**What is Routing in Angular?**

The Angular Routing is a mechanism which is used for navigating between pages and displaying appropriate component or pages on the browser.

In other words we can say that the Routing in Angular Application helps us to navigate across the application from one one view to another view. It also allows us to maintain the state, implement modules, and then loads the modules based on the roles of the user

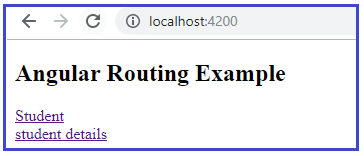
##### ****Why Routing in Angular Application?****

We access our application through one URL such as **http://localhost:4200** and our application is not aware of any other URLs such as **http://localhost:4200/Login**

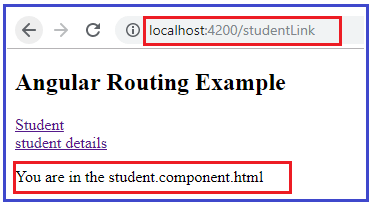
Most web applications need to support different URLs to navigate different pages of the application. This is where angular routing comes into picture.

##### ****Understanding Angular Routing with an example:****

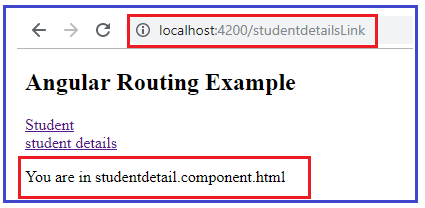
Let us understand the Routing in Angular Application with an example. We want to create a page as shown below.



When the user click on the student link, we need to display the following.



And when the user click on the student details link, we need to display the following.



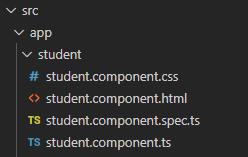
Let us see how to implemeny this using Angular Routing. Here, we need to create two components i.e. student and studentdetails component.

##### ****Creating Student Component:****

Open terminal and type **ng g c student**and press enter as shown in the below image.

Creating Student Component:

Once you press enter it will create four files within the **student**folder which you can find within the **src/app** folder as shown in the below image.



###### **Modifying the student.component.html file:**

Open student.component.html file and then copy and paste the following code in it.

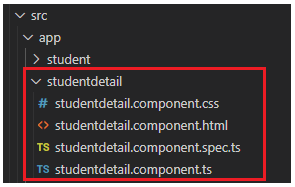
**<p>You are in the student.component.html</p>**

##### ****Creating Student Details Component:****

Open terminal and type **ng g c studentdetail**and press enter as shown in the below image.

Creating Student Details Component:

Once you press enter it will create four files within the **studentdetail**folder which you can find inside the **src/app** folder as shown in the below image.



###### **Modifying studentdetail.component.html file:**

Open **studentdetail.component.html** file and then copy and paste the following code in it.

**<p>You are in studentdetail.component.html</p>**

##### ****Adding Routing in Angular:****

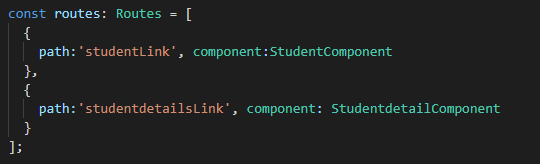
When you are creating an angular 9 application, at the time of creation it is asking whether you want to add angular routing to your project or not. If you select yes, then it automatically add the routing model in your project. If you select no then you need to add it manually. You can use the below CLI code to generate the router in Angular Application.

**ng generate module app-routing –flat –module=app**

Here **–flat** puts the file in **src/app** folder instead of its own folder and **–module=app** tells the CLI to register it in the imports array of the AppModule.

##### ****Configuring Routes in Angular Application:****

Once you created the routing module, then you need to configure the path and their respective component in the AppRoutingModule as shown in the below image. As you can see, here we have created two paths i.e. studentLink and studentdetailsLink (you need to use the path properties to set the path. You can give any meaningful name here) and also set the respective components using the component property (Here you need to provide the component class name).



So, open **app-routing.module.ts** file and then copy and paste the following code in it.

