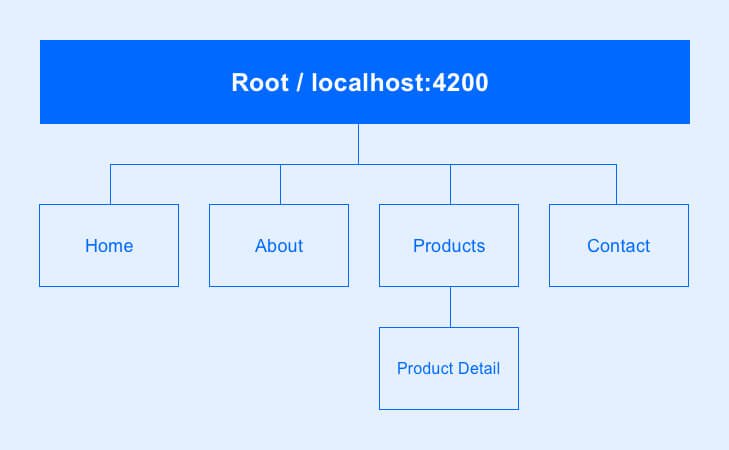
## What is Routing

Routing allows you to move from one part of the application to another part or one View to another View.

**Introducing Angular 13 Router**

8.4M

120

**Introduction to Angular Routing**

**Navigating from one page to another page in a web application is known as Routing.** Routing in Angular is called Angular Routing. Below is a familiar model of application navigation which a web browser follows:

* The web browser navigates to a corresponding page when you enter a URL in the address bar
* The browser navigates to a new page when you click a link on the page
* When you click the browser’s forward and backward buttons, it navigates to the corresponding pages through the history of pages you have visited

The Angular Router adopts from this model for navigation. A

ngular is a Single Page Application (SPA) that uses multiple Angular routes. In SPA, all the functionalities of your application exist in a single HTML page. The browser renders only the parts that are required for the user rather than loading a new page.

The Angular router is the fundamental block of the Angular platform. It enables developers to build Single Page Applications(SPAs) with multiple views and navigation between them.

**Introducing the Angular Router**

The Angular router is an essential element of the Angular platform. It allows developers to build **Single Page Applications** with multiple states and views using routes and components and allows client side navigation and routing between the various components.

It’s built and maintained by the core team behind Angular development and it’s contained in the **@angular/router package.**

You can use the browser's URL to navigate between Angular components in the same way you can use the usual server side navigation.

**Angular Router has a plethora of features such as:**

* **The support for multiple Router outlets which helps you easily add complex routing scenario like nested routing,**
* **Various path matching strategies ( prefix and full) to tell the Router how to match a specific path to a component,**
* **Easy access to route parameters and query parameters,**
* **Resolvers,**
* **Lazy loading of modules,**
* **Route guards for adding client side protection and allow or disallow access to**
* **components or modules, etc.**

## https://cdn-images-1.medium.com/max/1019/1*KsCNdY6CObkvwZq63FYGMg.png

## Content image

## The Angular Router Module

The Router is a separate module in Angular. It is in its own library package, **@angular/router.** The Router Module provides the necessary service providers and directives for navigating through application views.

Using Angular Router you can

* Navigate to a specific view by typing a URL in the address bar
* Pass optional parameters to the View
* Bind the clickable elements to the View and load the view when the user performs application tasks
* Handles back and forward buttons of the browser
* Allows you to dynamically load the view
* Protect the routes from unauthorized users using Guards

## Components of Angular Router Module

### Router

The Angular Router is an object that enables navigation from one component to the next component as users perform **application tasks like clicking on menus links, buttons or clicking on back/forward button on the browser**. We can access the router object and use its methods like navigate() or navigateByUrl(), to navigate to a route

### Route

Route tells the Angular Router which view to display when a user clicks a link or pastes a URL into the browser address bar. Every Route consists of a path and a component it is mapped to. The Router object parses and builds the final URL using the Route

### Routes

Routes is an array of Route objects our application supports

### RouterOutlet

The outerOutlet is a directive (<router-outlet>) that serves as a placeholder, where the Router should display the view

### RouterLink

The RouterLink is a directive that binds the HTML element to a Route. Clicking on the HTML element, which is bound to a RouterLink, will result in navigation to the Route. The RouterLink may contain parameters to be passed to the route’s component.

### RouterLinkActive

RouterLinkActive is a directive for adding or removing classes from an HTML element that is bound to a RouterLink. Using this directive, we can toggle CSS classes for active RouterLinks based on the current RouterState

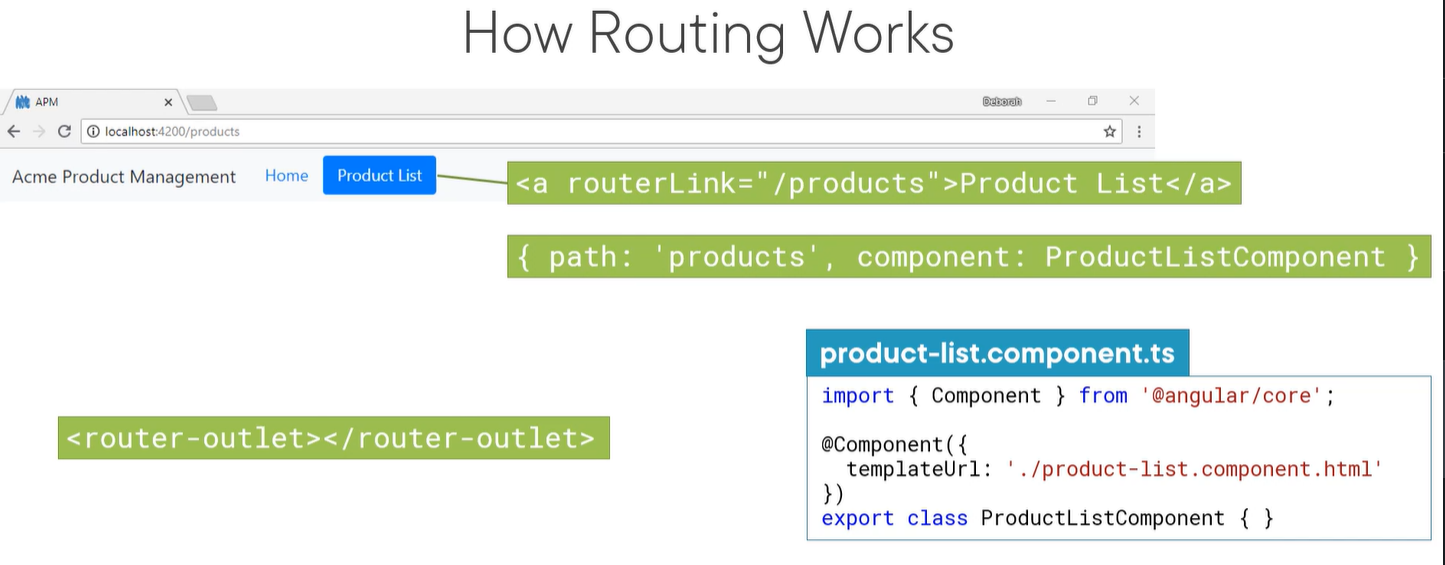
### ActivatedRoute

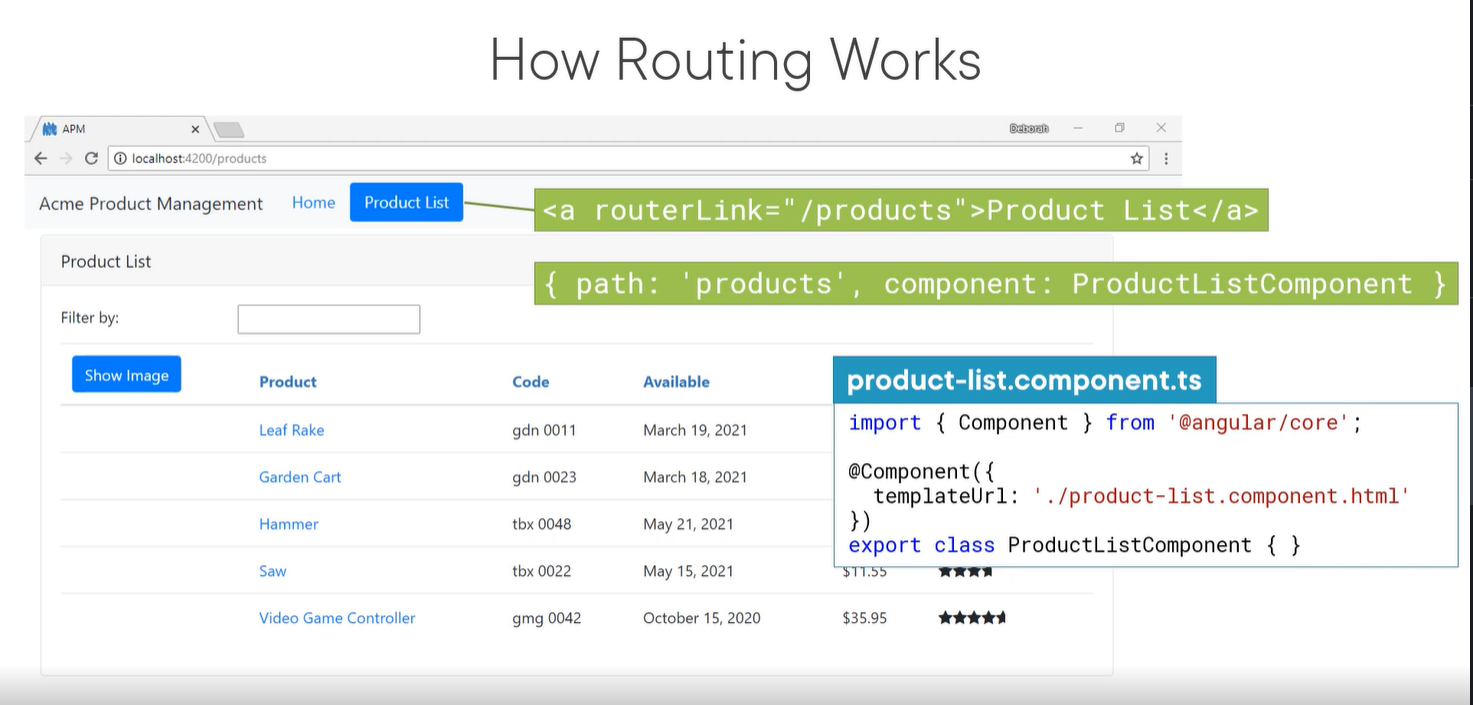
The ActivatedRoute is an object that represents the currently activated route associated with the loaded Component.

