el Componente Product Details***

Implementemos también el componente de detalles del producto.

Abra el archivo src/app/product-detail/product-detail.component.ts y agregue el siguiente código:

```
import { Component, OnInit } from '@angular/core';
import { ProductService } from "../product.service";
import { Product } from "../product";


@Component({
  selector: 'app-product-detail',
  templateUrl: './product-detail.component.html',
  styleUrls: ['./product-detail.component.css']
})
export class ProductDetailComponent implements OnInit {
```

```
  product: Product = new Product(-1,'No Product');
  constructor(private productService: ProductService) { }
  ngOnInit() {
    this.productService.getProduct(1).subscribe((product:
Product) =>{
      this.product = product;
    })
  }
}
```

Importamos ProductService y Product desde sus rutas.

A continuación, definimos una variable de producto de tipo Product que contendrá un producto y la inicializaremos con un producto inexistente.

A continuación, inyectamos ProductService como productService.

Finalmente, en el evento ngOnInit(), llamamos al método getProduct() de ProductService para recuperar un producto con id 1 y asignarlo a la variable de producto.

Estamos creando una variable de producto que contendrá el producto para mostrarlo en la plantilla. Por ahora, contiene el producto con la codificación identificaciónda 1, pero luego veremos cómo podemos usar el parámetro de ruta como fuente para obtener el producto apropiado por identificación para almacenar en esta variable.

Ahora abra src/app/product-detail/product-detail.component.html y agregue el siguiente código:

```
<div>
  <h1>
    Product #{{product.id}}
  </h1>
  <p>{{ product.name }}</p>
</div>
```

***Step 8: Definiendo las Routes***

Después de crear e implementar los componentes de nuestra aplicación, ahora debe agregarlos al enrutador.

Abrir src/app/app-routing.module.ts e importar sus componentes:

```
import { ProductListComponent } from "./product-list/product-
list.component";
```

```
import { ProductDetailComponent } from "./product-detail/product-detail.component";
```

A continuación, agregue el siguiente objeto a la matriz de rutas:

```
  {path: 'products' , component: ProductListComponent},
```

Esta primera ruta vincula la ruta /products a ProductListComponent, de modo que cuando se visita la URL /products, el enrutador representará el componente de la lista de productos.

A continuación, agregue la segunda ruta:

```
 {path: 'product/:id' , component: ProductDetailComponent}
```

Esto vinculará rutas con ID dinámicos como /producto/1 o /producto/9, etc. a ProductDetailComponent.

Nota: tenga en cuenta que, en este punto, ****ProductDetailComponent está vinculado a la ruta dinámica del producto/:id, pero no tiene la lógica para obtener el valor de la identificación de la ruta. Esto se tratará en el siguiente tutorial.

También podemos agregar esta ruta que redirigirá la ruta vacía a /products, de modo que cada vez que el usuario visite la ruta vacía, será redirigido al componente de productos:

```
{ path: '',  redirectTo: '/products', pathMatch: 'full' },
```

pathMatch se utiliza para especificar la estrategia de coincidencia completa o prefijo. full significa que la ruta completa de la URL debe coincidir con el algoritmo de coincidencia. prefijo significa que se elegirá la primera ruta en la que la ruta coincida con el inicio de la URL. En el caso de rutas vacías, si no establecemos la estrategia de coincidencia completa, no obtendremos el comportamiento deseado ya que cualquier ruta comienza con una cadena vacía.

***Step 9: Agregando Navigation Links***

Lo último que debe hacer es agregar los enlaces de navegación que lo llevan de un componente a otro.

Angular proporciona las directivas routerLink y routerLinkActive que deben agregarse a los anclajes <a>.

La directiva routerLink debe usarse en lugar del atributo href.

La directiva routerLinkActive se usa para agregar una clase CSS a un elemento cuando la ruta del enlace se vuelve activa.

Abra el archivo src/app/product-detail/product-detail.component.html y agregue un enlace para navegar a la lista de productos:

```
<a [routerLink] = "'/products'">

  Go to Products List

</a>
```

A continuación, abra el archivo src/app/product-list/product-list.component.html y agregue un enlace a cada producto para llevarlo al componente de detalles del producto:

```
<div>
  <h1>
    Products
  </h1>
  <ul>
    <li *ngFor="let product of products">
      <a routerLink= "/product/{{product.id}}">
      {{ product.name }}
      </a>
    </li>
  </ul>
</div>
```

Se ha visto los conceptos básicos del enrutamiento del lado del cliente, luego se observo los conceptos básicos del enrutador y finalmente fue creada una aplicación de una sola página con una construcción básica como componentes, servicios y módulos.

Hemos visto cómo puede configurar automáticamente el enrutamiento en su aplicación Angular 12 usando la CLI v12 y cómo puede simular un servidor backend usando la API web en memoria sin tener realmente un servidor backend.

En la siguiente sección, veremos cómo podemos acceder a los parámetros de ruta en estos componentes.

## Route Parameters con Snapshot y ParamMap

A continuacion, vamos a ver cómo manejar los parámetros de ruta con el ejemplo de Angular 12 Router usando diferentes métodos: Snapshot y ParamMap Observable.

Angular proporciona una potente biblioteca de enrutadores que permite a los desarrolladores implementar funciones avanzadas en sus aplicaciones Angular, además del enrutamiento básico de componentes, como:

- Route protection using guards,
- Route parameters,
- Child routes,
- Auxiliary routes etc.