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

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

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

**import** **{** StudentdetailComponent **}** from './studentdetail/studentdetail.component';

const routes: Routes = **[**

**{**

path:'studentLink', component:StudentComponent

**}**,

**{**

path:'studentdetailsLink', component: StudentdetailComponent

**}**

**]**;

@NgModule**({**

imports: **[**RouterModule.forRoot**(**routes**)]**,

exports: **[**RouterModule**]**

**})**

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

**Note:**While generating the link you need to use the string **studentLink** and **studentdetailsLink**. Let us see how to use these routing path to generate link and navigates.

##### ****Generating to Links:****

In order to generate links, open **app.component.html** file and then copy and paste the following code in it.

**<h2>**Angular Routing Example**</h2>**

**<a** [routerLink] = "['/studentLink']" **>**Student**</a>** **<br/>**

**<a** [routerLink] = "['/studentdetailsLink']" **>**student details**</a>**

**<div>**

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

**</div>**

With the above changes in place, now run the application and you should get the output as expected. From the above HTML Code, we need to understand two important concepts i.e. **routerLink** and **router-outlet**

##### ****Router-outlet:****

The Router Outlet is a dynamic component that the router uses to display views based on router navigation. In our example, whenever the user click on the Student Link, then it will display student component view in the router-outlet div. So, the role of **<router-outlet>** is to mark where the router displays the view. This is the location where angular will insert the component.

The **<router-outlet>** tells the router where to display routed view. The RouterOutlet is one of the router directives that become available to the AppComponent because AppModule imports AppRoutingModule which exported RouterModule.

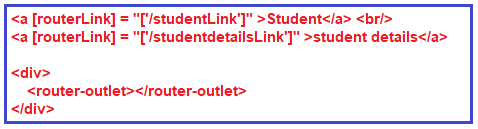
##### ****Router Link:****

With the help of **routerLink**directive, you can link to routes of your application right from the HTML Template. You just need to add the directive to an HTML Element. When the user clicks on that element, angular navigates to that specified location.

The routerLink is the selector for the RouterLink directive that turns user clicks into router navigations. You can assign a string to the Router link. This directive generates the link based on the route path.

##### ****Router Link: Client side****

The syntax is given below. As you can see within the anchor tag we have routerLink and we have specified the studentLink as its path. If you remember we have set this studentLink path in the routing module and it is pointing to the Student Component. So, when the Student List is clicked, then the student component is going to be load on the router-outlet directive.



##### ****Router Link: Server side****

Sometimes it is also required to set the route dynamically based on some condition and that can be done at server side. For your application to work with server side rendering, the element hosting directive has to be a link (anchor) element.

It is also possible to navigate to a route from code. To do so, we need angular router and this need to be done in your typescript file. The syntax is given below.

Router Link: Server side

Once we have the router, then the navigation is quite simple. Just call the navigate function or Router. This function takes an array. The first element of the array defines the route we want to navigate. The second is optional and allows us to pass a route parameter. The syntax is given below.

Understanding Angular Routing with an example

##### ****Let us see an example to understand this:****

First modify the **app.component.ts** file as shown below.

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

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

@Component**({**

selector: 'app-root',

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

styleUrls: **[**'./app.component.css'**]**

**})**

**export** **class** AppComponent **{**

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

GetStudent**()**

**{**

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

**}**

GetStudentDetails**()**

**{**

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

**}**

**}**

Then modify **app.component.html** file as shown below.

**<h2>**Angular Routing Example**</h2>**

**<button** (click)="GetStudent()"**>**Student**</button>**

**<button** (click)="GetStudentDetails()"**>**GetStudentDetails**</button>**

**<div>**

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

**</div>**

With the above changes in place, now run the application and it should works as expected.

##### ****What is Angular Router?****

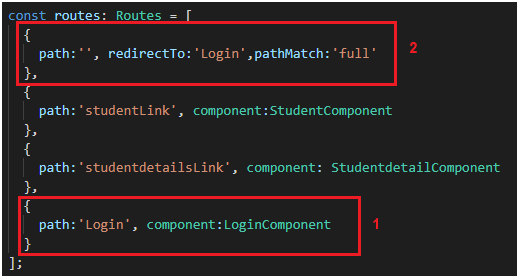
The Angular Router is an official Routing Library managed by the Angular Core Team. Its a JavaScript router implementation that’s designed to work with Angular and is packaged as **@angular/router**

The Angular Router will take cares of the duties of a JavaScript router. It activates all required Angular Components to compose a page when a user navigates to a certain URL. It lets user navigate from one page to another without page reload.