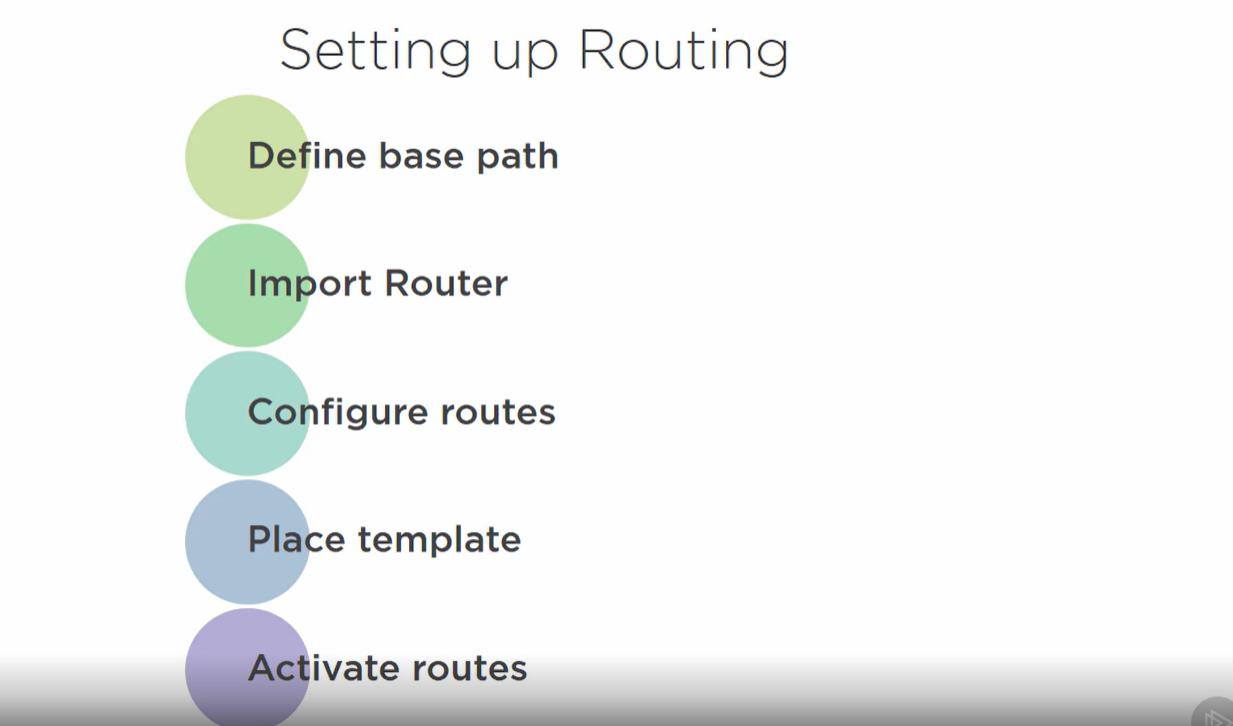
### RouterState

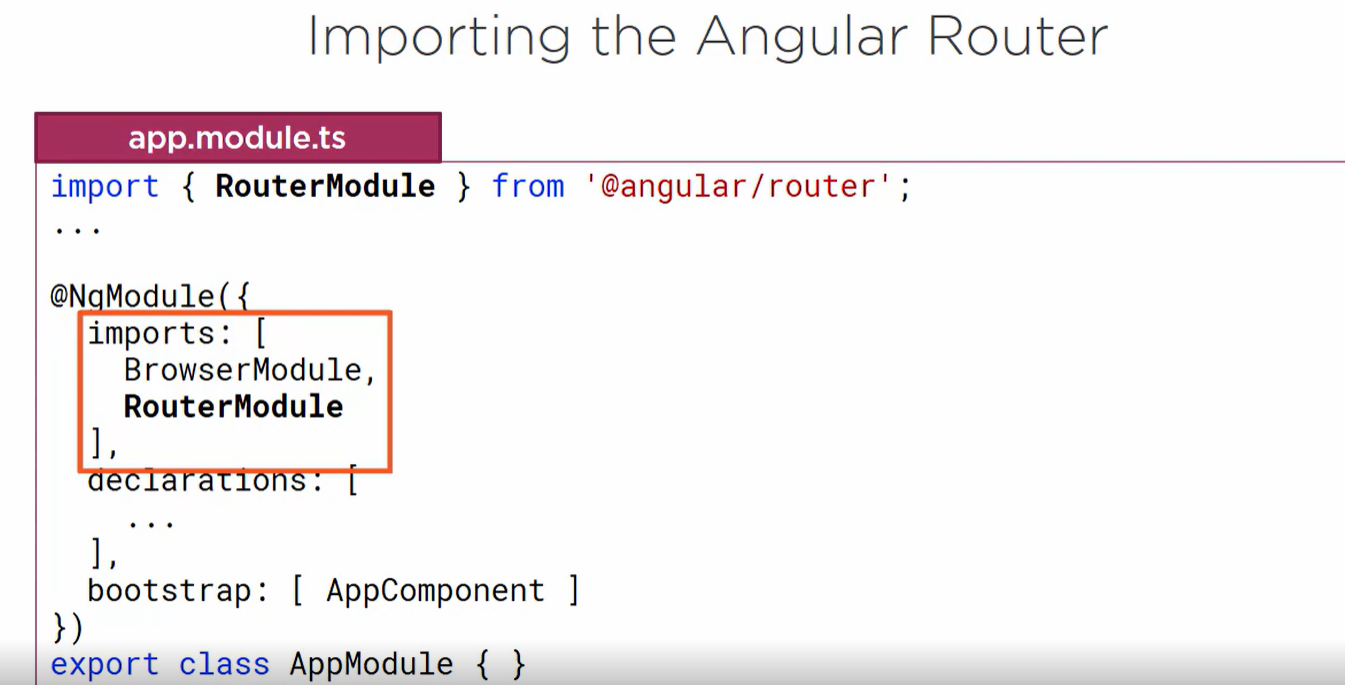
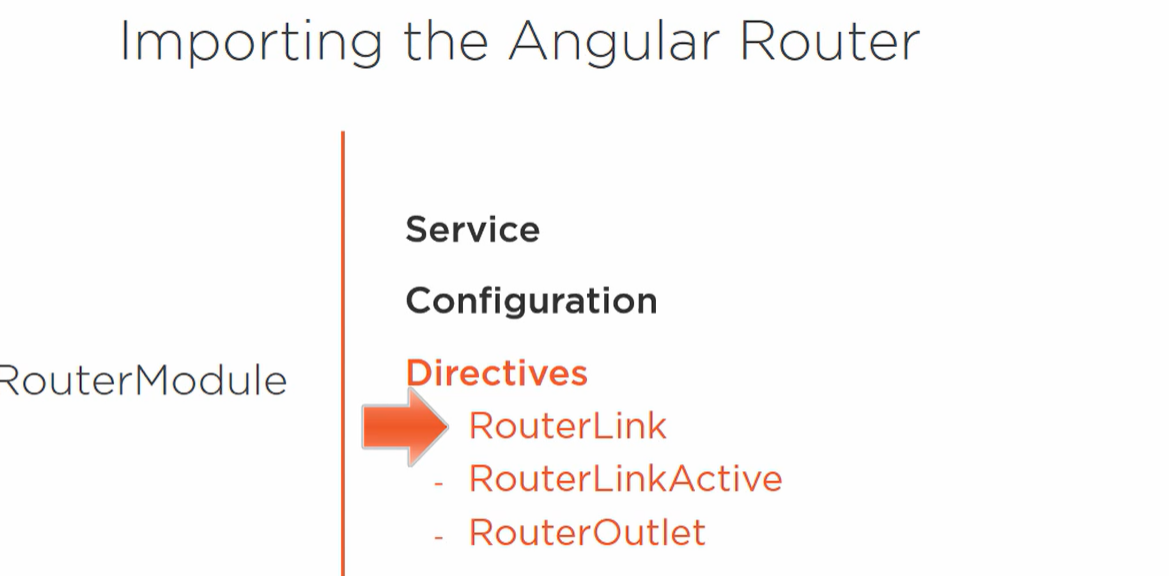
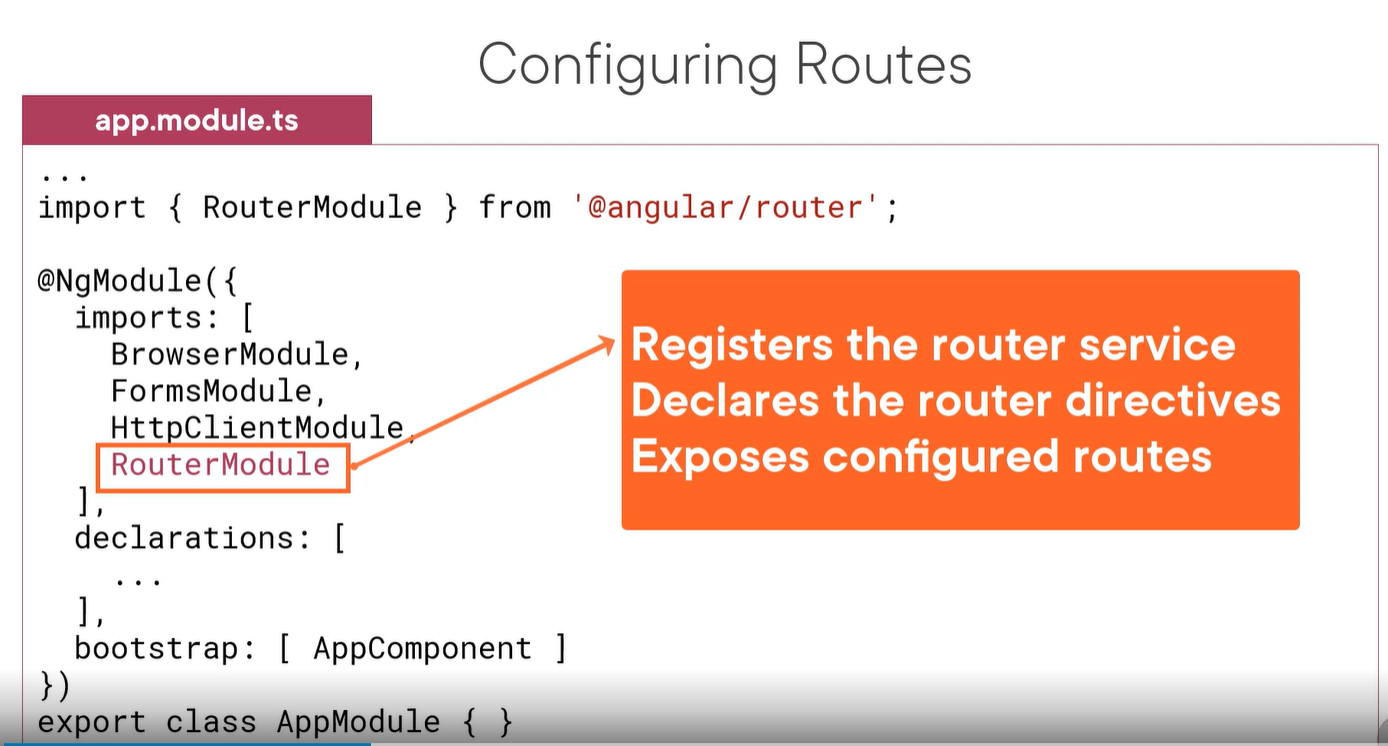
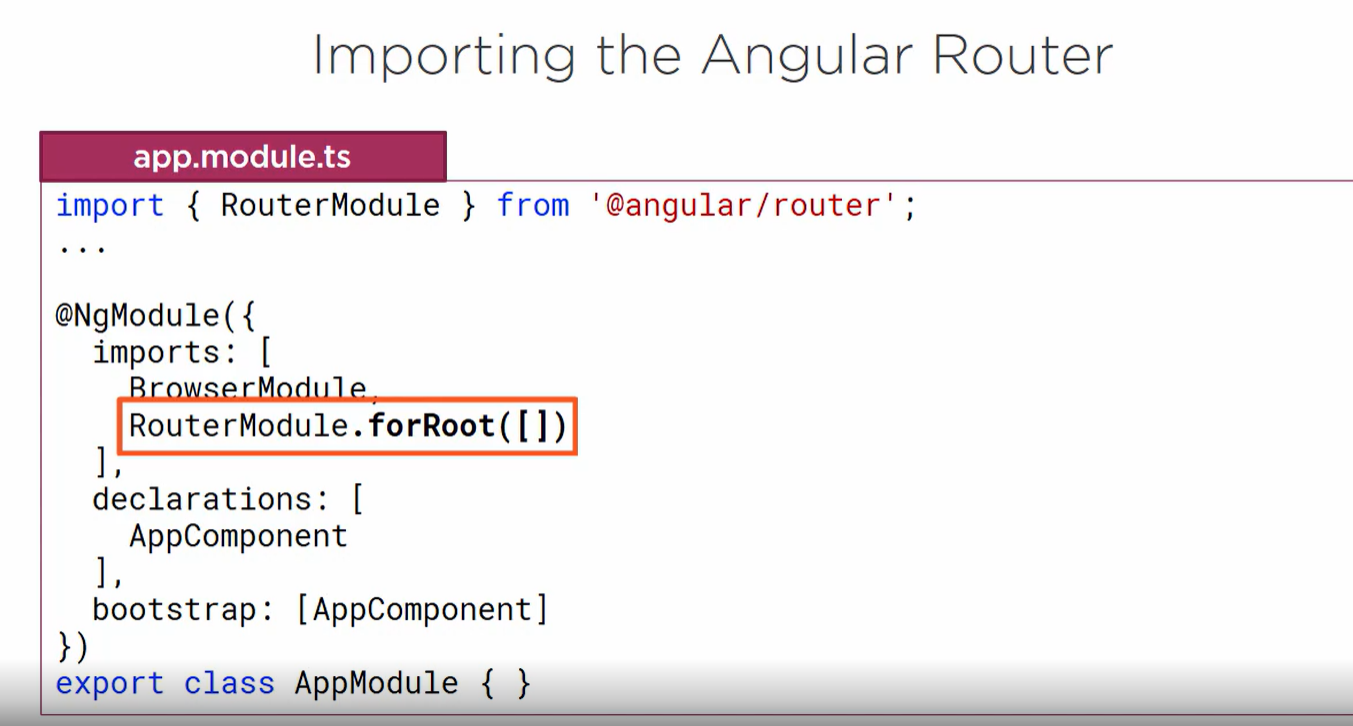
The current state of the router including a tree of the currently activated routes together with convenience methods for traversing the route tree.

How Routing works

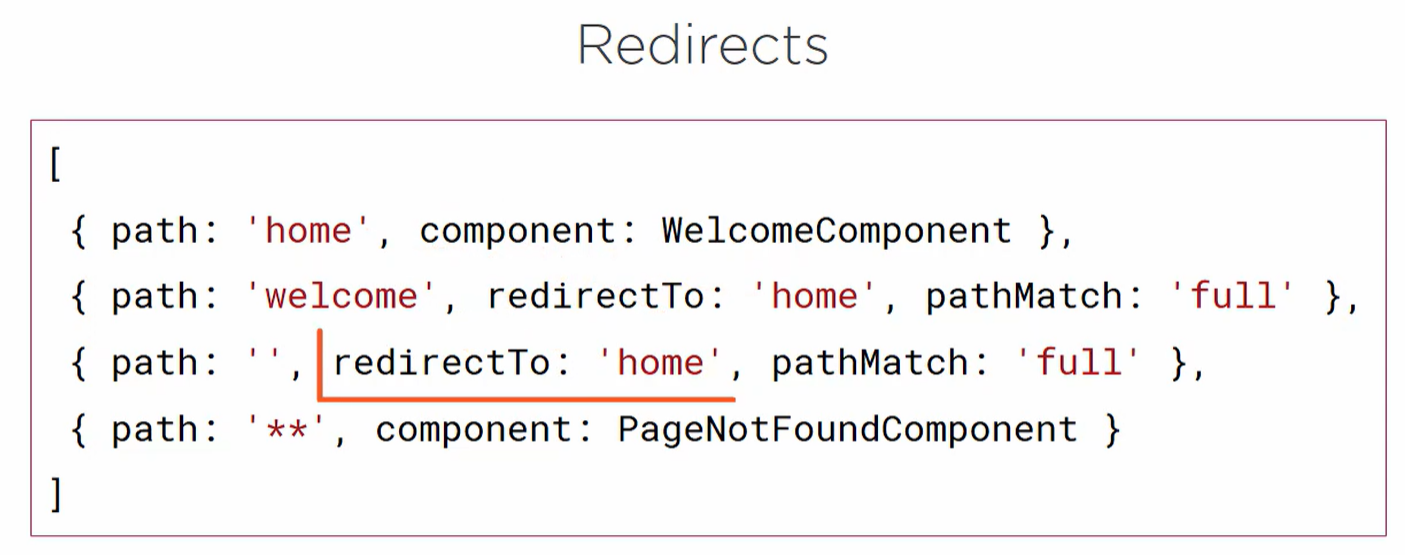
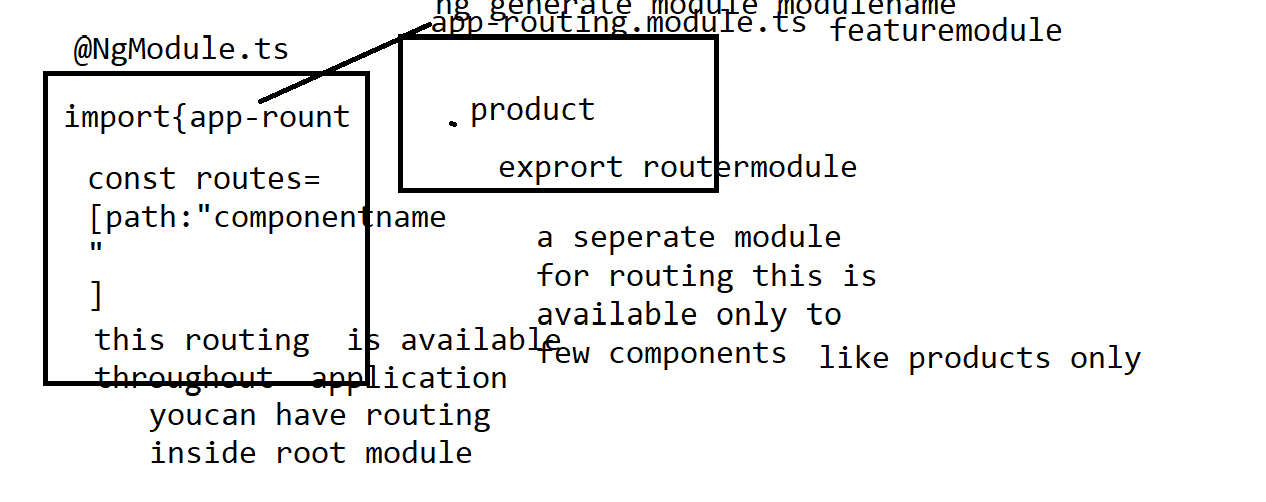
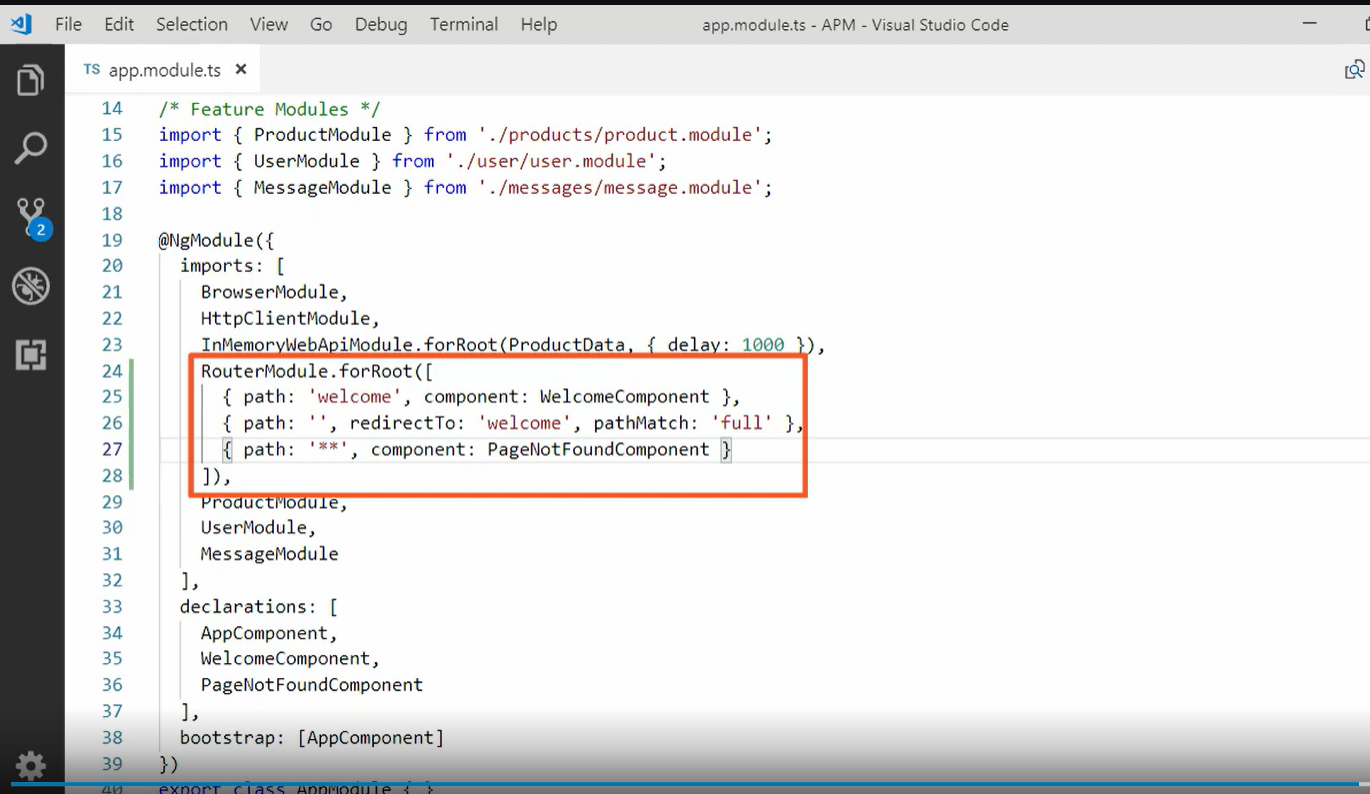
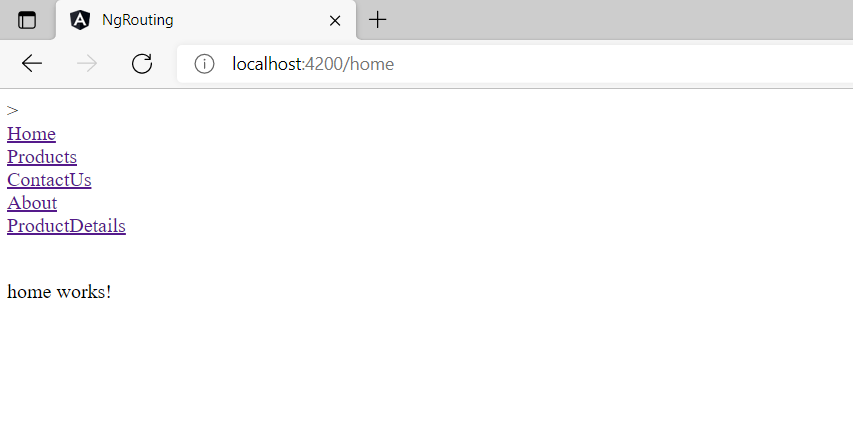