## Manejando Route Parameters con Angular 12

En la parte anterior, hemos creado un enrutamiento básico entre componentes con Angular Router. En esta parte, se mostrara cómo manejar los parámetros de ruta en Angular 12.

Vamos a comenzar con la aplicación Angular simple que creamos en el tutorial anterior que puede encontrar en este repositorio o en CodeSandBox

Esta es la implementación del ProductDetailComponent:

```
import { Component, OnInit } from "@angular/core";
import { Product } from "../models/product";
@Component({
  selector: "product-detail",
  templateUrl: "./product-detail.component.html",
  styleUrls: []
})
export class ProductDetailComponent implements OnInit {


  public products: Product[] = [
    new Product(1, "Product 001"),
    new Product(2, "Product 002"),
    new Product(3, "Product 003"),
    new Product(4, "Product 004"),
    new Product(5, "Product 005"),
    new Product(6, "Product 006"),
    new Product(7, "Product 007"),
    new Product(8, "Product 008")
  ];
```

```
  product: Product = this.products[0];// this will store the
current product to display


  constructor() {}
  ngOnInit() {
  }
}
```

## Ejemplo de un Angular Route con un Parameter

En el apartado anterior, agregamos este objeto de ruta en la configuración de nuestro enrutador:

```
  { path: "product/:id", component: ProductDetailComponent }
```

El marcador de posición :id (llamado parámetro de enrutador dinámico) significa que el componente ProductDetailComponent se activará cuando el usuario visite cualquier ruta que coincida con esta expresión: /product/[0-9|a-b|A-B]+.

El enrutador angular también le permitirá recuperar el valor de: id del componente activado (es decir, en este caso, ProductDetailComponent), considerando la siguiente manera, a continuacion.

## Como obtener Route Parameters

Angular Router provee 2 formas diferentes para obtener route parameters:

- Usando route snapshot,
- Usando Router Observables

## Navigation Using The RouterLink Directive with Parameters

Abra src/app/product-list/product-list.component.html y luego cambie la lista de productos para usar etiquetas de anclaje con la directiva routerLink para poder navegar por el componente ProductDetailComponent.

```
<h1>Products List</h1>
<ul>
  <li *ngFor="let product of products">
    <a [routerLink]="['/product',product.id]"></a>
  </li>
</ul>
```

También puede crear enlaces utilizando:

```
<a routerLink="/product/"></a>
```

Después de crear los enlaces con parámetros. Ahora podemos proceder a ver cómo podemos recuperar el parámetro id de la URL en ProductDetailComponent.

Angular Router proporciona la clase ActivatedRoute que se puede inyectar en el componente. Entonces, primero comience importándolo usando:

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

A continuación, debe inyectar esta clase en el componente a través del constructor:

```
  constructor(private route: ActivatedRoute) {}
```

En el método de ciclo de vida ngOnInit del componente, agregaremos el código necesario para capturar el parámetro de ruta suscribiéndonos al mapa de parámetros o paramMap (instancia de ParamMap, disponible solo en Angular 4+) de la instancia inyectada:

```
  ngOnInit() {
    this.route.paramMap.subscribe(params => {
      this.products.forEach((p: Product) => {
        if (p.id == params.id) {
          this.product = p;
        }
      });
    });
  }
```

Después de suscribirnos a paramMap, tomamos los parámetros por sus nombres en el objeto de ruta (es decir, en nuestro caso id) params.id. El resto del código simplemente itera sobre la matriz de productos para encontrar el producto correspondiente para mostrar a través de su id.

También puede usar el objeto de instantánea de la instancia ActivatedRoute: this.route.snapshot.params.id

```
  ngOnInit() {
      this.products.forEach((p: Product) => {
        if (p.id == this.route.snapshot.params.id) {
          this.product = p;
        }
      });
```

```
}
```

El enrutador angular le permite recuperar fácilmente los parámetros de la URL, que es una funcionalidad esencial que requieren la mayoría de las aplicaciones web. Puede usar ambas formas: el paramMap observable o la forma de instantánea, pero la última requiere que tenga cuidado al reutilizar componentes.

## Angular 12 RouterLink, Navigate y NavigateByUrl

En las secciones anteriores, hemos visto cómo usar el enrutamiento básico entre componentes y cómo manejar los parámetros de ruta usando diferentes métodos. También hemos visto cómo usar la directiva RouterLink para crear enlaces de ruta. Esta sección es la continuación de la sección anterior con los otros métodos para implementar la navegación.

## Ejemplo de RouterLink con Angular 12

Echemos un segundo vistazo a cómo usamos la directiva RouterLink en los tutoriales anteriores.

Se crean enlaces básicos usando:

```
<a routerLink="/">Go To Home</a>
```

Or also:

```
<a [routerLink]="'/'">Go To Home</a>
```

Se crea a link con parámetro usando:

```
<a [routerLink]="['/product/',product.id]"></a>
```

## Navegando programáticamente usando Router.navigate() y Router.navigateByUrl() Angular 12

El enrutador Angular 12 proporciona dos métodos que puede usar para navegar a otros componentes en su clase de componente en lugar de usar la directiva RouterLink en la plantilla. Los dos métodos son navegar() y navegarByUrl() y pueden ser útiles en múltiples situaciones en las que necesita activar la navegación a través del código. Devuelven una promesa que se resuelve en verdadero o falso.

navegarByUrl() toma una cadena como parámetro. navegar() toma una matriz de segmentos de URL.

