**Experiment No.: 05**

**Title:** Demonstrate page routing in angular

**Objectives:**

1. To demonstrate routing in angular to show required component according to the requirement.

**Theory:**

Routing in Angular refers to the mechanism of navigating between different components or views within an Angular application. Angular provides its own routing module called the Angular Router, which helps manage navigation and rendering different components based on the URL.

To use the Angular router, an application needs to have at least two components so that it can navigate from one to the other. To create a component using the CLI, enter the following at the command line where first is the name of your component:

*ng generate component first*

Repeat this step for a second component but give it a different name. Here, the new name is second.

*ng generate component second*

The CLI automatically appends Component, so if you were to write first-component, your component would be FirstComponentComponent.

1. Importing your new components:

To use your new components, import them into app.routes.ts at the top of the file, as follows:

import {FirstComponent} from './first/first.component';

import {SecondComponent} from './second/second.component';

2. Defining a basic route

There are three fundamental building blocks to creating a route. Import the routes into app.config.ts and add it to the provideRouter function.

The Angular CLI performs this step for you. However, if you are creating an application manually or working with an existing, non-CLI application, verify that the imports and configuration are correct. The following is the default ApplicationConfig using the CLI.

export const appConfig: ApplicationConfig = {

providers: [provideRouter(routes)]

};

1. Set up a Routes array for your routes

The Angular CLI performs this step automatically.

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

export const routes: Routes = [];

2. Define your routes in your Routes array.

Each route in this array is a JavaScript object that contains two properties. The first property, path, defines the URL path for the route. The second property, component, defines the component Angular should use for the corresponding path.

const routes: Routes = [

{ path: 'first-component', component: FirstComponent },

{ path: 'second-component', component: SecondComponent },

];

3. Add your routes to your application.

Now that you have defined your routes, add them to your application. First, add links to the two components. Assign the anchor tag that you want to add the route to the routerLink attribute. Set the value of the attribute to the component to show when a user clicks on each link. Next, update your component template to include <router-outlet>. This element informs Angular to update the application view with the component for the selected route.

<h1>Angular Router App</h1>

<nav>

<ul>

<li><a routerLink="/first-component" routerLinkActive="active" ariaCurrentWhenActive="page">First Component</a></li>

<li><a routerLink="/second-component" routerLinkActive="active" ariaCurrentWhenActive="page">Second Component</a></li>

</ul>

</nav>

<!-- The routed views render in the <router-outlet>-->

<router-outlet></router-outlet>

You also need to add the RouterLink, RouterLinkActive, and RouterOutlet to the imports array of AppComponent.

@Component({

selector: 'app-root',

standalone: true,

imports: [CommonModule, RouterOutlet, RouterLink, RouterLinkActive],

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

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

})

export class AppComponent {

title = 'routing-app';

}

When user clicks on First Component, he/she gets first component’s content in <router-outlet> element. Simillarl, if user clicks on Second Component, he/she gets second component’s content in <router-outlet> element.

**Key Concept:** Components, Routing

**Steps:**

1. Create Angular App with name my-first-app.

2. Create two components using ng generate component name\_component

3. Set up the routes.

4. Check the result by clicking on different links