These are the steps of the first section: 

* Step 1: Creating an Angular 12 Project
* Step 2: Understanding routing
* Step 2.1: Adding <base href=”/”> in index.html under head tag
* Step 2.2: Creating a Routing Module
* 
* Step 2.2: Importing the Router and Setting up Routing
* 
* Step 2.3: Adding the Router-Outlet
* **Router Outlet**
* **Now we need to add a**
* **<router-outlet > tag in app.component.html**
* app.component.html
* <router-outlet></router-outlet>
* html
* We can access the configured path by visiting the same URL given in the path variable heading with your host and port.
* Step 2.4: Importing the Routing Module in the Main Application Module
* 
* 
* Step 3: Setting up a Service for Getting Data
* Step 4: Creating a Model
* Step 5: Creating Components
* Step 6: Implementing The Product List Component
* Step 7: Implementing the Product Details Component
* Step 8: Defining the Routes

A typical Angular [Route](https://angular.io/api/router/Route) has two properties:

* path: a string that matches the URL in the browser address bar.
* component: the component that the router should create when navigating to this route.
* 
* A special case of using the full property is when you want to match the empty path. Because using the prefix strategy will match all paths since the empty path prefixes all path
* Routes can be comfigured either in app.module.ts
* Or create separate module as feature module and configure routes
* And refer this feature module in rootmodule
* 
* Below diagram ashows configuring roots in RootModule
* That is NgModule.ts
* 
* Step 10: Adding Navigation Links
* 
* Example-class example
* Craate components
* Produng generate modulects
* Abouts
* Productdetails
* Home
* Contact us
* ng generate module module name
* **import { NgModule, Component } from '@angular/core';**
* **import { CommonModule } from '@angular/common';**
* **import {Routes, RouterModule } from '@angular/router';**
* **import { HomeComponent } from '../home/home.component';**
* **import { ContactUsComponent } from '../contact-us/contact-us.component';**
* **import { AboutComponent } from '../about/about.component';**
* **import { ProductsComponent } from '../products/products.component';**
* **import { ProductDetailsComponent } from '../product-details/product-details.component';**
* **import { PageNotFoundComponent } from '../page-not-found/page-not-found.component';**
* **//import { ProdcutsComponent } from '../prodcuts/prodcuts.component';**
* **const routes=[**
* **{path:'home',component:HomeComponent},**
* **{path:'', redirectTo:'/home',pathMatch:'full'},**
* **{path:'ContactUs',component:ContactUsComponent},**
* **{path:'About',component:AboutComponent},**
* **{path:'Products',component:ProductsComponent},**
* **{path:'ProductDetails',component:ProductDetailsComponent},**
* **{path:'\*\*',component:PageNotFoundComponent},**
* **];**
* **@NgModule({**
* **declarations: [**
* **],**
* **imports: [**
* **[RouterModule.forRoot(routes)],**
* **],**
* **exports: [RouterModule]**
* **})**
* **export class AppRoutingModule { }**
* **App.module.ts**
* **import { NgModule } from '@angular/core';**
* **import { BrowserModule } from '@angular/platform-browser';**
* **import { AppComponent } from './app.component';**
* **import { HomeComponent } from './home/home.component';**
* **import { AboutComponent } from './about/about.component';**
* **import { ContactUsComponent } from './contact-us/contact-us.component';**
* **import { AppRoutingModule } from './app-routing/app-routing.module';**
* **import { ProductsComponent } from './products/products.component';**
* **import { ProductDetailsComponent } from './product-details/product-details.component';**
* **import { Routes,RouterModule, RouteConfigLoadEnd } from '@angular/router';**
* **import { PageNotFoundComponent } from './page-not-found/page-not-found.component';**
* **@NgModule({**
* **declarations: [**
* **AppComponent,**
* **HomeComponent,**
* **AboutComponent,**
* **ContactUsComponent,**
* **ProductsComponent,**
* **ProductDetailsComponent,**
* **PageNotFoundComponent**
* **],**
* **imports: [**
* **BrowserModule,**
* **RouterModule,**
* **AppRoutingModule.ts**
* **],**
* **providers: [],**
* **bootstrap: [AppComponent]**
* **})**
* **export class AppModule { }**
* **app.component,html**
* **<div>**
* **<a routerLink='/home'>Home</a><br>**
* **<a routerLink='/Products'>Products</a><br>**
* **<a routerLink='/ContactUs'>ContactUs</a><br>**
* **<a routerLink='/About'>About</a><br>**
* **<a routerLink='/ProductDetails'>ProductDetails</a><br><br>**
* **</div>**
* **<div>**
* **<router-outlet></router-outlet>**
* **</div>**
* 
* **Create a service to get products**
* **Ng generate Service Prouctlist**
* import { Injectable } from '@angular/core';
* @Injectable({
* providedIn: 'root'
* })
* export class ProductlistService {
* constructor() { }
* GetAllProdoucts():any
* {
* const products:any[] =[
* {productid:100,productname:'tv',price:200,Quantity:1},
* {productid:200,productname:'refrigirator',price:300,Quantity:1},
* {productid:300,productname:'LAPTOP',price:400,Quantity:1},
* {productid:400,productname:'AC',price:500,Quantity:1},
* {productid:500,productname:'FAN',price:600,Quantity:1},
* ]
* return products;
* }
* }
* App.component.ts
* import { Component } from '@angular/core';
* import { ProductlistService } from "./productlist.service";
* @Component({
* selector: 'app-root',
* templateUrl: './app.component.html',
* styleUrls: ['./app.component.css']
* })
* export class AppComponent {
* title = 'ng-routing';
* Products:any[];
* constructor(private \_productlistService:ProductlistService)
* {
* this.Products =this.\_productlistService.GetAllProdoucts();
* }
* }
* App.component.html
* <div>
* <a routerLink='/home'>Home</a><br>
* <a routerLink='/Products'>Products</a><br>
* <a routerLink='/ContactUs'>ContactUs</a><br>
* <a routerLink='/About'>About</a><br>
* <a routerLink='/ProductDetails'>ProductDetails</a><br><br>
* </div>
* <div>
* <router-outlet></router-outlet>
* <div>
* <p>products works!</p>
* <table>
* <thead>
* <tr>
* <th>
* ProductID
* </th>
* <th>
* ProductName
* </th>
* <th>
* Price
* </th>
* <th>
* Quantity
* </th>
* </tr>
* </thead>
* <tbody>
* <tr \*ngFor="let product of Products">
* <td>{{product.productid}}</td>
* <td>{{product.productname}}</td>
* <td>{{product.Quantity}}</td>
* <td>{{product.price}}</td>
* </tr>
* </tbody>
* </table>
* </div>
* </div>

## Location Strategies in Angular Router

Being a Single Page Application, the Angular applications should not send the URL to the server and should not reload the page, every time user requests for a new page.

The URLs are strictly local in Angular Apps. The [Angular router](https://www.tektutorialshub.com/angular/angular-routing-navigation/) navigates to the new component and renders its template and updates the history and URL for the view. All this happens locally in the browser.

There are two ways, by which Angular achieves this. These are called Location Strategies.

The Location Strategy defines how our URL/Request is resolved. It also determines how your URL will look like

Angular supports two Location Strategies:

1. **HashLocationStrategy**  
   Where URL looks like http://localhost:4200/#/product
2. **PathLocationStrategy**  
   Where URL looks like http://localhost:4200/product

Before going further lets first understand what is client-side routing is

### Client-Side Routing

In a Multi-page web application, Every time the application needs to a display a page it has to send a request to the webserver. You can do that by either **typing the URL in the address bar,**

**clicking on the Menu link/**

**button. Every such action results in browser sending a new request to the Web server**

* But, the Angular Applications are single-page applications or SPA.

All the components are displayed on a single page

In a Typical Single Page Application, when the Web application is loaded it loads the single HTML page. Whenever the user interacts with the page, only a part of the page is dynamically updated.

If you open the index.html in any of the angular application, you would see the following HTML markup

|  |  |
| --- | --- |
|  | Index.htnl  <body>      <app-root>Loading...</app-root>  </body> |

The "app-root" is a placeholder (selector), which is defined in the root component.

Angular generates and loads the view associated with the root component inside the "app-root". Any subsequent components are also loaded dynamically inside the "app-root" selector

Angular does all this behind the scenes.

In such a scenario, we are not required to change the URL. But that brings a few cons

* You won’t be able to refresh the page
* You won’t be able to go to a particular view by typing the URL
* Sharing the URL with someone is not possible
* The Back button will not work as you cannot go back to the previous page
* SEO is not possible

*That is where the client-side routing comes into the picture*

The Client-side routing simply mimics server-side routing by running the process in the browser. It changes the URL in the browser address bar and updates the browser history, without actually sending the request to the server

### How Client-Side Routing works

The Client-side routing is handled in two ways

1. Hash style Routing
2. HTML 5 Routing

## Hash Style Routing

The Hash style routing using the anchor tags technique to achieve client-side routing.

The anchor tags, when used along with the # allows us to jump to a place, within the web page.

For Example

**Index.html**

|  |  |
| --- | --- |
|  | <a name="contact">Contact Us</a> |

And we visited the URL **http://mysite.com/index.html#contact,** the browser would scroll to the location of the Contact us label

When the requested anchor tag is on the current page, then the browser does not send the request to the Web server.

The Hashstyle Routing uses this technique to create the URL

The URL would look like something like

|  |  |
| --- | --- |
| 5 | http://www.example.com  http://www.example.com/#/about  http://www.example.com/#/contact |

In all the above examples, only the URL sent to the server is http://www.example.com the URL’s "#/about" and #/contact is never sent to the server

## HTML 5 routing

The introduction of HTML5, now allows browsers to programmatically alter the browser’s history through the history object.

Using [history.pushState()](https://developer.mozilla.org/en-US/docs/Web/API/History_API) method, we can now programmatically add the browser history entries and change the location without triggering a server page request.

The [history.pushState](https://www.sitepoint.com/javascript-history-pushstate/) method accepts the following three parameters.

1. **State object:** A state object is a JavaScript object which is associated with the new history entry created by pushState()
2. **Title:** This is an optional title for the state
3. **URL:** The new history entry’s URL. The browser won’t jump to that page.

For example

|  |  |
| --- | --- |
| 1  2  3  4 | var stateObj= { message: "some message" };  history.pushState(stateObj, "title", newUrl); |

Using history.pushState the method, The browser creates new history entries that change the displayed URL without the need for a new request.

Example

When you request for http://www.example.com the server sends the index.html

Now, When you click on ProductList link, Angular use’s the history.pushState method to push the state and change the URL to http://www.example.com/ProductList

Now, when you click on the specific Product, we again the use history method to push the state and change the URL to http://www.example.com/product/1

Here, when you click the back button, the browser will retrieve the http://www.example.com/ProductList from history and displays it.

But there are cons to this approach

1. Not all browsers support HTML 5
2. The older browser does not support HTML5. So if you want to support older browser, you have to stick to the hash style routing
3. The server support is needed for HTML5 based routing.

#### Why Server Support Needed for HTML 5 routing

Now, consider the above example

What would happen, when you type the URL http://www.example.com/ProductList and hit the refresh button.

The browser will send the request to the webserver. Since the page ProductList does not exist, it will return the 404 (page not found) error.

This problem could be solved, if we are able to redirect all the request to the index.html

It means that when you ask from http://www.example.com/ProductList, the Web server must redirect it to index.html and return the request. Then in the Front-end Angular will read the URL and dynamically load the ProductListComponent.

To make HTML5 routing work you need to send the instruction to the webserver to serve /index.html for any incoming request, no matter what the path is.

## Location Strategy

As mentioned earlier, Angular implements both Hashstyle & HTML 5 Routing. HashLocationstrategy implements the Hashstyle routing & Pathlocationstrategy implements the HTML5 style routing

## PathLocationStrategy Vs HashLocationStrategy

### PathLocationStrategy

**Pros:**

* Produces a clear URL like http://example.com/foo
* Supports Server-Side Rendering

Server-side Rendering is a technique that renders critical pages on the server that can greatly improve perceived responsiveness when the app first loads

Cons:

* Older browser does not support
* Server Support needed for this to work

### HashLocationStrategy

**Pros:**

* Supported by all browsers

**Cons:**

* Produces a URL like http://example.com/#foo
* Will not Support Server-Side Rendering

## PathLocationStrategy

The PathLocationStrategy is the default strategy in Angular application.

To Configure the strategy, we need to add <base href> in the <head> section of root page (index.html) of our application

|  |  |
| --- | --- |
| 1  2  3 | <base href="/"> |

The Browser uses this element to construct the relative URLs for static resources (images, CSS, scripts) contained in the document.

If you do not have access to <head> Section of the index.html, then you can follow either of the two steps

Add the APP\_BASE\_HREF value as shown in the[provider’s](https://www.tektutorialshub.com/angular/angular-providers/) section of the root module

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import {Component, NgModule} from '@angular/core';  import {APP\_BASE\_HREF} from '@angular/common';    @NgModule({  providers: [{provide: APP\_BASE\_HREF, useValue: '/my/app'}]  })  class AppModule {} |

or use the absolute path for all the static resources like CSS, images, scripts, and HTML files.

## HashLocationStrategy

You can use the HashLocationStrategy by providing the useHash: true in an object as the second argument of the RouterModule.forRoot in the AppModule.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | @NgModule({  declarations: [      AppComponent,HomeComponent,ContactComponent,ProductComponent,ErrorComponent  ],  imports: [      BrowserModule,      FormsModule,      HttpModule,      Hashlocationstrategy RouterModule.forRoot(appRoutes, { useHash: true }  ],  providers: [ProductService],  bootstrap: [AppComponent]  }) |

## Which Location Strategy to Use

We recommend you to use the HTML 5 style (PathLocationStrategy ) as your location strategy.

Because

* It produces clean and SEO Friendly URLs that are easier for users to understand and remember.
* You can take advantage of the server-side rendering, which will make our application load faster, by rendering the pages in the server first before delivering it the client

Use hash location strategy only if you have to support the older browsers.

# Routes and Paths

In Angular, a **route** is an object (instance of [Route](https://angular.io/api/router/Route)) that provides information about which component maps to a specific path.

A **path** is the fragment of a URL that determines where exactly is located the resource (or page) you want to access. You can get the path by taking off the domain name from the URL.

In Angular, you can define a route using route configurations or instances of the [Route](https://angular.io/api/router/Route) interface.

A collection of routes defines the router configuration which is an instance of [Routes](https://angular.io/api/router/Routes).

Each route can have the following properties:

* path is a string that specifies the path of the route.
* [pathMatch](https://angular.io/api/router/Route#pathMatch) is a string that specifies the matching strategy. It can take prefix (default) or [full](https://angular.io/api/core/Version#full).
* component is a component type that specifies the component that **should be mapped to the route.**
* [redirectTo](https://angular.io/api/router/Route#redirectTo) is the URL fragment to which you will be redirected if a route is matched.

These are the commonly used properties of routes but there are many others. You can find the rest of the properties from the [official docs](https://angular.io/api/router/Routes#description).

