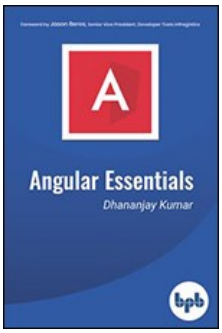

Angular Essentials: The Essential Guide to Learn Angular

by Dhananjay Kumar

Chapter 9: Angular Routing

In this chapter, you will learn about different ways of routing and navigation using strategies, query parameters. The topics that will be covered under this chapter are as follows:

- Create Route
- RouterModule.forChild()
- Routing Strategies
- Dynamic Route Parameters
- Navigate using Code
- Query Parameter
- Child Route
- Route Guards

In Angular Routing, you load a component or more than one component dynamically in Router Outlet. You can have more than one Router Outlet and according to Route configuration, components will be loaded dynamically. You can configure application level route and child routes for feature modules. Angular supports Auxiliary Routes using named Router Outlet. Auxiliary Routes means more than one components can be loaded on DOM on the same URL. In Angular application, when you change URL, a particular component will be loaded depending on the route configuration. Angular Routing also supports either hash based or HTML 5 based URL strategy. In a nutshell, you use routing to load component dynamically from the component tree.

Create Route

Starting Angular 7, while creating a new project using Angular CLI, it will ask whether you want to add Angular Routing or not, as shown in the *figure 9.1*.

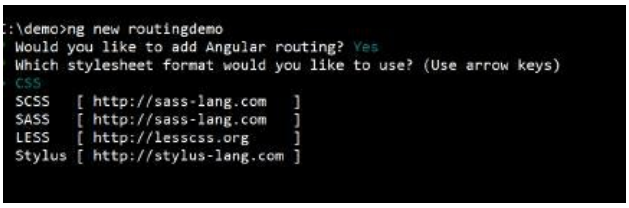

Figure 9.1

If you are working on earlier version of Angular, you can add routing using:

- CLI command
- Manually by creating a routing module

Angular 7 adds a routing module in your project as shown in the *code listing 9.1:*

Code Listing 9.1

app-routing.module.ts

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

const routes: Routes =   [];

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

This is application level routing module, which is imported in **`AppModule`** as shown in the *code listing 9.2:*

Code Listing 9.2:

app.module.ts

```
import { AppRoutingModule } from './app-routing.module'; import  { AppComponent  }  from V/app.component';
@NgModule({
  declarations:   [
      AppComponent
   ] ,
   import:   [
   BrowserModule,
   AppRoutingModule
   ],
   providers:   [],
   btoohptrtp:   [AppComponent]
})
export class AppModule {}
```

To work with routing, let us first add few components to the application. Using Angular CLI, add following components to the application.

- ngp g component login
- ng g component home
- ng g component pagenotfound
- ng g c welcome

Now you should have a project structure as shown in the *figure 9.2:*

Figure 9.2

In Angular,wenavigate from one component to another. You can create a very basic route for **LoginComponent, HomeComponent** as by modifying routes in **app-routing.module.ts** as shown in the *code listing 9.3:*

Code Listing 9.3

```
const routes: Routes =   [
   {path:   'home',  component:  HomeComponent},
   {path:   '',  redirectTo:   'home', pathMatch:   'full'},
   {path:   'login',  component:  LoginComponent},
   {path:   '**',  component:  PagenotfoundComponent}
];
```

Let us walk through the code:

1. The path property defines part of the URL. Therefore, whenever user enters **baseurl/path,** cross ponding component will be loaded.
2. The component property defines component to load for that particular path.
3. The **pathMatch** property set to **full** means that the whole URL path needs to be matched.
4. The **pathMatch** property set to **prefix** means first route where the path matches the start of the URL is chosen, but then the route matching algorithm is continuing to search for matching child routes where the rest of the URL matches.