It updates the browsers history so that the user can use the back and forward buttons when navigating back and forth between pages.

##### **Redirecting Routes in Angular Application**Redirecting Routes in Angular:

When the application start, it navigates to the empty route by default. We can configure the router to redirect to a named route by default. So, a redirect route translates the initial relative URL (”) to the desired default path. For example, if may want to redirect to Login page or registration page by default when the application start. Then you need to configure the redirectTo as shown below.



This route redirects a URL that fully matches the empty path to the route whose path is ‘/Login’. The empty path in the first route represents the default path for the application. This default route redirects to the route for the /Login URL and therefore will display the Login Component.

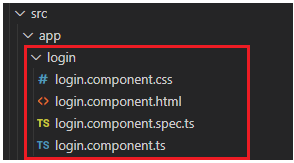
A redirect route requires a **pathMatch** property to tell the router how to match a URL to the path of a route. The router throws an error if you don’t. For the special case of an empty URL we also need to add the **pathMatch**: ‘**full**’ property so angular knows it should be matching exactly the empty string and not partially the empty string.

**Understanding Redirecting Route in Angular with an Example:**

Let us understand the concept Redirecting Route with an example. In order to understand this we are going to create a component called Login Component. So, open terminal and then type **ng g c Login** and press enter as shown in the below image.

Understanding Redirecting Route in Angular with an Example:

Once you press the enter button, it will create a Folder called login and inside that folder it will creates four files as shown in the below image.



**Modify app.component.html file:**

Open **app.component.html** file and then copy and paste the following code in it.

**<h2>**Angular Routing Example**</h2>**

**<a** [routerLink] = "['/studentLink']" **>**Student**</a>** **<br/>**

**<a** [routerLink] = "['/studentdetailsLink']" **>**student details**</a>**

**<div>**

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

**</div>**

**Modify app.component.ts file:**

Open **app.component.ts** file and then copy and paste the following code in it.

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

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

@Component**({**

selector: 'app-root',

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

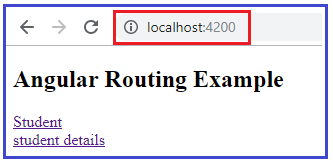
styleUrls: **[**'./app.component.css'**]**

**})**

**export** **class** AppComponent **{**

**}**

At this point, if you run the application, then you will get the following output. If you notice the URL it is empty and displaying default page.



Instead of showing the default page when the URL is empty, we want to redirect to the Login page. This is where Redirecting Routes comes into picture in Angular Application.

**Modifying the app-routing.module.ts file:**

Open app-routing.module.ts file and then copy and paste the following code in it. Here, we did three things. First import the login component. Second create a path for login component and finally create an empty path and set redirectTo property value to the login path.

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

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

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

**import** **{** StudentdetailComponent **}** from './studentdetail/studentdetail.component';

**import** **{** LoginComponent **}** from './login/login.component';

const routes: Routes = **[**

**{**

path:'', redirectTo:'Login',pathMatch:'full'

**}**,

**{**

path:'studentLink', component:StudentComponent

**}**,

**{**

path:'studentdetailsLink', component: StudentdetailComponent

**}**,

**{**

path:'Login', component:LoginComponent

**}**

**]**;

@NgModule**({**

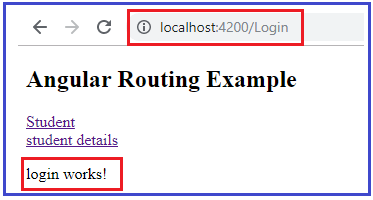
imports: **[**RouterModule.forRoot**(**routes**)]**,

exports: **[**RouterModule**]**

**})**

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

With the above changes in place, now run the application, go to the default URL i.e. **http://localhost:4200** it will automatically redirect to the Login page as shown in the below image.



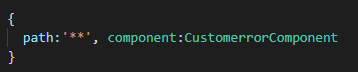
## **Wildcard Route in Angular Application**

**What is Wildcard Route in Angular?**

The **Wildcard Route** is basically used in Angular Application to handle the invalid URLs. Whenever the user enter some invalid URL or if you have deleted some existing URL from your application, then by default 404 page not found error page is displayed. In such scenarios instead of showing the default error page, if you also show a custom error page and this is possible by using the **Angular Wildcard Route**.