For example, this is the definition of a route that maps the /my/path/ path to the MyComponent component:

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

The path can be the empty string which usually refers to the main URL of your application or can be also a **wildcard** string (\*\*) which will be matched by the router if the visited URL doesn’t match any paths in the router configuration.

This is usually used to display a **page doesn’t exist** message or redirect the users to an existing path.

# Route Matching Strategies

The Angular router has a powerful matching algorithm with various built-in and custom matching strategies.

The built-in matching strategies are **prefix** (the default) and **full.**

When the matching strategy of a route is “**prefix”**, the router will simply check if the start of the browser’s URL is prefixed with the route’s path. If that’s the case, it will render the related component.

This is not always the wanted behavior. In some scenarios, you want the router to match the full path before rendering a component. You can set the full strategy using the pathMatch property of a route. For example:

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

A full strategy ensures that the path segment of the browser’s URL equals exactly the route’s path.

A special case of using the full property is when you want to match the empty path. Because using the prefix strategy will match all paths since the empty path prefixes all paths.

http:localhost:4200/

For example, we want to redirect the user to the /products route when they visit our application. This is how our route configuration should look like:

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

You can also use a custom matcher if the combination of the path property and matching strategy doesn’t help you match your component to a specific URL.

You can provide a custom matcher using the matcher property of a route definition. For an example, see [UrlMatcher](https://angular.io/api/router/UrlMatcher#description).

# Route Parameters

Dynamic routes are often used in web applications to pass data (parameters) or states to the application or between various components and pages. The Angular router has support for dynamic paths and provides an easy-to-use API to access route parameters.

You can define a route parameter using the colon syntax followed by the name of the parameter.

For instance:

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

In the example, id is the route parameter. /product/1, /product/2, /product/p1 … are examples of URLs that will be matched via this route definition.

The last segment of these URLs are the values of the id parameter that will be passed to ProductDetailComponent.

In your matched components, you can access the route parameters using various APIs:

* Using the [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) service,
* Using the [ParamMap](https://angular.io/api/router/ParamMap) Observable available starting with Angular 4.

**Angular Router Navigate**

To **navigate** different routes, use the **Angular** **router** as it provides methods to navigate different routes using routerLink. To navigate from one route to another, we need two Angular components. Each component contains a specific view.

**Navigation Directive**

Angular Router supports the routerLink directive to create navigation links. The navigation directive takes the path associated with the component to navigate to.

**Syntax**

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

In the above code, **‘/home’**is the path. Therefore, we need to map a specific component to the path /**home.**

In Angular 13, there is an**app-routing.module.ts** module file. In that file, you can define the routes array that contains objects. We can define a different **path**and **component in those objects,**and angular will map the path to that component accordingly. So it will render specific

components to a particular path.

### RouteLink Parameters array

The Parameters or arguments to the Route. It is an array which you can bind to RouterLink directive or pass it as an argument to the Router.navigate method.

**Angular 13 routerLink**

If you don’t know how to update to Angular 13, then check out the updated [Angular CLI version](https://appdividend.com/2022/01/26/update-angular-cli/) guide.

To create a new angular 13 project, type the following command.

ng **new** angularguard

While creating a new project, please enable the routing. So it will create the **app-routing.module.ts**file.

Now, create two following components.

1. HomeComponent
2. DashboardComponent

To create a component in Angular, you have to type the following command.

ng g c home --skipTests=true

ng g c dashboard --skipTests=true

After creating a component, we will define the routes array containing different objects.

So, write the following code inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

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

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

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

**import** { DashboardComponent } **from** './dashboard/dashboard.component';

**const** routes: Routes = [

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

{ path: 'dashboard', component: DashboardComponent }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

})

**export** **class** **AppRoutingModule** { }

You can see that we have defined the **routes array** that contains two objects. The keys to the objects are the following.

1. path
2. component

You can map the different routes to different components here and, finally, register all the routes using **RouterModule.forRoot()** function.

**Angular 13 Router-outlet**

The Router-Outlet is the directive that the angular router library provides, where the router adds the component that gets matched based on the current browser’s URL.

You can add multiple router outlets in your application as per your requirements, enabling you to implement advanced routing scenarios. See the Router-outlet syntax.

**<router-outlet></router-outlet>**

**Any component that gets matched by the Router will render it a sibling of the Router outlet.**

Let’s write **<router-outlet>**directive inside the **app.component.html**file in our example.

<h3>Welcome to router navigate</h3>

<router-outlet></router-outlet>

Now, you have three URLs to render. One is the root, and the other two, we have defined in the routes array. You can access the components by the following URLs.

1. **http://localhost:4200/**
2. **http://localhost:4200/home**
3. **http://localhost:4200/dashboard**

Right now, there is no navigation bar. So let’s add bootstrap navigation by installing bootstrap 4.

npm install **bootstrap** --**save**

Register the **bootstrap.min.css** file inside the **angular.json** file.

"styles": [

"src/styles.css",

"./node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

Now, we will add the navigation bar inside the **app.component.html**file.

<nav class="navbar navbar-light navbar-expand-lg" style="background-color: #e3f2fd;">

<div class="container">

<a class="navbar-brand">AppDividend</a>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link">

Home

</a>

</li>

<li class="nav-item">

<a class="nav-link">

Dashboard

</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container">

<router-outlet></router-outlet>

</div>

You can see that we have not written a routerLink directive inside the **anchor tag.**

To navigate from one route to another route, we have to put the **routerLink** directive.

The **routerLink** directive takes a route path that we have already defined inside the **routes array.**

Let’s add the directive.

*<!-- app.component.html -->*

<nav class="navbar navbar-light navbar-expand-lg" style="background-color: #e3f2fd;">

<div class="container">

<a routerLink="/" class="navbar-brand" routerLinkActive="active" >AppDividend</a>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav ml-auto">

<ng-container>

<li class="nav-item">

<a routerLink = "home" class="nav-link">

Home

</a>

</li>

<li class="nav-item">

<a routerLink = "dashboard" class="nav-link">

Dashboard

</a>

</li>

</ng-container>

</ul>

</div>

</div>

</nav>

<div class="container">

<router-outlet></router-outlet>

</div>

Save the file, and now you can navigate easily from one route to another.

**Navigating Programmatically via Angular Router**

To navigate programmatically in Angular, use the Router service we inject into our component.

The Angular 13 Router service provides two methods that you can use to navigate from one component to other components in your component class instead of using the RouterLink directive in the template, as we have just seen.

The two methods are the following.

1. navigate()
2. navigateByUrl()

They can be helpful in multiple programming scenarios where you need to trigger the navigation via code. Both functions return a promise that resolves to either true or false.

The **navigateByUrl()** method takes a string as a parameter.

The **navigate()** method takes an array of URL segments.

**Angular router navigate() method.**

The router navigates () method accepts the same one-item link parameters array that you can bind to the [routerLink] directive.

To navigate programmatically in angular, use the router navigate() method.

In our example, let’s see how to use the router navigate() method.

Write the following code inside the **home.component.ts**file.

*// home.component.ts*

**import** { Component, OnInit } **from** '@angular/core';

**import** { Router } **from** '@angular/router';

**@Component**({

selector: 'app-home',

templateUrl: './home.component.html',

styleUrls: ['./home.component.css']

})

**export** **class** HomeComponent **implements** OnInit {

**constructor**(**private** router: Router) { }

ngOnInit(): void {

setTimeout(() => {

**this**.router.navigate(['/dashboard']);

}, 1000);

}

}

In this example, we programmatically navigate the route from **/home** to **/dashboard.**

The **ngOnInit()** function automatically gets called when the component is initialized, and then after waiting for 1 second, the router programmatically navigates to the **/dashboard**route using the router.navigate() function.

So, go to the home route using the navbar, and then after a second, you will navigate to the **/dashboard** route. You can programmatically navigate to a specific route in Angular based on your custom condition.

**Angular router.navigateByUrl()**

The **router.navigateUrl()** function takes redirect URL.

In our scenario, we can use the **router.navigateByUrl()** function instead of router.navigate() function.

Write the following code inside the **home.component.ts**file.

*// home.component.ts*

ngOnInit(): **void** {

setTimeout(() => {

**this**.router.navigateByUrl('/dashboard');

}, 1000);

}

Save the file, and you will get the same output. However, you will be redirected to **/dashboard**route after a second**.**

**Conclusion**

Navigating routes in single-page applications are the most common programming implementation. In this tutorial, we have seen how to navigate different routes using routerLink, a standard way, and **router.navigate()** or **router.navigateByUrl()** function, which is programmatically in Angular.

**Angular 13 Routing and Sub Routing**

To navigate to different pages in your application, you also want the application to be a **SPA** (Single Page Application), with no page reloading. Your app needs routing, and angular makes it very easy. Routing means navigating between the pages. You have seen many websites with links that direct you to the new page. This can be achieved using routing.

We also see the sub routing or children routing for our components. That means, in our application, there is one root route, and other routes are for their respective components.

4.3M

Type the following command to create it. Then, please install or update [Angular CLI](https://appdividend.com/2020/02/18/how-to-update-angular-cli-to-version-9-angular-9-cli-upgrade/) if you have not done it already.

PLAY

UNMUTE

Loaded: 1.16%

FULLSCREEN

CANCEL

**Example-1**

most important parts of the Angular router, and we'll build a practical example along the way.

## 1. Setting Up the Angular Router

First things first, you must have the Angular router installed in your project.

The Angular router is an extremely powerful JavaScript-based router. It is created and managed by the core Angular team.

The router can be installed easily from the @angular/router package. Installing this package will give your project complete access to all the routing features you would need: from router guards to path matching methods to the creation of several router outlets.

### Step 1: Install the @angular/router Package

There are two different ways of installing the @angular/router package, depending on the package manager you use in your project.

**With NPM:**

|  |  |
| --- | --- |
| **1** | **npm i --save @angular/router**  **npm install -routing --save @angular/router**  **generate angular routing module**  **ng generate app-routing - -flat - -module=app**  **the flat option preventsthe module from being**  **in its folder and the module option will be sure to**  **import the routing module in appmodule.ts**  **D:\Angular\AngularLearning\AngularRouting\ng-routing>ng generate app-routing --flat--module=app**  **An unhandled exception occurred: Schematic "app-routing" not found in collection "@schematics/angular".**  **See "C:\Users\IT\AppData\Local\Temp\ng-LCttxn\angular-errors.log" for further details**  **Install package**  **ng add @oktadev/schematics@3.4.1** |

### Step 2: Set Up the Base Location in index.html

Before you use the features of the @angular/router package, you need to set the root route of the application in your **index.html** file. If you look at the root folder of your application, you will find the **index.html** file. Here, add the base location using the base tag. We are going to use the app folder as the root of the application using the "/" path.

Making this change tells Angular the starting location for all the different routes in your application.

Index.html

|  |  |
| --- | --- |
|  | **<!doctype html>**  **<html>**  **<head>**  **<base href="/">**  **<title>Angular Routing</title>**  **</head> .**  **</html>** |

### Step 3: Adding Dependencies to app.module.ts

If you are going to make use of the @angular/router package, you have to import the router into NgModule. The main App Module of your application must have the RouterModule. In our case, the **app.module.ts** file has the NgModule declaration. This file has to be modified with the following components:

|  |  |
| --- | --- |
|  | import { NgModule } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  **import { RouterModule } from '@angular/router';**    import { AppComponent } from './app.component';    @NgModule({    imports: [BrowserModule, RouterModule],    bootstrap: [AppComponent],    declarations: [AppComponent],  })  export class AppModule {} |

Advertisement

### Step 4: Configuring RouterModule

RouterModule needs information about the routes in your single-page application. Until this information is provided, RouterModule will not be useful. RouterModule has two static methods which can be used to provide the required configuration of the routes in the application to the router. This is when the entire setup gets completed.

When routes are being defined for the root configuration of the application, a static method called RouterModule.forRoot can be used. This will give the main NgModule access to the multiple router directives in your application. Here is a small snippet to show how RouterModule.forRoot can be used.

|  |  |
| --- | --- |
| 11  12  13 | import { Routes, RouterModule } from '@angular/router';  .  .  .  export const ROUTES: Routes = [];    @NgModule({    imports: [BrowserModule, RouterModule.forRoot(ROUTES)],    .    .    .  })  export class AppModule {} |

Here, we pass an array of routes to the configuration. It is best practice in Angular routing to use a variable or a constant inside forRoot. In many cases, it would be most readable and easiest to maintain if the routes are stored in a separate file and imported into the AppModule of your application. We'll look at how to specify the ROUTES array below.

Next, we have RouterModule.forChild, which is extremely similar to the functionality offered by RouterModule.forRoot. The main difference is that RouterModule.forChild can be treated like a feature module. It saves developers the effort of defining all the required routes in a single page as does forRoot. Modules within the application can define and configure their own routes. Based on need, these routes can be imported into the main module.

Here is a small code snippet where we use RouterModule.forChild.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | import { NgModule } from '@angular/core';  import { Routes, RouterModule } from '@angular/router';    export const ROUTES: Routes = [];    @NgModule({  **imports: [RouterModule.forChild(ROUTES)],**    .    .    .  })  export class ChildModule {} |

### Step 5: Loading Components

Now we've finished the basic configuration of our routes. The next step is to help Angular load components. For this, we make use of a directive called **router-outlet**.

The moment our router identifies a component to load for a route, the component gets created dynamically. And the dynamically created component is injected along the router-outlet element. The newly created component is treated as a sibling for the router-outlet element. The router-outlet directive can be inserted anywhere in your main application component.

Meanwhile, if you are building a simple AppComponent, you can inject the directive as follows:

|  |  |
| --- | --- |
| **01**  **02**  **03**  **04**  **05**  **06**  **07**  **08**  **09**  **10**  **11**  **12** | **import { Component } from '@angular/core';**    **@Component({**  **selector: 'appComponent',**  **template: `**  **<div>**  **<h1>Angular Routing</h1>**  **<router-outlet></router-outlet>**  **</div>**  **`,**  **})**  **export class AppComponent {}** |

And that's where the setup comes to an end. Now, we can look into the specific needs of the project and start building routes.

### Step 6: Creating Routes for a Single-Page Application

We will be building the routes for our application in the following file: **app.routing.ts**. When you build a single-page application, it is crucial to have a static route. In our case, the static route would be the home page loaded by the application.

Here is our routes array:

|  |  |
| --- | --- |
| 1  2  3  4  5 | export const AppRoutes: Routes = [      { path: '', component: HomeComponent },      { path: 'calc', component: CalcComponent },      { path: '\*\*', component: NotFoundComponent }  ]; |

* The empty path denotes the static route.
* The calc path will be used to load the calculator component we built in the first part of this series.
* The wildcard path \*\* denotes what should be loaded in case there is no other matching route.

Advertisement

### Step 7: Let's Navigate

With the above created route and configuration changes, we are ready to navigate. Let's start by creating a main component for our application called HomeComponent. In our **home.component.html** file, you would see two new directives: routerLink and routerLinkActive.

routerLink is an Angular directive where you can route from one route to another without changing the value in the URL bar. If you want to create a dynamic route, you need to wrap routerLink and pass an array. The syntax for a dynamic router link is [routerLink]=['/path', variable].

Here's our **home.component.html** file with a static router link.

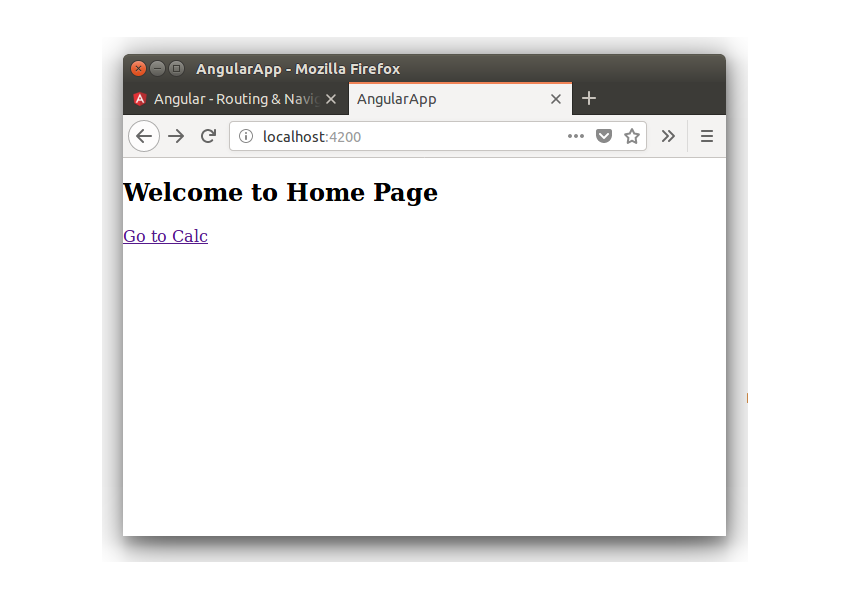
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <h2>      Welcome to Home Page  </h2>    <nav>        <a routerLink="/calc" routerLinkActive="active">Go to Calc</a>  </nav> |

The routerLinkActive directive is used when you want to tell the user which is active. This directive needs to be used along with routerLink.