5. The **pathMatch** property set to ** means that if nothing matches, go here.

Besides, above properties, there are other properties also which we will discuss in subsequent section. Next, we need to find where to load routes. For that, you need to use:

<router-outlet></router-outlet>

This should be used on the root component. We can create a top-level menu to navigate between the components or rather load components directly as shown in the *code listing 9.4:*

Code Listing 9.4

```
<div>
   <nav class="navbar navbar-default">
      <div class = "container-fluid">
         <a class="navbar-brand">{{pageTitle}}</a>
         <ul class="nav navbar-nav">

         <li>
            <a   [routerLink]="['/home']">Home</a>
         </li>
         <li>
            <a>Show Messages</a>
         </li>
         <li>
            <a   [routerLink]=" p/il.ogii.n']"dLog In</a>
         </li>
      </ul >
      <ul claiss = "nav navbar-nav navbar-right">
         <ul>
      >/div>
   >/ nav>
   <div crasst"conta/ner">
      <router-outlet></router-outlet>
   </div>
</iv>
```

As you see to navigate, we are using **`[routerLink]`** property binding and binding it to the route name from the routing configuration. See *figure 9.3&94.*

URL : **baseur 1/home**
AppComponent

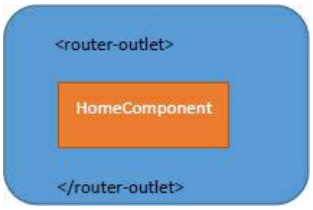

Figure 9.3

URL : **baseur 1/login**
AppComponent

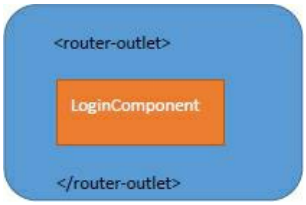

Figure 9.4

So far if you change URL to **`baseurl/home`** then dynamically **`HomeComponent`** will beloaded in the **`<router-outlet>`** and if you change it to **`baseurl/login`** then dynamically **`LoginComponent`** will be loaded.

`RouterModule.forChild()`

Let us start adding another feature module called Product. What we are going to do is to configure its own route for the **`ProductModule`**. To configure routes for feature modules, you need to follow same approach, however instead of **`RouterModule.forRoot()`** use **`RouterModule. forChild()`**. Add code in feature module **`ProductModule`** as shown in the *code listing 9.5:*

Code Listing 9.5

```
Product.module.ts  (Excerpts)
const productRoutes: Routes =   [
   {path:   'products',  component:  ProductsComponent},
   {path:  'addproduct', component: AddproductComponent},
         {path:  'editproduct/:id',  component:
EditproductComponent},
         {path:        'productdetails/:id',        component:
ProductdetailsComponent},
];
@NgModule({
   declarations:   [
         ProductsComponent,
           AddproductComponent,
           EditproductComponent,
           ProductdetailsComponent],
imports:   [
   CommonModule,
   RouterModule.forChild(productRoutes)
 ]
})
export class ProductModule {  }
```

Let us talk through code, we have configured routing array, and then using **`RouterModule.forChild`**O to create route. We are importing **`ProductModule`** in **`AppModule`** as shown in the *code listing 9.6:*

Code Listing 9.6

```
app.module.ts  (Excerpts)
@NgModule({
declarations:   [
   AppComponent,
   PagenotfoundComponent,
   HomeComponent,
   WelcomeComponent,
   LoginComponent
],
imports:   [
   BrowserModule,
   ProductModule,
   ApcRoutingModule
] ,
provider's:  [],
   bootstrap:   [AppComponent]
} )
export  class AppModule {}
```

Essentially **`RouterModule`** has two methods to create routes:

1. **`forRoot()`** methodshould be used only once, as it creates instance of Router Service. You need only one instance per application to work with this.
2. **`forRoot ()`** is used coconfigure appcation level route.
3. **`forRoot ()`** is used Co declare the rcmter directives.
4. For feature module, you should use `forChild()` method.
5. **`forChild ()`**method does notregistel Router Service.
6. Both **`forRoot()`** and **`forChild()`** methods accepts array of routes.