Así que modifiquemos nuestra aplicación Angular anterior para navegar usando uno de estos métodos. Adelante, abra src/app/product-list/product-list.component.ts y luego importe e inyecte la clase de enrutador:

```
import { Component } from "@angular/core";
import { Product } from "../models/product";
import { Router } from "@angular/router";
@Component({
  selector: "product-list",
  templateUrl: "product-list.component.html"
})
export class ProductListComponent {
  public products: Product[] = [
    new Product(1, "Product 001"),
    new Product(2, "Product 002"),
    new Product(3, "Product 003"),
    new Product(4, "Product 004"),
    new Product(5, "Product 005"),
    new Product(6, "Product 006"),
    new Product(7, "Product 007"),
    new Product(8, "Product 008")
  ];
  constructor(private router: Router){}
}
```

A continuación, agregue el método gotoProductDetails() que toma una URL y los parámetros de identificación que se pasan como una matriz de segmentos al método de navegación():

```
public gotoProductDetails(url, id) {

    this.router.navigate([url, id]).then( (e) => {

      if (e) {

        console.log("Navigation is successful!");

      } else {

        console.log("Navigation has failed!");

      }

    });
```

```
}
```

También puede crear una cadena a partir de estos parámetros y usar navegarByUrl():

```
  public gotoProductDetailsV2(url, id) {

    var myurl = `${url}/${id}`;

    this.router.navigateByUrl(myurl).then(e => {

      if (e) {

        console.log("Navigation is successful!");

      } else {

        console.log("Navigation has failed!");

      }

    });

  }
```

A continuación, abra la plantilla del componente en src/app/product-list/product-list.html, luego agregue un botón y vincule su acción de clic a uno de los métodos anteriores:

```
 <button   (click)="gotoProductDetails('/product',product.id)">Go To Details</button>
```

esta es toda la plantilla:

```
<h1>Products List</h1>

<ul>

    <li *ngFor="let product of products">

        <a  [routerLink]="['/product/',product.id]"></a>  <button (click)="gotoProductDetails('/product',product.id)">Go        To Details</button>

    </li>

</ul>

<a routerLink="/">Go To Home</a>
```

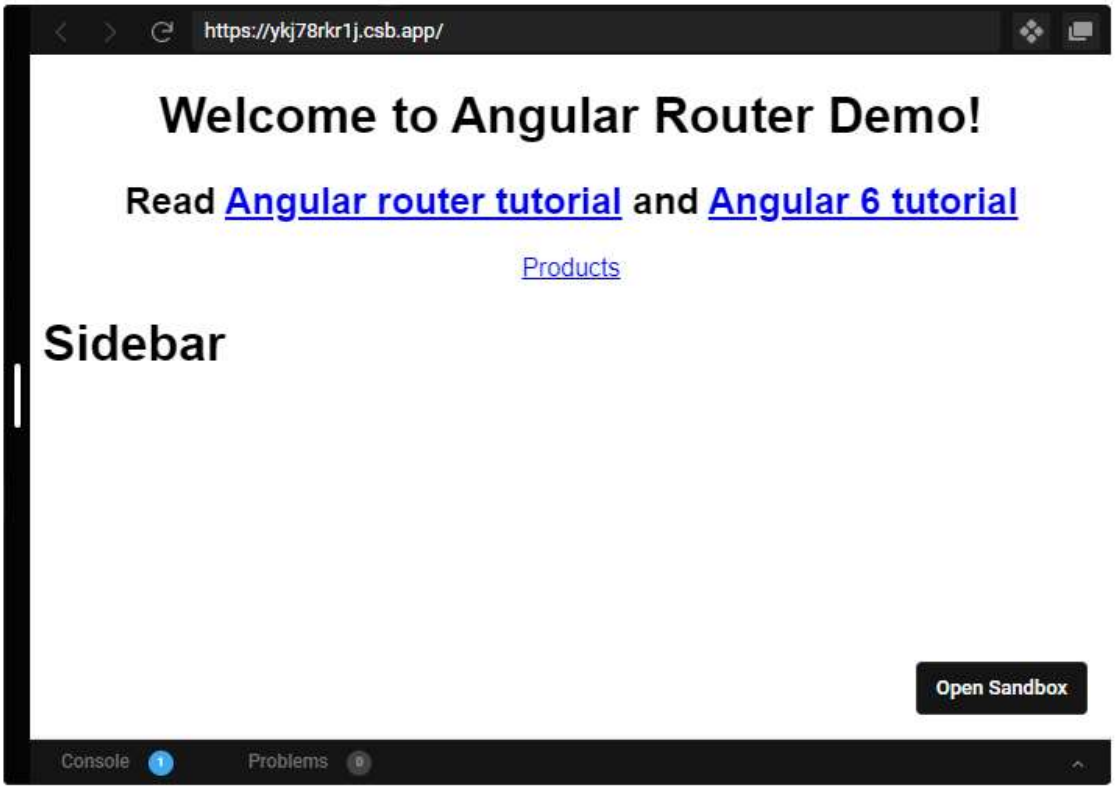
En este documento, hemos visto diferentes métodos para implementar la navegación con el enrutador Angular 12, es decir, usar la directiva routerLink con las etiquetas de anclaje en la plantilla HTML de los componentes o usar los métodos Router.navigate() y Router.navigateByUrl() en situaciones en las que desea para navegar en la clase de componente.

## Ejemplo de Outlets multilpes y con nombre(Auxiliary Routes)

En las secciones anteriores, hemos visto los conceptos básicos del enrutador Angular 12. Hemos visto cómo configurar el enrutador Angular y cómo agregar rutas.