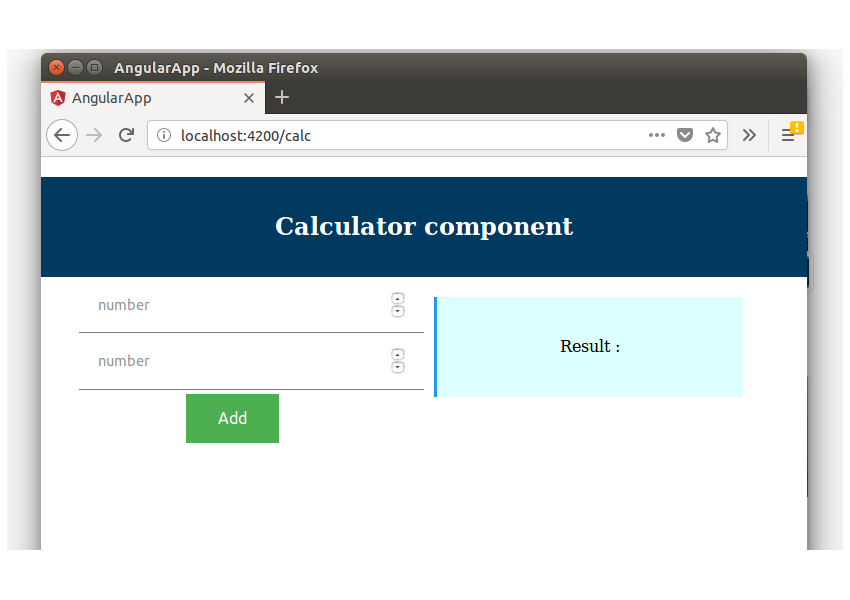
And here's what the corresponding **home.component.ts** file looks like.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import { Component } from '@angular/core';    @Component({    selector: 'home',    templateUrl: 'home.component.html'  })  export class HomeComponent {} |

By default, when you load the application, the HomeComponent gets loaded. Why? Because the '' path corresponds to the HomeComponent.



When you navigate to /calc, the following view will appear. Why? Because /calc loads the CalcComponent in our **app.routing.ts** component.



Now, you have understood the basics of routing, You have created a landing page for your application, and a route! Next, we are going to cover a few advanced concepts in Angular routing.

## Advanced Routing Concepts

### Dynamic Route Parameters

In single-page applications, there is a chance that you would reach a specific route with several operations in mind. For example, in our calculator application, the CalcComponent can be launched only for "addition", "subtraction", or "division". We can achieve this with the help of a colon (:) in the URL segment. The colon tells the router that the value entered is not a part of the URL, but is instead a route parameter.

|  |  |
| --- | --- |
| 1  2  3  4  5 | export const AppRoutes: Routes = [      { path: '', component: HomeComponent },      { path: 'calc/:operation', component: CalcComponent },      { path: '\*\*', component: NotFoundComponent }  ]; |

What we have established is a dynamic route. Now, when we create a link or type an address in the location bar, we can easily swap :operation for any of the operations like addition, subtraction, or division.

Now comes our next challenge. How would you extract the dynamic route information from the route URL? Most of the time, we make use of ngOnInit. If you have a moment, do spend some time on our [**post about Angular Components**](https://code.tutsplus.com/tutorials/beginners-guide-to-angular-4-components--cms-29674) where you'll read about ngOnInit and its role in the lifecycle of a component.

In order to access dynamic route parameters, we make use of ActivatedRoute.

ActivatedRoute contains an Observable called params. By subscribing to params, we can easily extract the route parameters. The best place to subscribe would be ngOnInit. When we route from one parameter to another, the actual route component will not be destroyed. This is a technique Angular uses to boost performance. When you subscribe at ngOnInit, you will receive a notification every time the router parameter changes.

Here is a simple piece of code to help you understand the use of ActivatedRoute.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17 | //the Route is defined as  path = '/calc/:operation'    import { Component, OnInit } from '@angular/core';  import { ActivatedRoute } from '@angular/router';    @Component({    selector: 'calc.component',    templateUrl: './calc.component.html'  })  export class CalcComponent implements OnInit {    operation: string;    constructor(private route: ActivatedRoute) {}    ngOnInit() {      this.route.params.subscribe((params) =&gt; this.operation = params.operation);    }  } |

### Creating Child Routes

So far, we have seen routes that stop with a single level. Now, let's define routes that go into multiple levels. For instance, a single-level route would be /calc, and a multi-level route would be /calc/addition.

We can assign /calc its very own component, and /calc/addition will have another component. To achieve this behavior, we make use of children with the path.

Additionally, to make the child routes work, it is important to include <router-outlet> in the parent component, where the routes would be rendered from.

Sometimes, we may not use the parent route at all for a specific component. In this case, we can remove the component from our route declaration. This is also known as a componentless route.

Here's a simple example to help you understand these concepts.

|  |  |
| --- | --- |
| 41 | //DEFINING THE ROUTE  export const ROUTES: Routes = [    {      path: 'calc',      component: CalcComponent,      children: [        { path: 'addition', component: CalcAdditionComponent },        { path: 'subtraction', component: CalcSubstractionComponent },      ],    },  ];    //COMPONENT-LESS ROUTE  export const ROUTES: Routes = [    {      path: 'calc',      children: [        { path: 'addition', component: CalcAdditionComponent },        { path: 'subtraction', component: CalcSubstractionComponent },      ],    },  ];  @NgModule({    imports: [BrowserModule, RouterModule.forRoot(ROUTES)],  })      //BUILDING THE PARENT COMPONENT  import { Component } from '@angular/core';    @Component({    selector: 'calc.component',    template: `    <div>        <calc-addition></calc-addition>        <calc-subtraction></calc-subtraction>        <router-outlet></router-outlet>      </div>    `,  })  export class CalcComponent {} |

### Fetching Routes From Different Modules

As your application becomes bigger, handling routes can be a challenge. There will be times when you need to fetch routes from different modules. In such cases, you can make use of loadChildren. This is when RouterModule.forChild() becomes useful.

Let's begin with an example:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //CHILD ROUTE  export const ROUTES: Routes = [    {      path: '',      component: CalcComponent,      children: [        { path: 'addition', component: CalcAdditionComponent },        { path: 'subtraction', component: CalcSubtractionComponent },      ],    },  ];    @NgModule({    imports: [CommonModule, RouterModule.forChild(ROUTES)],  })  export class CalcModule {}    //PARENT ROUTE  export const ROUTES: Routes = [    {      path: 'calc',      loadChildren: './calc/calc.module#CalcModule',    },  ];    @NgModule({    imports: [BrowserModule, RouterModule.forRoot(ROUTES)],  })  export class AppModule {} |

A few interesting aspects to be understood from the above code are:

* We have an empty path. This is because we don't want the route path to be /calc/calc.
* The CalcModule has to be imported in the Root AppModule.
* Angular comes with intuitive concepts like **Lazy Loading**. By using the above workflow, you can make sure that the child components are loaded only when the user goes to /calc/\*\*.

**Example-2**

**Step 1: Install the Angular 12 Project**

Type the following command to create it.

ng **new** angroute

Remember, you need to add the app routing by saying yes to the prompt when creating a new project. Here I have allowed add Angular routing.

Now, install the bootstrap CSS framework.

npm install **bootstrap** --**save**

Add the Bootstrap file inside the **angular.json**file.

"styles": [

"src/styles.css",

"./node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

The next step is to create one header component. So type the following command.

**ng g c header --spec=false**

We will create a navigation bar inside that component. So, write the following code inside the **header.component.html**file.

*<!-- header.component.html -->*

<nav class="navbar navbar-expand-lg navbar-light" style="background-color: #e3f2fd;">

<a class="navbar-brand brand-custom" href="#">Angular 12 Routing Example</a>

<div class="collapse navbar-collapse" id="navbarText">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link login-custom" href="#">Students <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link register-custom" href="#">Home</a>

</li>

</ul>

</div>

</nav>

Now, finally, replace the **app.component.html**code with the following code.

*<!-- app.component.html -->*

<div>

<app-header></app-header>

</div>

Save the file and start the angular development server.

ng serve *--open*

You will see the navigation bar with three nav items.

So here, one item is Home, and one is **Students.**

That means our application has one root route for Home, and others are sub-routes like for students module.

When creating the project, we have created one routing module called **app-routing.module.ts.**So we will define the Root routes inside that file.

**Step 2: Add Root Routes**

First, create a home component by the following command.

**ng g c home --spec=false**

Now, add that component inside the **app-routing.module.ts**file.

*// app-routing.component.ts*

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

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

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

**const** routes: Routes = [

{

path: 'home',

component: HomeComponent

}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

**export** **class** **AppRoutingModule** { }

So, here we have defined the root routes for our angular application. Now add the **router-outlet** inside the **app.component.html**file to display the content of the home component.

*<!-- app.component.html -->*

<div>

<app-header></app-header>

<div class="container">

<router-outlet></router-outlet>

</div>

</div>

Also, add the navigation link inside the **header.component.html**file.

*<!-- header.component.html -->*

<li class="nav-item">

<a class="nav-link register-custom" routerLink="/home">Home</a>

</li>

Save the file and go to the browser and click the **Home**link. You can see that we can see the content of the **home.component.html**file. So, we have taken care of the Root routes. Now, it is time to create a student module and define the student module’s sub-routes.

**Step 3: Create a student module and components.**

The first step is to create a module called the student. So let us create using the following command.

ng g **module** **student**

So, it will create a folder inside the app folder called the **student**, and inside that folder, it will create a **student.module.ts** file.

The next step is to create the three angular components related to the student module. So let us do that.

ng g c student/student --spec=false

ng g c student/student-list --spec=false

ng g c student/student-detail --spec=false

It will create the three folders inside the **src >> app >> student**folder.

Now, all these four components are already imported inside the **student.module.ts**file.

// student.module.ts

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

**import** { CommonModule } **from** '@angular/common';

**import** { StudentComponent } **from** './student/student.component';

**import** { StudentListComponent } **from** './student-list/student-list.component';

**import** { StudentDetailComponent } **from** './student-detail/student-detail.component';

@NgModule({

declarations: [StudentComponent, StudentListComponent, StudentDetailComponent],

imports: [

CommonModule

]

})

**export** **class** **StudentModule** { }

We do not need to import all these components inside the **app.module.ts**file.

Instead, we need to import this **student.module.ts**file inside the **app.module.ts**file.

// app.module.ts

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

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

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

**import** { StudentModule } **from** './student/student.module';

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

**import** { HeaderComponent } **from** './header/header.component';

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

@NgModule({

declarations: [

AppComponent,

HeaderComponent,

HomeComponent

],

imports: [

BrowserModule,

AppRoutingModule,

StudentModule

],

providers: [],

bootstrap: [AppComponent]

})

**export** **class** **AppModule** { }

So, all of our student components are registered to the angular application.

**Step 4: Create a Student route.**

Now, inside the **src >> app >> student**folder, we can create a routing file called **student-routing.module.ts**and add the following code inside it.

*// student-routing.module.ts*

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

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

**import** { StudentComponent } **from** './student/student.component';

**import** { StudentListComponent } **from** './student-list/student-list.component';

**import** { StudentDetailComponent } **from** './student-detail/student-detail.component';

**const** routes: Routes = [

{

path: 'student',

component: StudentComponent,

children: [

{

path: 'list',

component: StudentListComponent

},

{

path: 'detail',

component: StudentDetailComponent

}

]

}

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

**export** **class** **StudentRoutingModule** { }

So, here we have defined the sub-routing for the student module. The main path is a  **/student**and its children are **/student/list**and **/student/detail.**

So that means we have defined the subroutes for the student module. The only remaining thing is to register this routing module to the **student.module.ts**file.

Remember, both **student.module.ts**and **student-routing.module.ts**files are different. You can see this structure as same as our root angular project structure like **app.module.ts**and **app-routing.module.ts.**

// student.module.ts

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

**import** { CommonModule } **from** '@angular/common';

**import** { StudentRoutingModule } **from** './student-routing.module';

**import** { StudentComponent } **from** './student/student.component';

**import** { StudentListComponent } **from** './student-list/student-list.component';

**import** { StudentDetailComponent } **from** './student-detail/student-detail.component';

@NgModule({

declarations: [StudentComponent, StudentListComponent, StudentDetailComponent],

imports: [

CommonModule,

StudentRoutingModule

]

})

**export** **class** **StudentModule** { }

Now, we need to display the routes. So add the following code inside the **student.component.html**file.

*<!-- student.component.html -->*

<div class="container">

<router-outlet></router-outlet>

</div>

This **router-outlet**will only show the component related to the **student**module. So it is different from the root routing’s router-outlet, which is still in the place inside the **app.component.html**file.

Also, now add the router link inside the **header.component.html**file.

*<!-- header.component.html -->*

<nav class="navbar navbar-expand-lg navbar-light" style="background-color: #e3f2fd;">

<a class="navbar-brand brand-custom" href="#">Angular 12 Routing Example</a>

<div class="collapse navbar-collapse" id="navbarText">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link login-custom" routerLink="/student/list">Students <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link register-custom" routerLink="/home">Home</a>

</li>

</ul>

</div>

</nav>

Save the file to the browser and navigate to the **http://localhost:4200/student/list**

You can see that it is rendering the correct component. Now, go to the **http://localhost:4200/student/detail**

It will also show the right component, and now our student module is working.

You can still go to **http://localhost:4200/home**, and it will render the correct component, which is HomeComponent.

This is how you can organize your project [Angular](https://appdividend.com/2018/12/09/how-to-create-angular-modules-to-organize-code/)module-wise with the root and children routing.

**Summary of Angular Routing**

1. You added the Angular router to navigate among different components.
2. You turned the AppComponent into a navigation shell with <[a](https://angular.io/api/router/RouterLinkWithHref)> links and a <[router-outlet](https://angular.io/api/router/RouterOutlet)>.
3. You configured the router in an **AppRoutingModule**.
4. You configured the router in the **StudentRoutingModule.**
5. You defined simple routes as redirect routes.
6. You used the [routerLink](https://angular.io/api/router/RouterLink) directive in anchor elements.

Angular RouterLink: The Complete Guide

[RouterModule](https://angular.io/api/router/RouterModule) provides RouterLink directives in Angular.

The RouterModule adds router directives and providers. RouterLink lets us link to the specific routes in our angular application.

**Angular RouterLink**

RouterLink is a built-in Angular Directive that lets you link to specific routes in your app. In the SPA(single-page application), you change what the user sees by showing or hiding portions of the display that correspond to particular components, rather than going out to the server to get a new page.

As users need to perform application tasks, they must move between the views you have defined. To implement navigation between different routes within your single-page app, use the Angular Router.

Consider the following route

configuration: **[{ path: ‘customer/:id’, component: CustComponent }]**.

When linking to this **customer/:id** route, you use the **RouterLink** directive.

If there is a static link, you can use the directive as follows: **<a routerLink=”/customer/21″>**link to cust component</a>

If you use dynamic values to generate a link, you can pass the array of path segments, followed by the parameters for each segment.

For instance [‘/group’, groupId, ‘user’, userName, {full: true}] means that we want to generate a link to **/group/21/user/billy;full=true**.

**Routing segments**

The first segment name can be prepended with **/**,  **. /**, or **. . /**:

1. If the first segment begins with **/**, the router will look up the route from the app’s root.
2. Suppose the first segment begins with **. Then,/**, or doesn’t start with a slash, the router will instead look in the children of the current activated route.
3. And if the first segment begins with **. . /**, the router will go up one level.

**Set Query Parameters in Angular**

To set query parameters in Angular, use the following syntax.

<**a** [routerLink]="['/customer/jt']" [queryParams]="{purchase: true}">

link to customer component

</**a**>

RouterLink will use these to generate this link: **/customer/jt;purchase=true.**

You can provide a [state](https://angular.io/api/animations/state) value to be persisted to the browser’s **History.state** property.