In our example above, we are using **`forRoot()`** method with **`AppModule`** because that is application level module. For

`ProductModule` which is a feature module, we are using **`forChild()`** method.

Also, one important thing you need to keep in mind that always load routes configured using **`forChild()`** before routes configured using **`forRoot()`**.

At this point of time on running application, you should have routing and navigation enabled as shown in the *figures 9.5*, *9.6* & *9.7:*

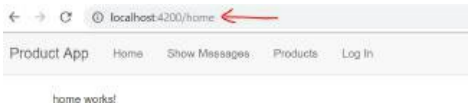

Figure 9.5

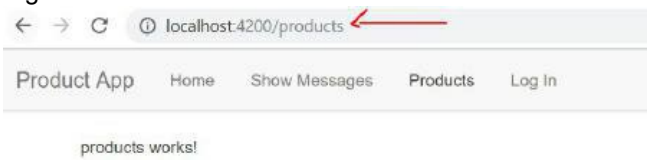

Figure 9.6

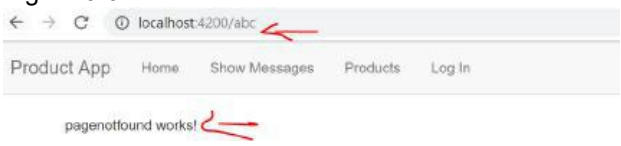

Figure 9.7

As we change URL, cross ponding component is loading dynamically inside **`<router-outlet>`** which is placed onthe **`AppComponent`**.At this point of time, template of **`AppComponent`** should look like *code listing 9.7:*

Code Listing 9.7

```
app.component.html
<div>
   <nav class="navbar navbar-default">
      <div class = "container-fluid">
         <a claiss = "navbar-brand">{{pageTitle}}</a>
         <ul class="nav navbar-nav">
            <li>
                 <a  [routerLink]="['/home']">Home</a>
            </li>
            <li>
               <a>Show Messages</a>
            </li>
            <li>
         <a [routerLink] = "[ 1 /products']">Products</
a>
         </li>
         <li>
            <a   [routerLink]="['/login']">Log
In</a>
            </li>
         </ul>
         <ul class="nav navbar-nav navbar-right">

         </ul>
         </div>
      </nav>
      <div class="container">
         <router-outlet></router-outlet>
   </div>
</div>
```

Routing Strategies

Angular Routing allows us to choose either of URL styling:

- HTML 5 style URL or PathLocationStrategy

- Hash-based URL or HashLocationStrategy

To enable hash-based URL or **`HashLocationStrategy`**, you need to configure app routing to use hash that can be done as shown in the *code listing 9.8:*

Code Listing 9.8

```
app-routing.module.ts
@NgModule({
   imports: [RouterModule.forRoot(routes, { useHash:
true})],
   exports:   [RouterModule]
})
export class AppRoutingModule {   }
```

Now when run the application, you will find hash-based URL as shown in the *figure 9.8:*

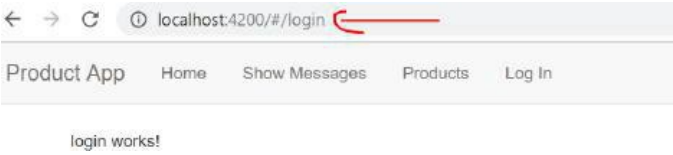

Figure 9.8

Main advantages of using that it worksin old browsers also. The # part oftheURL iscalled hashfragment. Biggest advantage of hash# is that anything after hash never gets sent to server. So, for example, if you have URL

Basurl:/Product/#/Home

On theserver,only **baseurl:/Product** willbesent andbrowser will ignore anything after #. Thisisveryuseful to maintain somedata atthe client side and for client side navigation.