Entonces, ahora comprende cómo agregar enrutamiento a su aplicación Angular 12 para crear un SPA (aplicación de una sola página) y también cómo vincular diferentes rutas usando RouterLink y RouterLinkActive. También comprende cómo usar la salida del enrutador (<router-outlet>). La salida del enrutador es un marcador de posición que Angular llena dinámicamente, según el estado actual del enrutador. Hemos utilizado la salida del enrutador para crear un enrutamiento básico. Ahora, veremos usos avanzados del componente <router-outlet>, como enrutamiento con nombre, múltiples outlets y enrutamiento auxiliar.

Esta es la aplicación que construiremos:




## Creando un Named Router Outlet con nombre

Puede crear una salida de enrutador con nombre usando la propiedad de nombre del componente <router-outlet>:

```
<router-outlet  name="outlet1"></router-outlet>
```

## Creando multiples Router-Outlets

Puede tener varios outlets en la misma plantilla:

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

<router-outlet  name="sidebar"></router-outlet>
```

La salida sin nombre es la salida principal.

Excepto el punto de venta principal, todos los demás outlets deben tener un nombre.

## Agregando multiples Outlets a nuestro Demo Application

Primero, puede obtener la fuente de nuestro ejemplo anterior de este repositorio.

Ahora, agreguemos varios outlets en nuestra plantilla AppComponent. Abra src/app/app.component.html y agregue los siguientes outlets:

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

<router-outlet  name="sidebar"></router-outlet>
```

## Conceptos sobre  Auxiliary Route

Un componente tiene una ruta principal y cero o más rutas auxiliares. Las rutas auxiliares le permiten utilizar y navegar por varias rutas. Para definir una ruta auxiliar, necesita una salida de enrutador con nombre donde se representará el componente de la ruta auxiliar.

El nombre que le estamos dando a la segunda salida sugiere que la salida se usará como una barra lateral para la aplicación. Ahora vamos a crear un componente de la barra lateral que se representará en la salida de la barra lateral:

```
ng g component sidebar
```

Queremos que el componente de la barra lateral se represente con cada uno de los demás componentes, al mismo tiempo. Así que agregaremos una ruta vacía y una salida en la barra lateral:

```
{
   path: "",
   component: SidebarComponent,
```

```
   outlet: "sidebar"
}
```

Dado que estamos utilizando una ruta vacía, el componente de la barra lateral se representará cuando se inicie nuestra aplicación.

## Navigando dentro de Auxiliary Outlets

Puede navegar dentro de una salida auxiliar usando la propiedad outlets:

```
router.navigate([{outlets: {primary: 'path' ,sidebar: 'path'}}]);
```

Or also using the routerLink directive

```
<a [routerLink]="[{ outlets: { primary: ['path'],sidebar: ['path'] } }]">

    Products List

</a>
```

## Router Outlets Primario y Auxiliary Angular 12 por Ejemplo

Entonces, digamos que queremos representar un componente de barra lateral diferente cuando el usuario navega a la URL /products. De esta forma, ProductListComponent se representará en la salida principal y, al mismo tiempo, ProductListSidebarComponent se representará en la auxiliary sidebar outlet.

Podemos lograr fácilmente este escenario creando el componente ProductListSidebarComponent (ng g componente ProductListSidebar) y agregando la siguiente configuración de ruta auxiliar:

```
  { path: "products", component: ProductListSidebarComponent, outlet: "sidebar" }
```

Esta es la configuración de enrutamiento completa para nuestro ejemplo:

```
import { RouterModule, Routes } from "@angular/router";
import { ModuleWithProviders } from "@angular/core";
import { ProductListComponent } from "./product-list/product-list.component";
import { ProductDetailComponent } from "./product-detail/product-detail.component";
import { SidebarComponent } from "./sidebar/sidebar.component";
import { ProductListSidebarComponent } from "./product-list-sidebar/product-list-sidebar.component";
```

```
const routes: Routes = [
  { path: "products", component: ProductListComponent },
  { path: "product/:id", component: ProductDetailComponent },
  {
    path: "",
    component: SidebarComponent,
    outlet: "sidebar"
  },
  {
    path: "products",
    component: ProductListSidebarComponent,
    outlet: "sidebar"
  }
];

export const routingModule: ModuleWithProviders = RouterModule.forRoot(routes);
```

También debe actualizar el enlace de navegación en nuestro AppComponent:

```
<a [routerLink]="[{ outlets: { primary: ['products'],sidebar: ['products'] } }]">

    Products List

</a>
```

Esto dice, cuando el usuario hace clic en el enlace Lista de productos. Ambas rutas con la ruta /products se activarán en los outlets de la barra lateral principal y auxiliar.

Debe especificar todos los outlets en los que desea que se realice la navegación, incluido el punto de venta principal.

Al tener outlets principales y auxiliares con nombre, puede implementar escenarios avanzados mediante la representación independiente de varios componentes al mismo tiempo.

En esta sección, hemos aprendido cómo usar Router-Outlets con nombre y múltiples y rutas auxiliares en Angular 12.

## Ejemplo de Resolve y Route Resolvers

El enrutador Angular 12 proporciona una propiedad de resolución que toma una resolución de ruta y permite que su aplicación obtenga datos antes de navegar a la ruta (es decir, resolver datos de ruta).