<**a** [routerLink]="['/customer/jt']" [state]="{transactionId: 123}">

link to customer component

</**a**>

**Method of RouterLink**

The RouterLink provides one method called **onClick().**

The onClick() method does not accept a single parameter and returns a Boolean value.

**Example of RouterLink directive**

To create a new Angular project, type the following command.

While creating a new project, please enable the routing by selecting y in the options.

ng **new** angularguard

You can name your project whatever you want.

Now, go inside the project folder and install Bootstrap CSS Framework.

npm install **bootstrap** --**save**

Register the CSS file inside the **angular.json**file.

"styles": [

"src/styles.css",

"./node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

Now, create the following two new Angular components.

1. HomeComponent
2. DashboardComponent

Please type the following commands to create it.

ng g c home --skipTests=true

ng g c dashboard --skipTests=true

After creating components, we will define the routes array containing different objects.

So, write the following code inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

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

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

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

**import** { DashboardComponent } **from** './dashboard/dashboard.component';

**const** routes: Routes = [

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

{ path: 'dashboard', component: DashboardComponent }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

})

**export** **class** **AppRoutingModule** { }

You can map the different routes to different components here and, finally, register all the routes using **RouterModule.forRoot()** function.

The Router-Outlet is the directive provided by the angular router library where the router adds the component that gets matched based on the current browser’s URL.

Let’s write **<router-outlet>**directive inside the **app.component.html**file in our example. We will also add the navigation bar inside the **app.component.html**as well.

<nav class="navbar navbar-light navbar-expand-lg" style="background-color: #e3f2fd;">

<div class="container">

<a class="navbar-brand">AppDividend</a>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link">

Home

</a>

</li>

<li class="nav-item">

<a class="nav-link">

Dashboard

</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container">

<router-outlet></router-outlet>

</div>

You can see that we have not written a routerLink directive inside the **anchor tag.**

To navigate from one route to another, we must put the **routerLink** directive.

The **routerLink** directive takes a route path that we have already defined inside the **routes array.**

Let’s add the directive.

*<!-- app.component.html -->*

<nav class="navbar navbar-light navbar-expand-lg" style="background-color: #e3f2fd;">

<div class="container">

<a routerLink="/" class="navbar-brand" routerLinkActive="active" >AppDividend</a>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav ml-auto">

<ng-container>

<li class="nav-item">

<a routerLink = "home" class="nav-link">

Home

</a>

</li>

<li class="nav-item">

<a routerLink = "dashboard" class="nav-link">

Dashboard

</a>

</li>

</ng-container>

</ul>

</div>

</div>

</nav>

<div class="container">

<router-outlet></router-outlet>

</div>

Okay, so now we have two views, and we can navigate between those views using routerLink.

We can also write the **routerLink**directive like this.

<ng-container>

<li class="nav-item">

<a [routerLink] = "['/home']" class="nav-link">

Home

</a>

</li>

<li class="nav-item">

<a [routerLink] = "['/dashboard']" class="nav-link">

Dashboard

</a>

</li>

</ng-container>

**Pass Query Parameters in Angular**

To pass the query parameters, you can write the anchor tag.

<ng-container>

<li class="nav-item">

<a [routerLink] = "['/home']" [queryParams] = "{success: true}" class="nav-link">

Home

</a>

</li>

<li class="nav-item">

<a [routerLink] = "['/dashboard']" class="nav-link">

Dashboard

</a>

</li>

</ng-container>

Start the dev server with the following command.

ng serve -o

It will open up the browser with **http://localhost:4200** URL.

Click on the **home**link, and you will see the following URL.

**http://localhost:4200/home?success=true**

That is it. We have successfully passed the query parameters to the HomeComponent.

We can capture the parameter and put some conditions based on that parameter values. For example, display the notification like a bootstrap alert box on success or failure.

Write the following code inside the **home.component.ts**file.

*// home.component.ts*

**import** { Component, OnInit } **from** '@angular/core';

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

**@Component**({

selector: 'app-home',

templateUrl: './home.component.html',

styleUrls: ['./home.component.css']

})

**export** **class** HomeComponent **implements** OnInit {

notify: string;

**constructor**(**private** router: Router, **private** route: ActivatedRoute) { }

ngOnInit(): void {

**this**.route.queryParams.subscribe(params => {

**if** (params.success === 'true') {

**this**.notify = 'You have passed data';

}

});

}

}

In this code, we have used the **queryParams.subscribe()**function to get the query parameters passed to the current URL.

Based on the condition, we are setting the **notify**message.

Now, write the following code inside the **home.component.html**file.

*<!-- home.component.html -->*

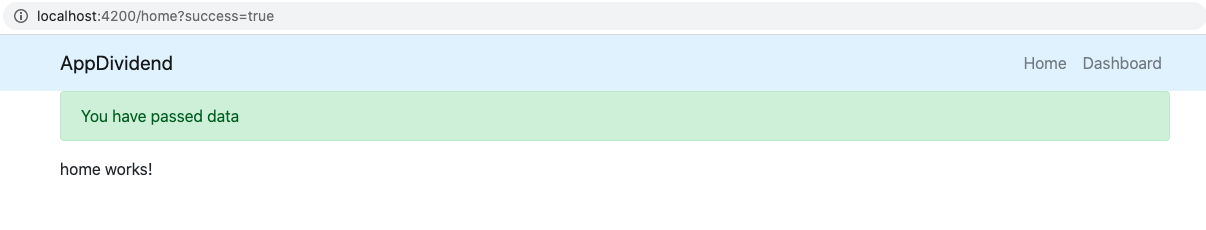
<div \*ngIf="notify" class="alert alert-success">

{{ notify }}

</div>

<p>home works!</p>

Okay, now see the output in the browser.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-RouterLink-Example.png)

If you remove the query parameters from the URL and just hit this URL: http://localhost:4200/home, you will not get any notification message.

**Conclusion**

In this example, we have seen how to use routerLink directive in navigation, pass query parameters and capture the queryParams using ActivatedRoute.

Angular Route Guard: Implement Route Guard In Angular 12

Angular route guard allows us to grant or remove access to certain parts of the navigation. Another route guard, the CanDeactivate guard, enables you to prevent a user from accidentally leaving a component with unsaved changes.

**Why do we need Angular guards?**

To prevent unauthorized access to certain parts of our navigation, use route guards in Angular.

The client-side route guards like this are not meant to be a security feature. However, they won’t prevent a smart user from figuring out how to get to the protected routes.

Such security should be implemented on the server-side. So you need to develop the logic for the server-side, and based on the response, we will change the routes.

Route guards are instead meant as a way to improve the UX for your apps.

**Types of routing guards**

Route guards in Angular can prevent users from navigating to parts of an app without authorization.

There are 4 route guards available in Angular.

1. **CanActivate**: It controls if a route can be activated.
2. **CanActivateChild**: It controls if children of a route can be activated.
3. **CanLoad**: It controls if a route can even be loaded. This becomes useful for lazy-loaded feature modules. They won’t also load if the guard returns false.
4. **CanDeactivate**: It controls if the user can leave a route. **Note that** this guard doesn’t prevent the user from closing the browser tab or navigating to a different address. It only prevents actions from within the application itself.

To use route guards, consider using component-less routes as this facilitates guarding child routes.

**How to Create Guard Service in Angular**

To create a service for your guard, type the following command.

ng **generate** guard your-guard-name

In your guard class, implement the guard you want to use. The following example uses CanActivate to guard the route.

export **class** **YourGuard** **implements** **CanActivate** {

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): **boolean** {

*// your logic goes here*

}

}

**Angular Route Guard Example**

In traditional server-side web applications, the application would check the permissions on the server and return the 403 error page if the user didn’t have permissions, or perhaps redirect them to a /auth/login/ page if they were not signed in.

We want to have the same functionality on our client-side SPA, and with **Router Guards,** we can implement that kind of functionality.

With **Router Guards,** we can prevent the users from accessing areas that they’re not permitted to access, or we can ask them for confirmation when leaving the particular area. Let’s perform a practical and see how we can set up a guard for our angular application.

**Step 1: Install the Angular 12 project.**

To create a new Angular 12 project, type the following command.

ng **new** angularguard

While creating a new project, please enable the routing so that the **app-routing.module.ts**file is created.

Now, go inside the project and create the following two components.

1. HomeComponent
2. DashboardComponent

Type the following commands to create components.

ng g c home --skipTests=true

ng g c dashboard --skipTests=true

From these components, we will prevent access to the dashboard component if the user is not logged in. Otherwise, the user can access the dashboard component.

That means we will set the auth guard to the dashboard component. So if the auth service returns true, then the user is authenticated; otherwise, it is not. Based on true or false, we will prevent access to the component.

**Step 2: Setup routing**

Now, let’s set up the routing for these two components.

Write the following code inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

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

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

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

**import** { DashboardComponent } from './dashboard/dashboard.component';

const routes: Routes = [

{ path: '', component: AppComponent },

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

{ path: 'dashboard', component: DashboardComponent }

];

**@NgModule**({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export **class** **AppRoutingModule** { }

We have defined three routes.

1. **‘/’** route for AppComponent.
2. **‘home’** route for HomeComponent
3. **‘dashboard’** route for DashboardComponent

Now, edit the **app.component.html** file and write the following code.

*<!-- app.component.html -->*

<p>Angular Auth Guard Example</p>

<router-outlet></router-outlet>

So, now you have three navigation parts in your angular 12 application.

1. **http://localhost:4200/**
2. **http://localhost:4200/home**
3. **http://localhost:4200/dashboard**

Right now, every part is accessible.

**Step 3: Create an auth service**

The auth service is responsible for returning a boolean value. If it will return **true,** then the user is logged in; otherwise, it is not logged in and returns **false**.

To create a service in Angular, type the following command.

ng g s auth --skipTests=true

The next step is to register the auth service inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

**import** { AuthService } **from** './auth.service';

**@NgModule**({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

providers: [AuthService]

})

Now, write the following function inside the **auth.service.ts** file.

*// auth.service.ts*

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

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthService {

**constructor**() { }

isLoggedIn(): boolean {

**return** false;

}

}

Right now, we are returning false. That means the user is not authenticated.

For proper authentication in Angular, please check out the [Angular JWT Authentication](https://appdividend.com/2020/07/09/angular-authentication-system-login-and-registration-in-angular/) Tutorial.

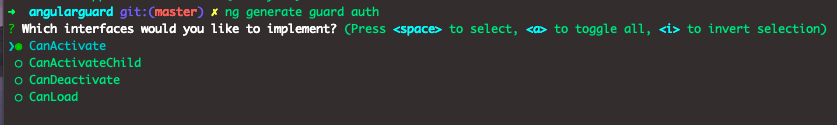
Okay, let’s create an auth guard.

**Step 4: Create an Angular route guard as a service**

To create a service for your guard, type the following command.

ng **generate** guard auth --skipTests=true

You will get to choose which type of guard you want to create, as in the following image.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-Route-Guard-Example.png)

I am choosing the CanActivate guard.

Your **auth.guard.ts**file will be created and looks like this.

*// auth.guard.ts*

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

**import** { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } **from** '@angular/router';

**import** { Observable } **from** 'rxjs';

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthGuard **implements** CanActivate {

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

**return** true;

}

}

Interface that a class can implement to be a guard deciding if the route can be activated. If all the guards return true, navigation will continue. If any guard returns false, the navigation will be canceled.

If any guard returns the UrlTree, current navigation will be canceled, and new navigation will be kicked off to the UrlTree returned from a guard.

Now, import the **auth.service.ts**file inside this guard.

After that, we will complete the **canActivate()**function.

Based on if the user is authenticated or not, the canActivate() function will return **true**or**false.**

Write the following code inside the **auth.guard.ts**file.

*// auth.guard.ts*

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

**import** { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } **from** '@angular/router';

**import** { Observable } **from** 'rxjs';

**import** { AuthService } **from** './auth.service';

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthGuard **implements** CanActivate {

**constructor**(**private** auth: AuthService) {}

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

**if** (**this**.auth.isLoggedIn()) {

**return** true;

}

window.alert('You don\'t have permission to view this page');

**return** false;

}

}

Here, if the user is logged in, it won’t prevent accessing the page; otherwise, we will show the user alert that you don’t have permission to view this page.

You can see that we have injected the auth service in the constructor to use its function.

If the user is logged in, the guard passes and lets the user through.

If the user is not logged in, the guard fails, we show the user an alert, and the page doesn’t navigate to the intended URL.

**Step 5: Attach the Auth Guard in the routing module.**

In your routing module, use the appropriate property in your **routes** configuration.

Add the following code inside the **app-routing.module.ts**file.

*// app-routing.module.ts*

**import** { AuthGuard } from './auth.guard';

const routes: Routes = [

{ path: '', component: HomeComponent },

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

{ path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }

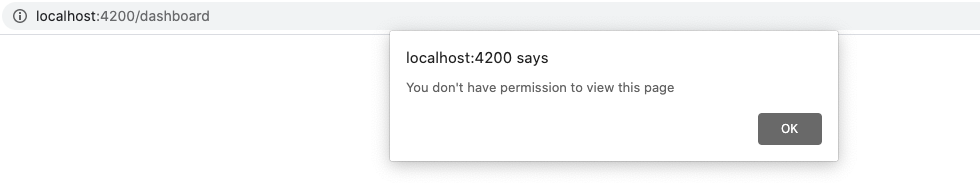
];

Here, **canActivate** tells the router to mediate navigation to this particular route.

Okay, now, let’s try to access the **dashboard route.**

Go to the browser and type this URL: **http://localhost:4200/dashboard.**

You will see something like this.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-10-Guards.png)

That is it. We have successfully prevented access using the Auth guard.

Now, let’s return **true**from **auth.service.ts** file’s **isLoggedIn()**function.

*// auth.service.ts*

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

**@Injectable**({

providedIn: 'root'

})

**export** **class** AuthService {

**constructor**() { }

isLoggedIn(): boolean {

**return** true;

}

}

Now, we should be able to access the dashboard component or route, or page.

Let’s try one more time. Go to the **http://localhost:4200/dashboard.**

Bingo!! You now have access to the page. See the following output.

[](https://appdividend.com/wp-content/uploads/2020/07/Auth-guard-example.png)

I have also put the code on Github. So, please check out that as well.

[GITHUB CODE](https://github.com/KrunalLathiya/AngularRouteGuard)

For more information on route guards, check out official [Angular documentation](https://angular.io/guide/router).

**Conclusion**

In this Angular Route guards tutorial, we have seen the following things.

1. How to create a guard using a default command.
2. How to inject Angular Service into Guard Service.
3. How to use canActivate guard to prevent access to a specific page.

Lab

n this Angular 13 router tutorial, we will learn how to enable routing & navigation service in an Angular app.

Routing allows users to navigate between one component to another component based on action taken by the user.

Routing is a mechanism in modern web or mobile applications, be it single-page applications, progressive web applications, or mobile apps.

A user can fetch content in the view based on the component a user wants to navigate. It can also be directly done by using the route URL in the browsers.

**Simple Angular 13 Router Example**

We will understand how to configure routing & navigation in Angular and will get started by installing a simple Angular app with some components, then we will find the answers for following questions:

* How to configure routing & navigation in Angular 13 with built-in APIs?
* How to import and configure the routing module in the main AppModule?
* How to implement an active class with a routerLink directive?
* How to enable routes for specific components?
* How to use router-outlet directive to enable views based on related component?
* How to configure wild card routes in Angular?
* How to Send/Get Parameters?