Since hash fragmentis neversentto theserver, client side state can be savedin hash. Itis ideal forSingle Page Application with baseurl always same for the server. Another advantage is hash fragment can be changed using JavaScript at the client side.

Thedefault Angular routingstrategy is **`PathLocationStrategy`**. It takes advtange ofHTML 5pushstate apito maintain thestate ofthe URL. To work with this, you have to set the base URL as shown in the next listing:

<base href="/">

Since, it is default strategy you don't have to configure anything to enable it. By using this URL can be changed without requesting the server and without using the hash fragment. This is very useful for Single Page Application with one major challenge that if you hard hit a URL:

Basurl:/Product/home

Servershould beable to return code for wholeURL instead of only root URL. For this you got to write cross ponding code at the server. If you are doing development using Angular CLI that takes care of that.

Dynamic Route Parameters

We may have requirement to create route dynamically. To create route dynamically, you need to pass route parameters. You can pass route parameters using single colon in route configuration. We did that in Product Route Configuration as shown in the *code listing 9.9:*

Code Listing 9.9:

```
const productRoutes:  Routes =   [
   {path:   'prodogts:,  component:  ProductsComponent},
```

```
   {path:  {addproduct' : component: AddneoduotComponent},
         {path:  'editproduct/:id',   component:
EditproductComponent},
         {path:        'productdetails/:id',        component:
ProductdetailsComponent},
];
```

As you see that when you navigate to**basurl/edit/1,EditProductComponent** will be loaded with dynamic data id passed to the component. See *figure 9.9*.

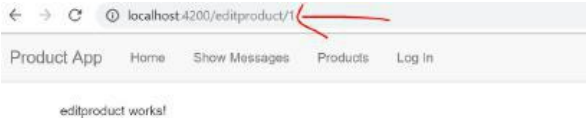

Figure 9.9

You can read passed data in the component. To read that you need to use **`ActivatedRoute`** service. First, import it and inject it in the component class. You need to import **`ActivatedRoute`** service from **`@angular/ router`** andthen injectitas shown inthe *codelisting 9.10.*

Code Listing 9.10

```
export class EditproductComponent implements Onlnit  {

   productidtoedit:  any;
   constructor(private route: ActivatedRoute)   {   }

   ngOnInit()   {
      this:route.params.subscribe(
      (p)  => {
         this.productidtoedit p.id;
});
   }
}
```

We are subscribing to params, which is a property of **`ActivatedRoute`** service. Since it is an observable, any change in route parameter will be notified.

There is one more way to read the route parameters. Instead of observable approach, you can use snapshot approach. See *code listing 9.11:*

Code Listing 9.11

```
   ngOnInit()   {

      this.productidtoedit = this.route.snapshot.
params['id'];
   }
```

Snapshot code is easier to implement and used to read parameter only once. Whereas observable code is complex but it watches for the parameter changes.

Navigate Using Code

To navigate using code we need to use imperative API that router provides. You may need to navigate using the code in various scenarios such as Master-Details etc. Let us say that on **`ProductsComponent`** is rendered as shown in the *figure 9.10:*

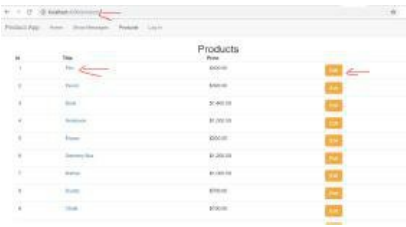

Figure 9.10

On clicking of Product Title, **`ProductdetailsComponent`** Component should be loaded and on click of **Edit** button, **`EditproductComponent`** should be loaded in the **`<router-outlet>`**. In addition, we are passing route parameters.