## Como crear un Angular 12 Route Resolver

Puede crear un route resolver en Angular 12 y versiones anteriores implementando la interfaz Resolver. Por ejemplo, este es un route resolver:

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

import { APIService } from './api.service';

import { Resolve } from '@angular/router';

@Injectable()

export class APIResolver implements Resolve<any> {

  constructor(private apiService: APIService) {}

  resolve() {

    return this.apiService.getItems();

  }

}
```

En el ejemplo, asumimos que ya hemos creado un APIService que tiene un método getItems() que obtiene datos de un extremo API remoto.

Importamos la interfaz Resolve del paquete @angular/router.

Luego creamos una clase APIResolver que implementa la interfaz Resolve<any>.

En el constructor del resolver inyectamos nuestro APIService como apiService y llamamos al método getItems() del servicio en el método resolve() que debe definirse en cualquier resolver

## Accessando a Route Parameters en Resolver

A menudo, al resolver los datos de la ruta, desea obtener acceso a los parámetros de la ruta en el resolver. Puede hacerlo usando la clase ActivatedRouteSnapshot. Por ejemplo, supongamos que nuestra ruta tiene un parámetro de fecha que debe pasarse al método getItems(date):

```
import { Injectable } from '@angular/core';
import { APIService } from './api.service';
```

```
import { Resolve } from '@angular/router';
import { ActivatedRouteSnapshot } from '@angular/router';

@Injectable()
export class APIResolver implements Resolve<any> {
  constructor(private apiService: APIService) {}

  resolve(route: ActivatedRouteSnapshot) {
    return this.apiService.getItems(route.params.date);
  }
}
```

Importamos la clase ActivatedRouteSnapshot del paquete @angular/router y proporcionamos una ruta de parámetros de tipo ActivatedRouteSnapshot al método resolve(). Finalmente usamos route.params.date para obtener el valor del parámetro de fecha.

## Pasando de Route Resolver al Angular 12 Router

Una última cosa que debe hacer es pasar el resolver que creamos para resolver la propiedad de la ruta correspondiente en la matriz de rutas de su módulo de enrutamiento angular:

```
{
  path: 'items/:date',
  component: ItemsComponent,
  resolve: { items: APIResolver }
}
```

## Route Animations por Ejemplo

El enrutador Angular 12 admite agregar animaciones al navegar entre diferentes rutas en su aplicación. En este tutorial, aprenderemos cómo usar la API de animaciones de Angular para reproducir animaciones cuando una ruta cambia en su aplicación.

## Creando un Angular 12 Project

En este apartado asumimos que ya tiene instalado Angular CLI 12. Luego puede generar un proyecto usando el siguiente comando desde su terminal:

```
$ ng new angular-project
```

We also need some components. Create them using:

```
$ ng g c list
```

```
$ ng g c detail
```

## Agregando Angular 12 Routes

A continuación, debe agregar rutas a los componentes Angular 12 creados en su módulo de enrutamiento app-routing.module.ts:

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


import { HomeComponent } from './home/home.component';
import { ListComponent } from './list/list.component';
import { DetailComponent } from './detail/detail.component';


export const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'list', component: ListComponent },
  { path: 'detail', component: DetailComponent },


];


@NgModule({
  imports: [
    RouterModule.forRoot(routes)
  ],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

En el archivo src/app/app.component.html podemos agregar el siguiente código para navegar entre los diferentes componentes:

```
<nav>
```

```
    <a routerLink="">home</a>
    <a routerLink="list">list</a>
    <a routerLink="detail">detail</a>
</nav>
<div>
  <router-outlet></router-outlet>
</div>
```

La directiva routerLink se utiliza para crear enlaces a rutas definidas en el módulo de enrutamiento.

El <router-outlet> es donde el enrutador Angular inserta los componentes que coinciden con la ruta actual.

## Agregando Angular Animations Module: BrowserAnimationsModule

Antes de que podamos crear animaciones de enrutamiento, debemos importar el módulo de animaciones en el módulo de la aplicación principal:

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

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

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

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

import { AppRoutingModule } from './app-routing.module';

import { HomeComponent } from './home/home.component';

import { ListComponent } from './list/list.component';

import { DetailComponent } from './detail/detail.component';

@NgModule({

  imports: [BrowserModule, BrowserAnimationsModule, AppRoutingModule],

  declarations: [AppComponent, HomeComponent, ListComponent,DetailComponent],

  bootstrap: [AppComponent]

})
```

```
export class AppModule { }
```

## Definiendo y Registrando Angular Router Animations

En app.component.ts, debe definir su animación y registrarse en el array de animaciones del componente:

```
import { Component } from '@angular/core';
import {
  transition,
  trigger,
  query,
  style,
  animate,
  group,
  animateChild
} from '@angular/animations';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  animations: [
trigger('myAnimation', [
  transition('* => *', [
    query(
      ':enter',
      [style({ opacity: 0 })],
      { optional: true }
    ),
    query(
      ':leave',
       [style({ opacity: 1 }), animate('0.3s', style({ opacity: 0
}))],
      { optional: true }
    ),
    query(
```

```
      ':enter',
      [style({ opacity: 0 }), animate('0.3s', style({ opacity: 1
}))],
      { optional: true }
    )
  ])
]);

  ] // register the animations

})
export class AppComponent { }
```

Primero importamos un montón de métodos del módulo @angular/animations. A continuación, en la matriz de animaciones del componente, definimos nuestra animación.

Usamos el método trigger() para crear un disparador myAnimation que se aplicará al  que contiene la salida del enrutador en la plantilla del componente.