**Table of Contents**

1. [Install Angular Project](https://www.positronx.io/angular-router-tutorial/#tc_7403_01)
2. [Configure Routes & RouterModule](https://www.positronx.io/angular-router-tutorial/#tc_7403_02)
3. [Angular Route Matching Strategies](https://www.positronx.io/angular-router-tutorial/#tc_7403_03)
4. [Wildcard Route in Angular](https://www.positronx.io/angular-router-tutorial/#tc_7403_04)
5. [Angular Route Parameters](https://www.positronx.io/angular-router-tutorial/#tc_7403_05)
6. [Angular Router Outlet & Navigation Directives](https://www.positronx.io/angular-router-tutorial/#tc_7403_06)
7. [Conclusion](https://www.positronx.io/angular-router-tutorial/#tc_7403_07)

**Install Angular Project**

Run command to install Angular project for Angular Router demo.

ng new angular-routing-tutorial

Bash

Copy

cd angular-routing-tutorial

Bash

Copy

Install Bootstrap, we will use Bootstrap 4 UI components for the demo purpose.

npm install bootstrap

TypeScript

Copy

Place **bootstrap.min.css** file path in the styles array in package.json:

"styles": [

"node\_modules/bootstrap/dist/css/bootstrap.min.css",

"src/styles.css"

]

JSON

Copy

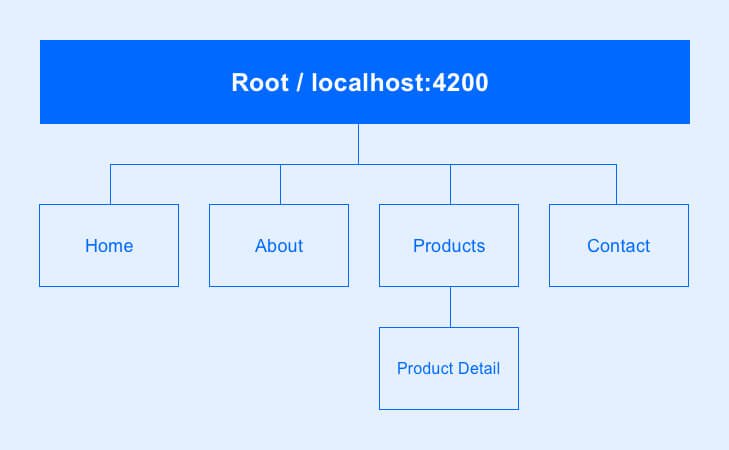
ng serve --open

Bash

Copy

**Configure Routes & RouterModule**

Before we configure routes in Angular, we have to have a look on the app’s navigation architecture and install the components based on the diagram.

  
As per the app, architecture create the components by using the Angular CLI command in the terminal.

ng generate component home

ng generate component about

ng generate component contact

ng generate component products

ng generate component product-detail

ng generate component NoPageFound

Bash

Copy

Create app-routing.module.ts file and copy the components from the **app.module.ts** file and define in app routing class.

In AppRoutingModule Routes and RouterModule service are responsible for enabling routing service in Angular.

Here, define the path of the route using the Angular components. Here in the path: "..." insert your route name.

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 { ProductsComponent } from './products/products.component';

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

import { NoPageFoundComponent } from './no-page-found/no-page-found.component';

const routes: Routes = [

{ path: '', redirectTo: '/home', pathMatch: 'full' },

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

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

{ path: 'products', component: ProductsComponent },

{ path: 'product-detail/:id', component: ProductDetailComponent },

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

{ path: '\*\*', component: NoPageFoundComponent },

];

@NgModule({

declarations: [

HomeComponent,

AboutComponent,

ContactComponent,

ProductsComponent,

ProductDetailComponent,

NoPageFoundComponent,

],

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

})

export class AppRoutingModule {}

TypeScript

Copy

This will be the final **app.module.ts** file, if you want you can copy and paste the given code within the file.

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

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

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

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

@NgModule({

declarations: [AppComponent],

imports: [BrowserModule, AppRoutingModule],

providers: [],

bootstrap: [AppComponent],

})

export class AppModule {}

TypeScript

Copy

Here are the routes we just created in Angular.

|  |  |
| --- | --- |
| **URL** | **Component** |
| <http://localhost:4200/home> | HomeComponent |
| <http://localhost:4200/about> | AboutComponent |
| <http://localhost:4200/products> | ProductsComponent |
| [http://localhost:4200/product-detail/:id](http://localhost:4200/product-detail) | ProductDetailComponent |
| <http://localhost:4200/contact> | ContactComponent |

**Angular Route Matching Strategies**

Angular offers **prefix** and **full** route matching strategies. These are built-in route matching mechanism to identify if the existing browser’s URL is **prefixed** with the **path**.

pathMatch: 'full' signifies that the complete URL path requires to be matched and is utilized by the route matching mechanism.

In the below example, we are redirecting to the /home component and matching the empty path using the **pathMatch: ‘full’** property.

{ path: '', redirectTo: '/home', pathMatch: 'full' }

TypeScript

Copy

The redirectTo property helps you redirect to the specified path if the particular route is matched.

pathMatch: 'prefix' indicates, If the route matching strategy of a particular route is set to prefix, In this case, the router identifies if the starting URL in the browser is prefixed with the route’s path, and it gets the associated component.

Here we are checking if the current URL in the browser prefixed with the path.

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

TypeScript

Copy

**Wildcard Route in Angular**

If the user lands on the page, which doesn’t exist in the app, to get rid of this problem, we can redirect the user to the 404 page. Fortunately, to sort out this issue, we have a wildcard route in Angular. It can be done quickly bypassing double asterisk, and it can be written something like this path: "\*\*" here is how you set wildcard route in routes array, check out the example below.

const routes: Routes = [

...

...

{ path: '\*\*', component: NoPageFoundComponent }

];

TypeScript

Copy

Setting up wildcard route in Angular is simple, you have to create a component first and let’s call it **NoPageFoundComponent** or whatever you and then import in the app-routing.module.ts file:

Go to no-page-found.component.html and add the following HTML code in it and then enter some random wrong name in the browser’s address bar.

<div class="col-md-12">

<div class="error-template">

<h3>Oops!</h3>

<h2>404 Not Found</h2>

<div class="error-details mb-3">

Sorry, Requested page not found!

</div>

<div class="error-actions">

<a routerLink="/home" class="btn btn-danger btn-lg">

Go Back to Home

</a>

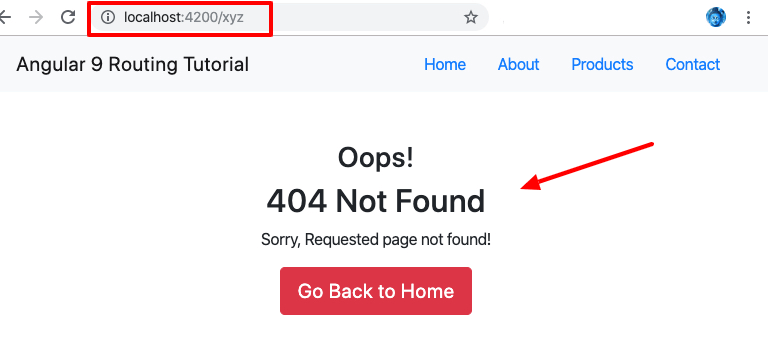
</div>

</div>

</div>

Markup

Copy



**Angular 13 Route Parameters**

Next, we will learn how to add a router for **products** and **product-details** components with the help of Angular route parameters or params. Here we will create a product-details route with id parameter. There are many ways to locate parameters in Angular Router.

Here is the quick look of the code we added in the app-routing.module.ts.

import { ProductsComponent } from './products/products.component';

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

const routes: Routes = [

{ path: 'products', component: ProductsComponent },

{ path: 'product-detail/:id', component: ProductDetailComponent }

];

TypeScript

Copy

Next, go to products.component.ts file and add the products array.

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-products',

templateUrl: './products.component.html',

styleUrls: ['./products.component.css']

})

export class ProductsComponent implements OnInit {

Products = [

{

id: "ABC8441189035",

name: "Tshirt",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit."

},

{

id: "DEF6510463347",

name: "Shoes",

description: "Proin ac metus in diam porttitor viverra eu sit amet ligula."

},

{

id: "GHI0831819467",

name: "Handbags",

description: "Duis sodales dui vitae urna varius, at ullamcorper purus tempor."

}

]

constructor() { }

ngOnInit() { }

}

TypeScript

Copy

Now, let’s understand what does the below code mean. We are passing the id parameter in the [routerLink]="..." route and sending the data to the params. When the user clicks on the product then the user will be redirected to associated id product in the product-details page.

<a [routerLink]="['/product-detail/', products.id]" class="btn btn-danger">Check Product</a>

TypeScript

Copy

Next, go to products.component.html file and add the following code.

<div class="row">

<div class="col-sm-4" \*ngFor="let products of Products">

<div class="card mb-3">

<div class="card-body">

<h5 class="card-title">{{products.name}}</h5>

<p class="card-text">{{products.description}}</p>

<a [routerLink]="['/product-detail/', products.id]" class="btn btn-danger">Check Product</a>

</div>

</div>

</div>

</div>

TypeScript

Copy

**GET Angular Route Parameter with ActivatedRoute**

We learned in the previous how to send parameters in the Angular router and create dynamic routes. Now, we will use the Activated route API to get the route parameter in an Angular app.

Go to product-detail.component.ts and add the below code in it.

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

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

@Component({

selector: 'app-product-detail',

templateUrl: './product-detail.component.html',

styleUrls: ['./product-detail.component.css'],

})

export class ProductDetailComponent {

product\_id: string;

constructor(private actRoute: ActivatedRoute) {

this.product\_id = this.actRoute.snapshot.params['id'];

}

}

Here we imported the **ActivatedRoute** service and set the **product\_id** variable. Then injected the **ActivatedRoute** service in the constructor and used the snapshot.params.id with the help of ActivatedRoute and assign the id parameter to the **product\_id** variable.

Next, go to product-detail.component.html file and add the below code, this will show the associated product id when the user visit the product-details page.

<h5 class="card-title">Product Id is: {{product\_id}}</h5>

TypeScript

Copy

**Angular Router Outlet & Navigation Directives**

The Router-Outlet directive allows components to display data dynamically based on the existing browser’s URL or path. It can be added inside the **app.component.ts** file, and multiple router outlets can be placed in an Angular app for fulfilling the advanced routing cases. We are using the Bootstrap container class and wrapping the router-outlet in it.

<div class="container text-center">

<router-outlet></router-outlet>

</div>

TypeScript

Copy

**Configure Routes with Navigation Directives**

To configure the routers in Angular and active class, we have

**routerLink and routerLinkActive directives,**

check out below how we used both the navigation directives in

**app.component.ts** file.

<nav class="navbar navbar-expand-lg navbar-light bg-light mb-5">

<div class="container">

<a class="navbar-brand" href="#">Angular Routing Tutorial</a>

<ul class="nav nav-pills">

<li class="nav-item">

<a class="nav-link" routerLinkActive="active" routerLink="/home">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLinkActive="active" routerLink="/about">About</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLinkActive="active" routerLink="/products">Products</a>

</li>

<li class="nav-item">

<a class="nav-link" routerLinkActive="active" routerLink="/contact">Contact</a>

</li>

</ul>

</div>

</nav>

<div class="container text-center">

<router-outlet></router-outlet>

</div>

n this chapter

, we learn

* **how to pass parameters (route parameter) or data to the Route in Angular**.

First, let us look at how to define the route, which accepts the parameter.  We then

learn how to pass the parameters to the route using the [routerLink](https://www.tektutorialshub.com/angular/routerlinkactive-router-outlet-styling/)directive. Finally, we learn how to retrieve the parameters using the ActivatedRoute Service.

 The parameters can be retrieved by either using snapshot method or by subscribe method. We take a look at both of these methods.

There are many scenarios, where you need to pass parameters to the route. For example, to navigate to the product detail view, we need to pass the product ID, so that component can retrieve it and display it to the user.

## What are Route Parameters

The Route parameters are a dynamic part of the Route and essential in determining the route.

|  |  |
| --- | --- |
| For example, consider the following route | { path: 'product', component: ProductComponent } |

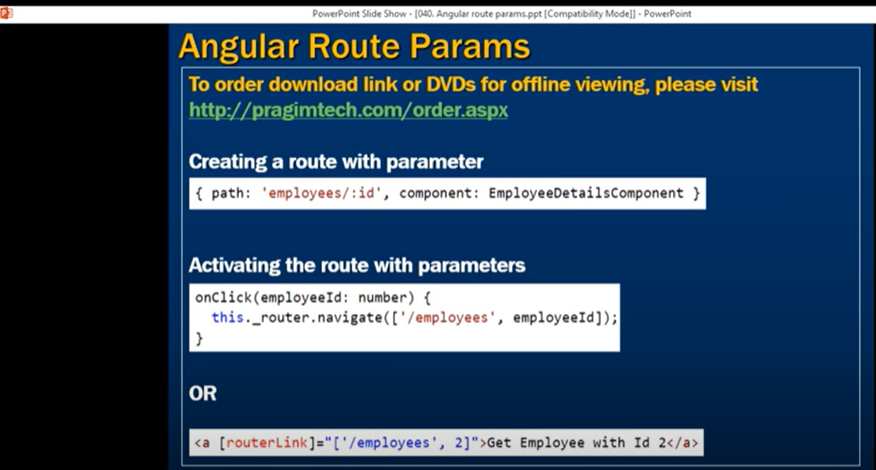
The above route match only if the URL is /product

To retrieve the product details, our URL should look something like

/product/1  
/product/2

Where the second URL segment ( 1 and 2 ) being the id of the product. The **id** is dynamic and changes as per the selected Product.

To handle such scenario[angular router](https://www.tektutorialshub.com/angular/angular-routing-navigation/) allows us to include **route parameters**, where we can send any dynamic value for a URL segment



## How to Pass parameters to Angular Route

### Defining the Route

We can define parameter by adding forward slash followed colon and a placeholder (id) as shown below

|  |  |
| --- | --- |
|  | **App-routing .module.ts**  **{ path: 'product/:id', component: ProductDetailComponent }** |

Now above path matches the URLs /product/1 , /product/2, etc.

If you have more than one parameter, then you can extend it by adding one more forward slash followed colon and a placeholder

|  |  |
| --- | --- |
|  | { path: 'product/:id/:id1/:id2', component: ProductDetailComponent } |

The name id, id1 & id2 are placeholders for parameters. We will use them while retrieving the values of the parameters.

### Defining the Navigation

We, now need to provide both **path** and the**route parameter** routerLink directive.

This is done by adding the productID as the second element to the routerLink parameters array as shown below

|  |  |
| --- | --- |
|  | **App.component.html**  **<a [routerLink]="['/Product', ‘2’]">{{product.name}} </a>** |

Which translates to the UR**L /product/2**

OR

|  |  |
| --- | --- |
|  | <a [routerLink]="['/Product', product.productID]">{{product.name}} </a> |

Which, dynamically takes the value of id from the product object.

### Retrieve the parameter in the component

Finally, our component needs to extract the route parameter from the URL

This is done via the **ActivatedRoute** service from angular/router module to get the parameter value

## ActviatedRoute

The [ActivatedRoute](https://angular.io/api/router/ActivatedRoute) is a service, which keeps track of the currently activated route associated with the loaded Component.

To use ActivatedRoute, we need to import it in our component

|  |  |
| --- | --- |
| **import { ActivatedRoute } from '@angular/router';** | |
| Then inject it into the component using dependency injection | constructor(private \_Activatedroute:ActivatedRoute) | |

### ParamMap

The Angular adds the map all the route parameters in the ParamMap object, which can be accessed from the ActivatedRoute service

The [ParamMap](https://angular.io/api/router/ParamMap) makes it easier to work with parameters.

We can use get or getAll methods to retrieve the value of the parameters in the component

 map that provides access to the required and optional parameters specific to a route. The map supports retrieving a single value with get() or multiple values with getAll().

interface [ParamMap](https://angular.io/api/router/ParamMap) { [**keys**: string[]](https://angular.io/api/router/ParamMap#keys)

[**has**(name: string): boolean](https://angular.io/api/router/ParamMap#has)

[get(name: string): string | null](https://angular.io/api/router/ParamMap#get)

[**getAll**(name: string): string[]](https://angular.io/api/router/ParamMap#getAll) }

. Use the has method to check if a certain parameter exists.

The Older version of ActivatedRoute class has a Params array which is an array of the parameter values, indexed by name. You can still use it but It is now deprecated

and is replaced by the ParamMap.

There are two ways in which you can use the ActivatedRoute to get the parameter value from the ParamMap object.

1. Using Snapshot
2. Using observable
3. **paramMap Observable**

### Using Snapshot

|  |  |
| --- | --- |
|  | this.id=this.\_Activatedroute.snapshot.paramMap.get("id"); |

The snapshot property returns the initial value of the route. You can then access the paramsMap array, to access the value of the id, as shown above,

Use the Snapshot option, if you only need the initial value.

### Using Observable

|  |  |
| --- | --- |
|  | this.\_Activatedroute.paramMap.subscribe(params => {      this.id = params.get('id');  });   1. paramMap is the method of ActivatedRoute and it will return an observable and it will give data only if we subscribe to it. The argument in the subscribe method is an arrow function. The observable provides the parameter which we strongly typed to paramMap. paramMap is coming from Router package. We should have imported the Router package. 2. The paramMap provides us with the get method to retrieve the parameters. |

You can retrieve the value of id by subscribing to the paramMap observable property of the activateRoute as shown above

Use this option if you expect the value of the parameter to change over time.