In template, we can use **`[routerLink]`** to navigate as shown in the *code listing 9.12:*

Code Listing 9.12

```
<a   [routerLink]="p/productdetails',  p.Id]">{{p. Title}}</a>
```

However, for **Edit** button, we need to write code in component class. For your reference template of **`ProductsComponent`** is as shown in the *code listing 9.13:*

Code Listing 9.13

```
Products.component.html
<div class="row">
   <h2 class="text-center">Products</h2>
</div>
<div class="row">
   <table class="table">
      <thead>
         <th>Id</th>
         <th>Titie</th>
         <th>Price</th>
      </thead>
      <tbody>
        <tr *ngFor="let p of products">
            <td>{{p.Id}} </td>
               <td><a   [routerLink]="['/productdetails',
p.Id]">{{p.Title}}</a></td>
         <td>{{p.Price   |  currency }}</td>
               <td><button   class="btn   btn-warning"
(click)='editProduct(p.Id)'>Edit</button></td>
      </tr>
   </tbody>
</table>
</div>
```

Now to navigate on the **Edit** button to **`EditproductComponent`**, we need to follow the following steps:

1. Import Router from @angular/router
2. Inject it in constructor of component class
3. Use navigate() method to navigate using code

Therefore, you can navigate to **`editproduct`** route as shown in the *code listing 9.14:*

Code Listing 9.14

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

editProduct(id)   {
      this.router.navigate([,editproduct',  id]);
   }
```

We are using router navigate method, in which passing route name and query parameter. For your reference whole source code for **`ProductsComponent`** is shown in the *code listing 9.15:*

Code Listing 9.15

```
Products.component.ts
import  {  Component,  Onlnit  }  from '@angular/core';
import  { Router }  from '@angular/router';

@Component({
   selector:   'app-products',
   templateUrl:   V/products.component.html',
```

```
   styleUrls:   ['./products.component.css']
})
export class ProductsComponent implements Onlnit  {

   products: any;

   constructor(private router: Router)   {  }

   editProduct(id)   {

 this.router.navigate(['editproduct',  id]);
}
ngOnInit()   {
   // in real app data will be fetched from API this.products = this.getProducts();
}
getProducts()   {
   return   [
       {
      Id: '1', Title: 'Pen', Price: '400'
   },
   {
      Id: '2', Title: 'Pencil', Price: '300'
   },
   {
      Id: '3', Title: 'Book', Price: '1400'
   },
   {
      Id: '4', Title: 'Notebook', Price: '1000'
   },
   {
      Id: '5', Title: 'Eraser', Price: '200'
   },
   {
      Id: '6', Title: 'Geomtry Box', Price: '1200'
   },
   {
      Id: '7', Title: 'Marker', Price: '1000'
   },
   {
      Id:   '8',  Title:   'Duster',  Price:   '700'
   },
   {
      Id:   '9',  Title:   'Chalk',  Price:   '100'
   },
   {
       Id:   '10',  Title:   'Stapler',  Price:   '1300'
      }
     ];
  }
}
```

On the **`EditproductComponent`**, you can read route parameter as shown in the *code listing 9.16:*

Code Listing 9.16

editproduct.component.ts

```
export class EditproductComponent implements OnInit  {

  productidtoedit: any;
  constructor(private route: ActivatedRoute)   {   }
  ngOnInit() {
      this.route.params.subscribe(
         (p) => {
            this.productidtoedit = p.id;
         });
   }
}
```

On the template, you can implement back button as shown in the *code listing 9.17:*

Code Listing 9.17

---

editproduct.component.html

```
<h2>
   Product to Edit  :  {{productidtoedit}}
</h2>
```

```
<button class="btn btn-default"   [routerLink]="['/
products']">Back</button>
```

Query Parameter

Route Parameters are required to navigate to a route. Query parameters allow you to pass optional parameters to a route such as pagination information.

You can pass query parameter as in the *code listing 9.18:*

Code Listing 9.18

```
<a  [routerLink]="p/productdetails', p.Id]"
[queryParams]="{page:l}">{{p.Title}}</a>
```

You can read query parameter as shown in *code listing 9.19:*

Code Listing 9.19

