Experiment No.: 09

Title: Demonstrate Page Routing.

Objectives:

1. To study page routing in angular.

Theory:

The Angular Router enables navigation from one view to the next as users perform application tasks. The browser is a familiar model of application navigation:

- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The Angular Router borrows from this model. It can interpret a browser URL as an instruction to navigate to a client-generated view. It can pass optional parameters along to the supporting view component that help it decide what specific content to present. You can bind the router to links on a page and it will navigate to the appropriate application view when the user clicks a link. You can navigate imperatively when the user clicks a button, selects from a drop box, or in response to some other stimulus from any source. And the router logs activity in the browser's history journal so the back and forward buttons work as well.

Example :

First we are going to need some additional components in the application. Let’s add a dashboard, sidebar and banner:

- ng g component core/dashboard
- ng g component core/sidebar
- ng g component core/banner

These components should be added under the core/ folder and declared in our CoreModule. Since these are root components for our CoreModule we don’t need to create a feature module for them. Next we are going to set up our application routing,

The first thing we will have to do is add an AppRoutingModule to define our root routes. In a

previous post we had placed our core component selector in our app.component.html. Instead of using the selector, we will setup routing and use routing to display our CoreComponent in app.component.html using a RouterOutlet.

To do so, do the following:

1. If your are following along from previous posts, replace <app-core></appcore> with <router-outlet></router-outlet> inside app.component.html. Otherwise replace the content in app.component.html with <router-outlet></router-outlet>.
2. Create a file called app-routing.module.ts at the same level as the AppComponent
   - In the future, when you create a feature module that is going to have routing. Use a --routing flag with the generate command and this file will be created for you.
3. Add the following code to app-routing.modules.ts

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

4. Add the new AppRoutingModule to the imports array in AppModule

If you run ng serve you will notice that the application is the same as when we had <app-core></app-core> inside of app.component.html. The difference now is that <routeroutlet></router-outlet> is dynamically selecting the CoreComponent based on the state of the Angular router. The <router-outlet></router-outlet> will check for a path corresponding to the routes provided to the RouterModule , in this case routes, and display the component that goes with that path.

In this case we have created two routes, one with the empty path and one with what is known as a wildcard path. Having CoreComponent with the empty path means that when you navigate to localhost:4200 then CoreComponent will be routed to. The wildcard path will match with any unrecognized paths. That means that any path we put after localhost:4200 will display the CoreComponent.

Key Concept: Angular Routing.