A continuación, en el segundo parámetro (array) del método trigger() proporcionamos un método transition() que toma dos parámetros: el primer parámetro especifica cuándo se deben aplicar las animaciones. En este caso cuando hay una transición de cualquier ruta a cualquier ruta ('* <=> *'). El segundo parámetro toma un array de animaciones.

También podemos especificar cualquier número de transiciones ya que el segundo parámetro del método trigger() es de tipo Array.

A continuación, creamos un group([]) de métodos query() para consultar cualquier componente que esté entrando o saliendo del DOM y aplicar estilos y animaciones a los estados: enter y :leave, lo que crea efectos de entrada y salida graduales.

## Applying the Animation on the Router Outlet

After defining the myAnimation animation we need to apply it to our router outlet

```
<div [@myAnimation]="o.isActivated ? o.activatedRoute : ''">

  <router-outlet #o="outlet"></router-outlet>

</div>
```

Usamos una referencia de plantilla para crear una referencia al router outlet #o="outlet". Esto es útil para saber cuándo la salida del enrutador está activa para que podamos activar la animación.

En esta sección rápida, hemos visto cómo definir y activar animaciones al navegar entre rutas en aplicaciones de Angular 12.

## Ejemplo 2: Redirección de rutas y manejo de rutas 404 mediante rutas wildcats de enrutador

En este ejemplo, veremos cómo redirigir a los usuarios a una nueva ruta de URL o componente usando el enrutador Angular 12 y cómo lidiar con páginas no encontradas y redirigir a un componente 404 si no se encuentra ninguna coincidencia usando rutas comodín.

Usaremos la última versión de Angular 12.

***Por qué redirigir a nuevas rutas en Angular?***

La redirección es una técnica común en el desarrollo web donde una ruta de URL específica se redirige a una nueva por muchas razones, como migrar una aplicación heredada o si la página solicitada no está disponible, etc.

Suponemos que ya tiene una máquina de desarrollo con Angular CLI instalada y que ha inicializado un proyecto Angular 12.

Cuando cree su proyecto con Angular CLI, se le preguntará si desea agregar enrutamiento. Si respondió Sí, se creará automáticamente un módulo de enrutamiento y simplemente puede comenzar a agregar las rutas de su aplicación.

De lo contrario, deberá configurar el enrutamiento manualmente.

***Step 1 – Setting up routing in your Angular app***

Antes de ver cómo redirigir a los usuarios a nuevas rutas o componentes, primero debemos configurar el enrutamiento en nuestro proyecto Angular 12.

Abra una nueva interfaz de línea de comandos y navegue hasta la carpeta de su proyecto, luego ejecute el siguiente comando: bash $ ng generar módulo app-routing --flat --module=app

El uso de la bandera --flat asegurará que el archivo de enrutamiento se agregará dentro de la carpeta src/app sin una subcarpeta.

El uso de --module=app le dice a Angular CLI que registre el módulo de enrutamiento como parte del módulo de la aplicación, lo que significa agregarlo en la matriz de importaciones del archivo src/app/app.module.ts.

Abra el archivo src/app/app-routing.module.ts, debe contener el siguiente código:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';


@NgModule({
  declarations: [],
  imports: [
    CommonModule
  ]
})
export class AppRoutingModule { }
```

Este es un módulo Angular típico y necesitamos agregarle una configuración de enrutamiento.

***Step 2 - Agregando configuración de routing***

Ahora agreguemos la configuración del enrutador al módulo de enrutamiento. Comience importando Routes y RouterModule desde @angular/router de la siguiente manera:

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

A continuación, defina un array de rutas que contendrá nuestras rutas de la siguiente manera:

```
const routes: Routes = [];
```

A continuación, incluya la llamada al método forRoot() de RouterModule con la matriz de rutas como argumento de la siguiente manera:

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

A continuación, debe agregar la salida del enrutador a la plantilla de la aplicación. Abra el archivo src/app/app.component.html y actualícelo de la siguiente manera:

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

Eso es todo, hemos configurado el enrutador en nuestro proyecto Angular 12.

***Step 3 – agregando componentes al router***

A continuación, antes de ver un ejemplo de cómo redirigir a los usuarios a nuevas rutas o componentes, necesitamos uno o más componentes en nuestro proyecto. Dirígete a tu interfaz de línea de comandos y ejecuta los siguientes comandos:

```
$ ng generate component home
```

```
$ ng generate component about
```

A continuación, en el archivo src/app/app-routing.module.ts importe los componentes de la siguiente manera:

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

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

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';
```

luego, se agrega las rutas como sigue:

```
const routes: Routes = [

  { path: 'home', component: HomeComponent },

  { path: 'about', component: AboutComponent }

];
```

Entonces, si visita la ruta /home, debe ir al componente home y si visita la ruta /about, debe ir al componente about.

***Step 4 - Redirigir la ruta vacía a la ruta de inicio***

Ahora que hemos agregado el enrutamiento a la página de inicio y sobre los componentes, veamos cómo redirigir a los usuarios a la ruta /home cuando visitan nuestra aplicación por primera vez desde la ruta vacía.

Simplemente necesitamos agregar una nueva ruta que coincida con la ruta vacía y redirigirla a la ruta /home de la siguiente manera:

```
const routes: Routes = [
```

```
  { path: '', pathMatch: ‘full’, redirectTo: 'home' },
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent }
];
```

Internamente, el enrutador usa una función llamada applyRedirects para procesar los redireccionamientos.