```
page:  any;
   constructor(
      private route: ActivatedRoute)   {}
   ngOnInit() {
      this.route.queryParams.subscribe(
            p => {
         this.page = +pppage']   ||  0;
           });
   }
```

Query Parameter should be used to pass optional parameters.

Child Route

Angular allows us to create child route. Every Angular route can support a child route inside it. Let us consider that for **`editproduct`** route, we need two child routes.

- Edit by sales representative
- Edit by manager of the product

We can achieve that using **Child Routes** inside editproduct route. We can create a child route as shown in the *code listing 9.20:*

Code Listing 9.20

```
{path:   'editproduct/:id',
   component:  EditproductComponent,
   children:   [
      {  path:   w,   redirectTo:   'bysalesrep',   pathMatch:
'full'   },
            {       path:        'bymanager',       component:
EditproductbymanagerComponent  },
            {       path:       'bysalesrep',       component:
Editproductbyesalesrep Component },
   ],
   }
```

We have added child routes to **`editproduct`** route using the children property. After adding child routes, whole routing configuration for product feature module will look like as shown in *code listing 9.21:*

Code Listing 9.21:

`product.module.ts`

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

```
import  {  CommonModule  }  from '@angular/common';
import { ProductsComponent } from './products/products. component';
import { AddproductComponent } from './addproduct/ addproduct.
component';
import { EditproductComponent } from './aditproduct/
aditproduct.component';
import { RouterModule, Routes } from '@angular/router';
import      {      ProductdetailsComponent      }      from './
productdetails/productdetails.component';
import     {     EditproductbyesalesrepComponent       }
from      './editproduct/editproductbyesalesrep/
editproductbyesalesrep.component';
import    {    EditproductbymanagerComponent    }    from  './
editproduct/editproductbymanager/editproductbymanager.
component';

const productRoutes: Routes =   [
     {path:   'products',  component:  ProductsComponent},
   {path:  'addproduct', component: AddproductComponent},
     {path:   'editproduct/:id',
       component: EditproductComponent,
       children:   [
      {  path:   w,   redirectTo:   'bysalesrep',   pathMatch:
'full'   },
            {       path:        'bymanager',       component:
EditproductbymanagerComponent },
            {       path:       'bysalesrep',       component:
EditproductbyesalesrepComponent },
    ],
   },
         {path:        'productdetails/:id',        component:
ProductdetailsComponent},
];

@NgModule({
   declarations:   [
            ProductsComponent,
              AddproductComponent,
              EditproductComponent,
              ProductdetailsComponent,
              EditproductbyesalesrepComponent,
              EditproductbymanagerComponent],
   imports:   [
      CommonModule,
      RouterModule.forChild(productRoutes)
   ]
})
export class ProductModule {  }
```

Next, we need to put a **`<router-outlet>`** on **`editproduct.component. html`**, in which child route will be dynamically loaded. See *Code Listing 9.22:*

Code Listing 9.22

```
<div class="row">
    <h2>Product to Edit  :  {{productidtoedit}} </h2>
  </div>

   <nav>
      <a  [routerLink]="pbysalesrep']">SalesRep</a>
      <a  [routerLink]="pbymanager']">Manager</a>
   </nav>

   <router-outlet></router-outlet>

   <br/>
   <br/>
   <button class="btn btn-default"   [routerLink]="'./
products']">Back</button>
   <br/>
```

Here, we are creating navigation and then loading child routes in the **<router-outlet>.** You can read route parameter passed from parent route in child route component as shown in *code listing 9.23:*

Code Listing 9.23