### Why use observable

We usually retrieve the value of the parameter in the [**ngOninit**](https://www.tektutorialshub.com/angular/angular-ngoninit-and-ngondestroy/) life cycle hook, when the component initialised.

When the user navigates to the component again, the Angular does not create the new component but reuses the existing instance. In such circumstances, the ngOnInit method of the component is not called again.

Hence you need a way to get the value of the parameter.

By subscribing to the observable paramMap property,

you will retrieve the latest value of the parameter and update the component accordingly.

The above difference is explained in our next chapter [Angular child routes tutorial](https://www.tektutorialshub.com/angular/angular-child-routes-nested-routes/).

## Example-Passing Parameters to Route: Example

**product.component.ts**

|  |  |
| --- | --- |
|  | **import { Component, OnInit } from '@angular/core';**    **import { ProductService } from './product.service';**  **import { Product } from './product';**    **@Component({**  **templateUrl: './product.component.html',**  **})**    **export class ProductComponent**  **{**    **products:Product[];**    **constructor(private productService:ProductService){**  **}**    **ngOnInit() {**  **this.products=this.productService.getProducts();**  **}**    **}** |

**product.component.html**

|  |  |
| --- | --- |
|  | <h1>Product List</h1>  <div class='table-responsive'>      <table class='table'>          <thead>              <tr>                  <th>ID</th>                  <th>Name</th>                  <th>Price</th>              </tr>          </thead>          <tbody>              <tr \*ngFor="let product of products;">                  <td>{{product.productID}}</td>                  <td><a [routerLink]="['/product',product.productID]">{{product.name}} </a> </td>                  <td>{{product.price}}</td>              </tr>          </tbody>      </table>  </div> |

**product.service.ts**

|  |
| --- |
| import { Observable } from 'rxjs';  import {Product} from './Product'      export class ProductService{        public getProducts() {            let products:Product[];            products=[              new Product(1,'Memory Card',500),              new Product(2,'Pen Drive',750),              new Product(3,'Power Bank',100)          ]            return products;      }          public getProduct(id) {          let products:Product[]=this.getProducts();          return products.find(p => p.productID==id);      }      } |

**product.ts**

|  |  |
| --- | --- |
|  | export class Product {        constructor(productID:number,    name: string ,   price:number) {          this.productID=productID;          this.name=name;          this.price=price;      }        productID:number ;      name: string ;      price:number;    } |

In the product.component.html, we have added product.productID as the second argument to the routerLink parameters array.

|  |  |
| --- | --- |
|  | <a [routerLink]="['/product',product.productID]">{{product.name}} </a> |

#### Product Details Component

**product-detail.component.ts**

|  |  |
| --- | --- |
|  | import { Component, OnInit, OnDestroy } from '@angular/core';  import { Router,ActivatedRoute } from '@angular/router';    import { ProductService } from './product.service';  import { Product } from './product';      @Component({    templateUrl: './product-detail.component.html',  })    export class ProductDetailComponent  {       product:Product;     id;       constructor(private \_Activatedroute:ActivatedRoute,                 private \_router:Router,                 private \_productService:ProductService){     }         /\* Using snapshot \*/     // ngOnInit() {       //    //This still works but is deprecated     //    //this.id=this.\_Activatedroute.snapshot.params['id'];       //    this.id=this.\_Activatedroute.snapshot.paramMap.get("id");         //    let products=this.\_productService.getProducts();     //    this.product=products.find(p => p.productID==this.id);     // }         /\* Using Subscribe \*/       sub;       ngOnInit() {          this.sub=this.\_Activatedroute.paramMap.subscribe(params => {           console.log(params);            this.id = params.get('id');            let products=this.\_productService.getProducts();            this.product=products.find(p => p.productID==this.id);        });          // This params is deprecated          //this.sub=this.\_Activatedroute.params.subscribe(params => {        //    this.id = params['id'];        //    let products=this.\_productService.getProducts();        //    this.product=products.find(p => p.productID==this.id);        //        //});     }       ngOnDestroy() {       this.sub.unsubscribe();     }       onBack(): void {        this.\_router.navigate(['product']);     }  } |

**product-detail.component.html**

|  |
| --- |
| <h3>Product Details Page</h3>      product : {{product.name}}  price : {{ product.price}}  <p>      <a class='btn btn-default' (click)="onBack()">Back </a>  </p> |

In the ProductDetailComponent, we have imported router and ActivatedRoute from the angular router module

|  |  |
| --- | --- |
| 1 | import { Component, OnInit, OnDestroy } from '@angular/core';  import { Router,ActivatedRoute } from '@angular/router'; |

In the constructor, we inject the ActivatedRoute, Router service along with ProductService

|  |  |
| --- | --- |
| 1 | constructor(private \_Activatedroute:ActivatedRoute,                 private \_router:Router,                 private \_productService:ProductService){     } |

Finally, we use ngOninit life cycle hook to retrieve the value of the id parameter and use that value to retrieve the details of the product.

Note that, there are two ways, by which you can retrieve the data.

**Using snapshot**

|  |  |
| --- | --- |
|  | ngOnInit() {         //This still works but is deprecated       //this.id=this.\_Activatedroute.snapshot.params['id'];         this.id=this.\_Activatedroute.snapshot.paramMap.get("id");           let products=this.\_productService.getProducts();       this.product=products.find(p => p.productID==this.id);  } |

**Using Subscribe**

We used snapshot method to retrieve the parameter in the ProductDetailcomponet.ts. To Subscribe to params remove the ngOnInit and replace it with the following code

We recommend you to use the subscribe method as it offers the benefit of responding to the parameter changes dynamically.

|  |  |
| --- | --- |
|  | ngOnInit() {          this.sub=this.\_Activatedroute.paramMap.subscribe(params => {           console.log(params);            this.id = params.get('id');            let products=this.\_productService.getProducts();            this.product=products.find(p => p.productID==this.id);        });          // This params is deprecated          //this.sub=this.\_Activatedroute.params.subscribe(params => {        //    this.id = params['id'];        //    let products=this.\_productService.getProducts();        //    this.product=products.find(p => p.productID==this.id);        //        //});     } |

#### The Routes

**app.routing.ts**

|  |  |
| --- | --- |
|  | import { Routes } from '@angular/router';    import { HomeComponent} from './home.component'  import { ContactComponent} from './contact.component'  import { ProductComponent} from './product.component'  import { ErrorComponent} from './error.component'    import { ProductDetailComponent} from './product-detail.component'    export const appRoutes: Routes = [    { path: 'home', component: HomeComponent },    { path: 'contact', component: ContactComponent },    { path: 'product', component: ProductComponent },    { path: 'product/:id', component: ProductDetailComponent },    { path: '', redirectTo: 'home', pathMatch: 'full' },    { path: '\*\*', component: ErrorComponent }  ]; |

We have added the following route to our routes array

|  |  |
| --- | --- |
|  | { path: 'product/:id', component: ProductDetailComponent }, |

#### Other components

**app.component.ts**

|  |  |
| --- | --- |
|  | import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'Routing Module - Parameters Demo';  } |

**app.component.html**

|  |  |
| --- | --- |
|  | <div class="container">    <nav class="navbar navbar-default">    <div class="container-fluid">      <div class="navbar-header">        <a class="navbar-brand" [routerLink]="['/']"><strong> {{title}} </strong></a>      </div>      <ul class="nav navbar-nav">          <li><a [routerLink]="['home']">Home</a></li>          <li><a [routerLink]="['product']">Product</a></li>          <li><a [routerLink]="['contact']">Contact us</a></li>      </ul>    </div>  </nav>    <router-outlet></router-outlet>    </div> |

**contact.component.ts**

|  |
| --- |
| import {Component} from '@angular/core';    @Component({       template: `<h1>Contact Us</h1>                  <p>TekTutorialsHub </p>                  `  })  export class ContactComponent {  } |

**home.component.ts**

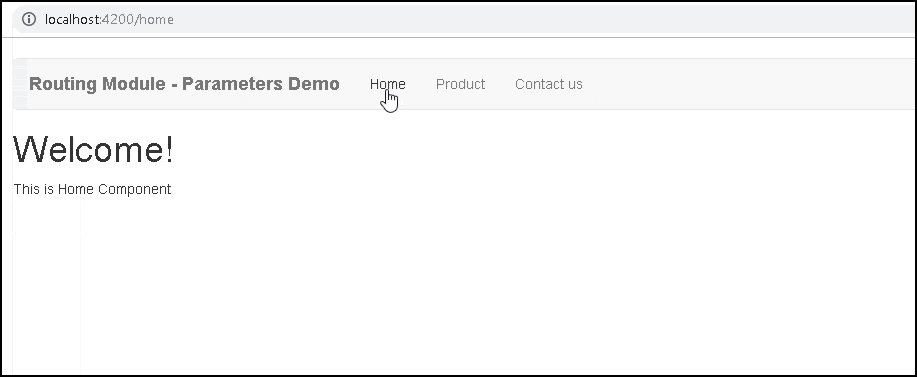
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import {Component} from '@angular/core';    @Component({      template: `<h1>Welcome!</h1>                <p>This is Home Component </p>               `  })    export class HomeComponent {  } |

**error.component.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import {Component} from '@angular/core';    @Component({      template: `<h1>Page not found</h1>                 <p>This is a Error Page</p>                `  })    export class ErrorComponent {  } |

**app.module.ts**

|  |
| --- |
| import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';  import { HttpModule } from '@angular/http';    import { RouterModule } from '@angular/router';    import { AppComponent } from './app.component';  import { HomeComponent} from './home.component'  import { ContactComponent} from './contact.component'  import { ProductComponent} from './product.component'  import { ErrorComponent} from './error.component'  import { ProductDetailComponent} from './product-detail.component'    import { ProductService } from './product.service';    import { appRoutes } from './app.routes';    @NgModule({    declarations: [      AppComponent,HomeComponent,ContactComponent,ProductComponent,ErrorComponent,ProductDetailComponent    ],    imports: [      BrowserModule,      FormsModule,      HttpModule,      RouterModule.forRoot(appRoutes)    ],    providers: [ProductService],    bootstrap: [AppComponent]  })  export class AppModule { } |



### ActivatedRoute

The ActivatedRoute service has a great deal of useful information including:

**url:** This property returns an array of Url Segment objects, each of which describes a single segment in the URL that matched the current route.

**params:** This property returns a Params object, which describes the URL parameters, indexed by name.

**queryParams:** This property returns a Params object, which describes the URL query parameters, indexed by name.

**fragment:** This property returns a string containing the URL fragment.

**Snapshot:** The initial snapshot of this route

**data:** An Observable that contains the data object provided for the route

**Component:** The component of the route. It’s a constant

**outlet:** The name of the RouterOutlet used to render the route. For an unnamed outlet, the outlet name is primary.

**routeConfig:** The route configuration used for the route that contains the origin path.

**parent:** an ActivatedRoute that contains the information from the parent route when using child routes.

**firstChild:** contains the first ActivatedRoute in the list of child routes.

**children:** contains all the child routes activated under the current route

**pathFromRoot:** The path from the root of the router state tree to this route

### Conclusion

We looked at how to pass parameters or data to the Route. The parameters are passed to the route by using routerLink parameters in the routerLink directive. We retrieve the parameters from ActivatedRoute Service by reading params collection of the snapshot object or by subscribing to the params observable

**Angular -13**

**Passing and Getting Route Parameters Example**

* Create Angular App
* Set Route Parameter with Slash in Angular
* Check If Query Param Exists in Angular Route
* Angular URL Parameter with Question Mark Example
* Angular Route Parameter with Hash Separator Example

**Create Angular App**

Start the first step installing the new angular application ignore if app already installed:

ng new ng-demo

Bash

COPY

**Disable Strict Angular TypeStrict Errors**

The latest version of Angular comes with strict mode, you have to manually disable the strict mode you can set “strict”: false, "noImplicitReturns": false and "strictTemplates": false inside the compilerOptions and angularCompilerOptions in **tsconfig.json** file.

Use command to generate new components:

ng generate component blog

Bash

COPY

ng generate component post-detail

Bash

COPY

**Set Route Parameter with Slash in Angular**

Setting parameters in the Route or URL is possible with slashes; you will see how to pass a parameter with slash separator inside the path property within the routes const:

http://www.positronx.io/post/5555/javascript

.properties

COPY

Here is the route structure in response to passing the parameters such as id and category in angular url:

Open the **app-routing.module.ts** file and update the below code:

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

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

import { BlogComponent } from './blog/blog.component';

import { PostDetailComponent } from './post-detail/post-detail.component';

const routes: Routes = [

{

path: '',

component: BlogComponent

},

{

path: 'post/:postid/:category',

component: PostDetailComponent

}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

**Access Route Parameter**

Import ActivatedRoute from the ‘@angular/router’ module and inject it inside the constructor.

You will access the post id and category using the snapshot method and subscribe to paramMap.

Through this template you will learn how to get the parameter from the route.

Open and update below code in **post-detail.component.ts** file.

import { Component, OnInit } from '@angular/core';

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

@Component({

selector: 'app-post-detail',

templateUrl: './post-detail.component.html',

styleUrls: ['./post-detail.component.css']

})

export class PostDetailComponent implements OnInit {

constructor(private actRoute: ActivatedRoute) { }

ngOnInit(): void {

let postid = this.actRoute.snapshot.params.postid;

console.log(postid);

let category = this.actRoute.snapshot.params.category;

console.log(category);

this.actRoute.paramMap.subscribe(res => {

postid = res.get('postid');

console.log(postid);

category = res.get('category');

console.log(category);

});

}

}

TypeScript

COPY

**Check If Query Param Exists in Angular Route**

You can verify if the existing parameter for URL exists:

import { Component, OnInit } from '@angular/core';

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

@Component({

selector: 'app-post-detail',

templateUrl: './post-detail.component.html',

styleUrls: ['./post-detail.component.css']

})

export class PostDetailComponent implements OnInit {

constructor(private actRoute: ActivatedRoute) { }

ngOnInit(): void {

let postid = this.actRoute.snapshot.params.postid;

console.log(postid);

if(postid === null){

console.log('Post id param does not exist');

}

let category = this.actRoute.snapshot.params.category;

console.log(category);

if(category === null){

console.log('Post category param does not exist');

}

}

}

TypeScript

COPY

**Angular URL Parameter with Question Mark Example**

In the previous paradigm, we used a slash separator, but now you will see the usage of a question mark and ampersand in the angular route.

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

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

import { BlogComponent } from './blog/blog.component';

import { PostDetailComponent } from './post-detail/post-detail.component';

const routes: Routes = [

{

path: '',

component: BlogComponent

},

{

path: 'post?postid=5555&category=javascript',

component: PostDetailComponent

}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

TypeScript

COPY

Here is the URL structure to display route with question mark and ampersand:

http://www.positronx.io/post?postid=5555&category=javascript

Bash

COPY

import { Component, OnInit } from '@angular/core';

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

@Component({

selector: 'app-post-detail',

templateUrl: './post-detail.component.html',

styleUrls: ['./post-detail.component.css']

})

export class PostDetailComponent implements OnInit {

constructor(private actRoute: ActivatedRoute) { }

ngOnInit(): void {

this.actRoute.queryParams.subscribe(params => {

let postid = params['postid'];

let category = params['category'];

console.log(postid);

console.log(category);

});

}

}

TypeScript

COPY

Further, you will access the route params using the **queryParams** property of the activated route.

**Angular Route Parameter with Hash Separator Example**

Define the parameters in Angular Route using # Hash separator:

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

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

import { BlogComponent } from './blog/blog.component';

import { PostDetailComponent } from './post-detail/post-detail.component';

const routes: Routes = [

{

path: '',

component: BlogComponent

},

{

path: 'post#postid=5555&category=javascript',

component: PostDetailComponent

}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

TypeScript

COPY

Generically, you might need the # Hash parameter in the route URL; here is the quick demo of angular hash parameter route example:

http://www.positronx.io/post#postid=5555&category=javascript

.properties

COPY

Next, you have to find out how to easily get the params property that we declared in the Hash fragment example

import { Component, OnInit } from '@angular/core';

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

@Component({

selector: 'app-post-detail',

templateUrl: './post-detail.component.html',

styleUrls: ['./post-detail.component.css']

})

export class PostDetailComponent implements OnInit {

constructor(private actRoute: ActivatedRoute) { }

ngOnInit(): void {

this.actRoute.fragment.subscribe((res: string) => {

const urlSearchParams = new URLSearchParams(res);

const postid = urlSearchParams.get('postid')

console.log(postid);

const category = urlSearchParams.get('category')

console.log(category);

})

}

}