***Step 5 – Manejo de  404 (not found pages) usando wildcard paths***

Ahora, veamos cómo manejar la página 404 (no encontrada) en Angular. Regrese a su terminal y ejecute el siguiente comando para generar un componente no encontrado:

```
$ ng generate component notfound
```

A continuación, abre el archivo src/app/app-routing.module.ts y añade estas dos rutas:

```
const routes: Routes = [
  // [...]
  {path: '404', component: NotFoundComponent},
  {path: '**', redirectTo: '/404'}
];
```

No olvides importar el componente en el módulo de enrutamiento de la siguiente manera:

```
import {NotFoundComponent} from './notfound/notfound.component';
```

Usamos la ruta comodín denotada por ** para atrapar cualquier ruta no existente y usamos la propiedad redirectTo para redirigirlos a la ruta /404 que mapea al componente no encontrado.

En esta sección, hemos visto por ejemplo cómo redirigir a los usuarios a diferentes rutas en tu aplicación Angular y cómo manejar 404 no encontrados o rutas inválidas usando wildcats para rutas.

## Ejemplo de Angular Router con Parametros y Guards

Angular proporciona un potente router para crear apps con múltiples vistas, parámetros, guardias y navegación, etc. En este tutorial, veremos cómo implementar el enrutamiento por ejemplo en Angular 12 pero esto también es válido para versiones anteriores como Angular 7/8.

Paso 1: Configurar el enrutamiento en Angular 12

Paso 2: Añadiendo múltiples vistas/páginas o configurando las rutas de Angular

Paso 3: Añadiendo navegación en Angular

Paso 4: Uso de parámetros de ruta en Angular

Paso 5: Usando Guardias de Ruta en Angular

En primer lugar, necesitas tener unos cuantos requisitos previos si pretendes seguir este proceso paso a paso:

Un entorno de desarrollo con Node.js y npm instalado,

**Step 1: Configurando Angular Routing**

A partir de Angular 7, la CLI permitirá configurar automáticamente el enrutamiento sin la molestia de crear y configurar un módulo de enrutamiento.

Así que todo lo que necesitas es iniciar un nuevo proyecto de Angular 12 ejecutando el siguiente comando en tu interfaz de línea de comandos:

```
$ ng new angular-routing-example
```

Se le preguntará si desea añadir enrutamiento a su proyecto - Debe responder con una "Y" para configurar automáticamente un módulo de enrutamiento en su proyecto.

Para el formato de las hojas de estilo, elegiremos CSS, pero puede elegir cualquier opción, ya que no afecta a cómo se realiza el enrutamiento.

A continuación, puede iniciar un servidor de desarrollo utilizando los siguientes comandos:

```
$ cd ./angular-routing-example
```

```
$ ng serve
```

You can access your app from the http://localhost:4200 address using your web browser.

**Step 2: Adding Multiple Views/Pages or Configuring Angular Routes**

Ahora que tienes el enrutamiento configurado, puedes añadir múltiples páginas o vistas con navegación.

Para crear una página, necesitas usar un componente Angular así que Lets create component y ver cómo usar el enrutamiento con él.

Puedes dejar abierta la interfaz de línea de comandos anterior para ejecutar el servidor de desarrollo y dirigirte a uno nuevo. A continuación, navega a tu proyecto y ejecuta el comando

```
$ cd ./angular-routing-example

$ ng generate component home

$ ng generate component about

$ ng generate component contact
```

Hemos nombrado los componentes para reflejar sus propósitos. Cada componente puede ser configurado como una nueva vista en su aplicación.

Puedes crear una vista mapeando un comonente a una ruta de URL usando el array de rutas de la configuración de tu router.

Abre el archivo src/app/app-routing.module.ts y empieza importando los componentes anteriores como sigue:

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

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

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';

Next, you need to add routes to the routes array:

const routes: Routes = [

    { path: 'home', component: HomeComponent },

    { path: 'about', component: AboutComponent },

    { path: 'contact', component: ContactComponent },

];

@NgModule({

    imports: [RouterModule.forRoot(routes)],

    exports: [RouterModule]
```

```
})

export class AppRoutingModule { }
```

Ahora tenemos tres vistas a las que se puede acceder desde las rutas /home, /about y /contact.

¿Cómo sabe el enrutador dónde debe renderizar estos componentes?

A través de la directiva router-outlet que es insertada automáticamente por Angular CLI al configurar el enrutamiento en el paso anterior en la plantilla src/app/app/component.html.

Esta es la plantilla asociada al componente raíz de nuestra aplicación que es renderizada por Angular cuando se inicia la app.

Después de añadir la salida del enrutador al componente de la app, ahora se puede referir a él como el shell de la app de esta aplicación.

Puedes añadir cualquier navegación o parte estática de tu UI en la app shell. Añadamos, por ejemplo, un menú de navegación.

***Step 3: Añadiendo Navegacion en Angular 12***

Abre el archivo src/app/app.component.html y añade la siguiente marca encima de la directiva <router-outlet>:

```
<ul>

    <li><a [routerLink]="['/home']">Home</a></li>

    <li><a [routerLink]="['/about']">About</a></li>

    <li><a [routerLink]="['/contact']">Contact</a></li>

</ul>

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

En HTML, utilizamos el atributo href de los elementos <a> para especificar la ruta de destino pero con Angular utilizamos la directiva routerLink para crear enlaces de navegación.

Esta directiva toma la ruta asociada al componente para navegar <a>.