```
export class EditproductbymanagerComponent implements OnInit {

   productid: any;
   constructor(private route: ActivatedRoute)   {   }

   ngOnInit() {
      //     this.route.parent.data.subscribe(data => {
      //    this.productid = data['id'];
      // });
      this.route.parent.params.subscribe(d => {
         this.productid = d[id'];
      });
   }
}
```

We are using parent method of **`ActivateRoute`** to fetch parameter of parent route. On running application, you will find child route for edit product route enabled as shown in the *figures 9.11* & *9.12:*

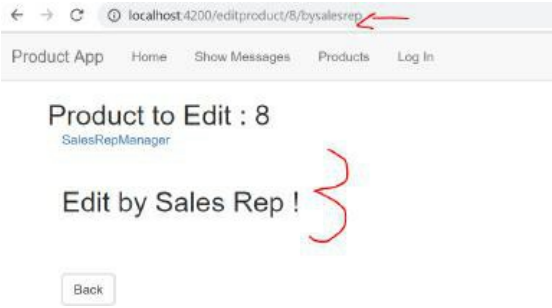

Figure 9.11

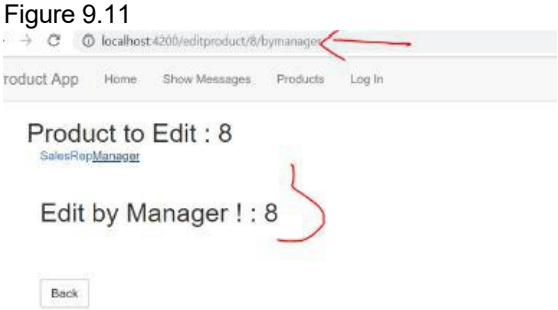

Figure 9.12

As you see in these images, child route is getting activated for **`editproduct`** route.

Auxiliary Route

Angular provides us to have more than one **`<router-outlet>`** in application. You can have named **`<router-outlet>`** and configure routes to load component in a specific named **`<router-outlet>`**. Secondary route is also known as Secondary

Route. To create an Auxiliary route, add **`<router-outlet>`** as shown in the *code listing 9.24:*

Code Listing 9.24

```
app.component.html
         <div class="col-md-9">
            <router-outlet></router-outlet>
         </div>

         <div class="col-md-3">
              <router-outlet name="showmessage"></
router-outlet>
         </div>
```

Now while configuring route, you can pass outlet value to determine in which router outlet, component would be loaded. You can do that as shown in the *code listing 9.25:*

Code Listing 9.25

```
app-routing.module.ts
//other codes const
routes: Routes = [
   {path: 'home', component: HomeComponent},
   {path: '', redirectTo: 'home', pathMatch: 'full'},
   {path: 'login', component: LoginComponent},
     {path: 'showmessage', component: WelcomeComponent,
outlet: 'showmessage' },
   {path: '**', component: PagenotfoundComponent} ];

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

You can activate or navigate to auxiliary route as shown in the *code listing 9.26:*

Code Listing 9.26

```
<a [routerLink]="[{outlets:{showmessage:
['showmessage']}}]"> Show Messages</a>
```

While activating, you can pass route name and outlet name. Now you can see in *figure 9.13* that on click of Show Messages, URL is changing and in auxiliary route, Welcome component is loaded.

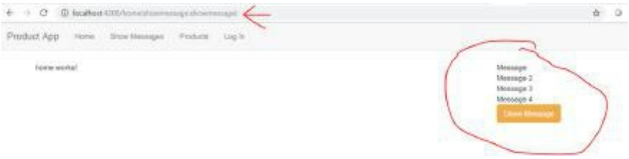

Figure 9.13

You cancloseanauxiliary route bynavigating to **`showmessage`** route in the code. Let us say you have a button on template of **`WelcomeComponent`** and on click of that button, you wish to close secondary route that can be done as shownin *code listing 9.27:*

Code Listing 9.27

```
welcome.component.ts
constructor(private router: Router) { }
     closemessage() {
        // To naviagte
           // this.router.navigate([{outlets: {showmessage:
['showmessage']}}]);
        // To close the router
           this.router.navigate([{outlets: {showmessage:
null}}]);
     }
```