**How to use Wildcard Route in Angular?**

A Wildcard route has a path consisting of two asterisks (\*\*). It matches every URL, the router will select this route if it can’t match a route earlier in the configuration. A Wildcard Route can navigate to a custom component or can redirect to an existing route. The syntax to use Wildcard Route is given below.



**Understanding Angular Wildcard Route with an example:**

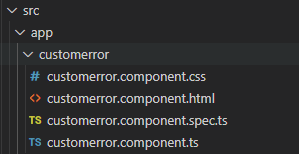
Let us understand Wildcard Route with an example. First create one component for display the custom error message.

**Creating Custom Error Component:**

Open terminal and type **ng g c customerror** and press the enter button as shown in the below image.

What is Wildcard Route in Angular?

Once you press the enter button it will create a folder called **customerror**with four files as shown in the below image.



Now open **customerror.component.html** file and then copy and paste the following code in it.

**<h3>**The page you are looking for does not exist**</h3>**

Next we need to create the Wildcard Route.

**Creating Wildcard Route:**

Open **app-routing.module.ts** file and then copy and paste the following code in it. As you can see the last route is the Wildcard route.

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

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

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

**import** **{** StudentdetailComponent **}** from './studentdetail/studentdetail.component';

**import** **{** LoginComponent **}** from './login/login.component';

**import** **{** CustomerrorComponent **}** from './customerror/customerror.component';

const routes: Routes = **[**

**{**

path:'', redirectTo:'Login',pathMatch:'full'

**}**,

**{**

path:'studentLink', component:StudentComponent

**}**,

**{**

path:'studentdetailsLink', component: StudentdetailComponent

**}**,

**{**

path:'Login', component:LoginComponent

**}**,

**{**

path:'\*\*', component:CustomerrorComponent

**}**

**]**;

@NgModule**({**

imports: **[**RouterModule.forRoot**(**routes**)]**,

exports: **[**RouterModule**]**

**})**

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

Next we need to create one invalid link.

**Creating Invalid Link:**

Open **app.component.html**file and then copy and paste the following code in it. Here, the third link is an Invalid link as we don’t have any path with the name invalidLink.

**<h2>**Angular Routing Example**</h2>**

**<a** [routerLink] = "['/studentLink']" **>**Student**</a>** **<br/>**

**<a** [routerLink] = "['/studentdetailsLink']" **>**student details**</a><br/>**

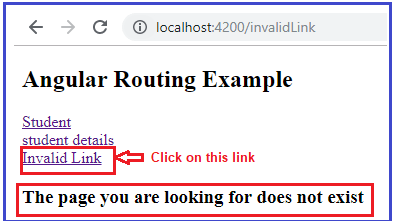
**<a** [routerLink] = "['/invalidLink']" **>**Invalid Link**</a>**

**<div>**

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

**</div>**

Now save all the change and open the browser and click on the Invalid link and you should see the custom error message as shown in the below image.



**Note:** If you add the Wildcard Route as the first route in Router Configuration then no other routes will be reached as the Wildcard Route always matched. In order to understand this better, please modify the **app-routing.module.ts**file as shown below as you can see we have configured the Wildcard route as the first route.

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

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

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

**import** **{** StudentdetailComponent **}** from './studentdetail/studentdetail.component';

**import** **{** LoginComponent **}** from './login/login.component';

**import** **{** CustomerrorComponent **}** from './customerror/customerror.component';

const routes: Routes = **[**

**{**

path:'\*\*', component:CustomerrorComponent

**}**,

**{**

path:'', redirectTo:'Login',pathMatch:'full'

**}**,

**{**

path:'studentLink', component:StudentComponent

**}**,

**{**

path:'studentdetailsLink', component: StudentdetailComponent

**}**,

**{**

path:'Login', component:LoginComponent

**}**,

**]**;

@NgModule**({**

imports: **[**RouterModule.forRoot**(**routes**)]**,

exports: **[**RouterModule**]**

**})**

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

Now save the change, and open the browser and click any of the links, you will always get the Custom error message.

**What should be the Order of Angular Routes?**

The order of the routes is very important. When matching routes find, the angular router uses first-match wins strategy. So more specific routes should be placed above less specific routes. So, Routes with a static path should be placed first, followed by the empty path route, that matches the default route. The wildcard route should be the last route in your router configuration as shown in the below image.