**Step 4: Usando parámetros de Angular Route**

A menudo, necesitamos utilizar rutas con parámetros en nuestra aplicación.

En el router de Angular, esto es soportado usando la sintaxis de dos puntos.

Vuelve a tu interfaz de línea de comandos y crea un nuevo componente:

```
$ ng generate component posts
```

A continuación, vuelve al archivo src/app/app-routing.module.ts e importa y crea una ruta para el nuevo componente:

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

import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
import { PostsComponent } from './posts/posts.component';
const routes: Routes = [
    { path: 'home', component: HomeComponent },
    { path: 'about', component: AboutComponent },
    { path: 'contact', component: ContactComponent },
    { path: 'posts/:id', component: PostsComponent}

];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

Añadimos una ruta que acepta un parámetro de id utilizando la sintaxis :id.

Ahora, podemos ir a rutas como /posts/1, ..., o /posts/2abc y podemos realmente acceder al ID pasado en el PostsComponent.

Puedes recuperar el parámetro de la ruta URL usando el servicio ActivatedRoute que está disponible en el paquete @angular/router.

Abre el archivo src/app/posts/posts.component.ts y comienza importando ActivatedRoute y Router como sigue:

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

A continuación, hay que inyectar ActivatedRoute a través del constructor del componente de la siguiente manera::

```
constructor(private activatedRoute: ActivatedRoute) { }
```

A continuación, puede recuperar el parámetro ID de la siguiente manera::

```
ngOnInit() {

    console.log(this.activatedRoute.snapshot.params['id'])

}
```

***Step 5: Usando Angular Route Guards***

El router de Angular nos permite proteger/guardar las rutas de la navegación del usuario utilizando guardias de ruta.

Un route guard permite ejecutar algún código cuando se navega por una ruta, y en base a ello, concede o deniega el acceso a la ruta.

Puedes crear una route guard extendiendo la interfaz CanActivate exportada desde el paquete @angular/router y sobrescribiendo el método canActivate() que debe contener el código que concede o deniega el acceso a una determinada ruta una vez que le aplicamos la guardia.

Vuelve a tu interfaz de línea de comandos y ejecuta el siguiente comando para crear una guardia de ruta:

```
$ ng generate guard login
```

Esto creará la clase LoginGuard con el método canActivate donde necesitas añadir algún código que devuelva true o false que conceda o deniegue el acceso a una ruta.

A continuación, vuelve al módulo de enrutamiento en el archivo src/app/app-routing.module.ts y aplica el guard a alguna ruta que quieras proteger. Por ejemplo:

```
import { LoginGuard } from './login/login.guard';

const routes: Routes = [

    { path: 'home', canActivate:[LoginGuard], component: HomeComponent },
```

```
    { path: 'about', canActivate:[LoginGuard], component: AboutComponent },

    { path: 'contact', canActivate:[LoginGuard], component: ContactComponent },

    { path: 'posts/:id', canActivate:[LoginGuard], component: PostsComponent}

];
```

Importamos la clase guard y la aplicamos a todas las rutas utilizando la propiedad canActivate.

## Modulos Lazy Loading (Ejemplo loadChildren())

En esta sección, veremos mediante un ejemplo cómo cargar lentamente componentes utilizando módulos de características y el método loadChildren().

Los módulos de carga lenta en Angular permiten que las aplicaciones carguen los módulos sólo cuando son necesarios, es decir, cuando se visita por primera vez la(s) ruta(s) correspondiente(s) a los componentes que pertenecen al módulo cargado de forma lenta. Esto tiene muchos beneficios en su aplicación Angular como el rendimiento y el tamaño.

Para añadir la carga lenta (lazy load) en la aplicación Angular 12 necesitas configurar el enrutamiento para utilizar el método loadChildren() y añadir los componentes que quieras cargar de forma lenta dentro de los módulos de características, es decir, fuera del módulo principal de la aplicación app-main.module.ts.

## Creando un Feature Module

Ahora tenemos que crear un módulo de características utilizando el siguiente comando:

```
$ ng g module lazymodule
```

También necesitamos crear componentes dentro de nuestro módulo de características:

```
$ ng g c lazymodule/component1

$ ng g c lazymodule/component2

$ ng g c lazymodule/component3
```

Estos comandos generarán tres componentes dentro del módulo lazymodule.

## Using loadChildren()

En el archivo principal de enrutamiento app-routing.module.ts, necesitas usar el método loadChildren() para cargar el módulo de características de forma lazy load:

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


const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'lazymodule', loadChildren:
'./lazymodule/lazymodule.module#LazyModuleModule' }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

El método loadChildren() toma la ruta de acceso al módulo, anexada al # nombre de la clase del módulo.

**Routing de componentes dentro del módulo de características**

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



import { Component1Component } from
'./component1/component1.component';


import { Component2Component } from
'./component2/component2.component';


import { Component3Component } from
'./component3/component3.component';


const routes: Routes = [
    { path: '', component: Component1Component },
    { path: 'component2', component: Component2Component },
```

```
    { path: 'component3', component: Component3Component },
];
@NgModule({
  imports: [
    RouterModule.forChild(routes)
  ],
  declarations:
[Component1Component,Component2Component,Component3Component]
})
export class LazyModuleModule {}
```

En el módulo de características, incluimos las rutas con el método forChild() de RouterModule en lugar del método forRoot().

En esta sección, hemos visto cómo cargar módulos de forma perezosa con el router de Angular 12 utilizando módulos de características y el método loadChildren().