You can have URL changing for auxiliary routes as shown in the *f igures 9.14&9.15:*

URL : **baseurl/home** URL : **home(showmessage:showmessage)** **AppComponent** **AppComponent**

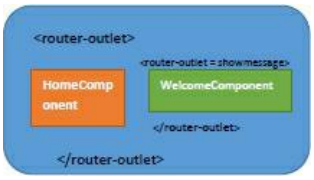

Figure 9.14

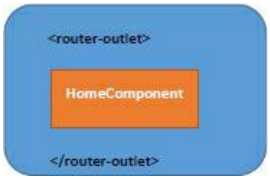

Figure 9.15

You should use auxiliary routes when you have toload morethan one component dynamically. The `<router-outlet>` without name is default. Besidesthat, you can have anynumber of named `<router-outlet>` as auxiliary or secondary routes. This is useful in master-detail scenario.

Route Guards

There are four types of route guards available inthe Angular routing. They are as follows:

1. CanActivate
2. CanActivateChild
3. CanLoad
4. CanDeactivate

Each route guards have different purposes. For example, `CanActivate` route guard controls whether a particular route will be activated or not and `CanActivateChild` controls whether child routes of a particular route will be activated or not. Summary of purposes of all four-route guards is shown in the *figure 9.16*.

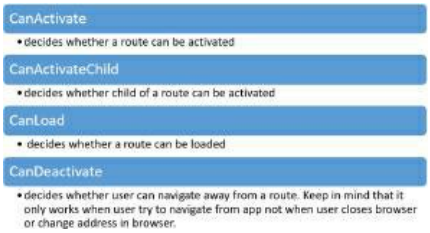

Figure 9.16

Let us create a simple route guard. To create a route guard, you need to follow steps:

1. Create a class.
2. Implement required route guard interface.

To create `canActivate` route guard, you need to implement `canActivate` interface in a class as shown in *code listing 9.28:*

Code Listing 9.28

```
import { Injectable } from '@angular/core';
import {
      CanActivate,
      ActivatedRouteSnapshot,
      RouterStateSnapshot
} from '@angular/router';

@Injectable()
export class CanActivateProductRouteGuard implements
CanActivate {

      constructor() { }

        canActivate(route: ActivatedRouteSnapshot, state:
RouterStateSnapshot): boolean {
         return false;
      }
}
```

In above route guard, activation logic is set to false, however in real application you will use a authentication service to determine, whether a particular route should be activated or not. Let us assume that you have a service to find whether a user should navigate to product route or not. In that case, you can use the service as shown in *code listing 9.29.*

Code Listing 9.29

```
 import { Injectable } from '@angular/core';
import {
     CanActivate,
     ActivatedRouteSnapshot,
     RouterStateSnapshot
} from '@angular/router';

import { ProductAuthService } from './product.auth.
service';

@Injectable()
export class CanActivateProductRouteGuard  implements
CanActivate {

   constructor(private loginauth: ProductAuthService)
{ }

      canActivate(route: ActivatedRouteSnapshot, state:
RouterStateSnapshot): boolean {
         return this.loginauth.isUserAuthenticated();
      }
}
```

To use a route guard, first you need to pass it in providers array of module, and then use it in route as shown in *code listing 9.30.*

Code Listing 9.30

```
const productRoutes: Routes = [
     {path: 'products', component: ProductsComponent,
canActivate : [CanActivateProductRouteGuard]},
   {path: 'addproduct', component: AddproductComponent},
           {path: 'productdetails/:id', component:
ProductdetailsComponent}
];
```

You are passing route guard in **`canActivate`** property of route. In this way, you can create and use a route guard in Angular routing.

Summary

In this chapter, we learnt about Routing and navigating from one route to another. We also learnt how there can be different routes like parent and child routes. In this chapter you learnt about following topics:

- Creating Routes
- Route Strategies
- Dynamic route Parameters
- Navigating using code
- Query Parameter
- Route Guards