STEP 15: Routing In Angular

1. A routed Angular application has one singleton instance of the Router service. When browser's URL changes, that router looks for a corresponding Route from which it can determine the component to display. A router has no routes until configured
2. Open app.routing.module.ts file
3. To configure routes, add paths and the corresponding component name to be displayed for that path as follows:

   ```
   const routes: Routes = [
       {path:'employees',   component: EmployeelistComponent},
       {path:'add',   component:EmpformComponent },
   ];
   ```

4. Need to tell angular which part of the html page content of component will be displayed. So add below in app.component.html and comment out rest of the code except header and footer

   ```
   <router-outlet></router-outlet>
   ```

5. Next go to the browser and type in below and see the component content getting displayed

   ```
   localhost:4200/employees
   localhost:4200/add
   ```

6. Lets add links instead of typing the url manually in header component:

   ```
   <nav class="navbar navbar-expand-lg navbar-light bg-light">
       <div class="container-fluid">
         <a class="navbar-brand" href="#">EMS</a>
         <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
           <span class="navbar-toggler-icon"></span>
         </button>
         <div class="collapse navbar-collapse" id="navbarSupportedContent">
           <ul class="navbar-nav me-auto mb-2 mb-lg-0">
             <li class="nav-item">
               <a class="nav-link" routerLinkActive="active" routerLink="/employees">Employees List</a>
             </li>
             <li class="nav-item">
               <a class="nav-link" routerLinkActive="active" routerLink="/add">Add Employee</a>
             </li>
           </ul>
         </div>
   ```

```
  </div>
 </nav>
```

The RouterLink directives on the anchor tags give the router control over the elements. The navigation paths are fixed, so you can assign a string to the routerLink (a "one-time" binding).
The RouterLinkActive directive on each anchor tag helps visually distinguish the anchor for the currently selected "active" route

7. Now if we route to any other path apart from configured we get error cannot match any routes.
   To handle all other routes apart from one's configured use wild route as follows :

   ```
   {path:'**',component:PagenotfoundComponent}
   ```

   Make sure to add the above route as the last route since wild route accepts requests for all routes

   CREATE A COMPONENT FOR PAGENOTFOUND and below content in html

   ```
   <div class="main">
   <h1>This page does not exist</h1>
   <div>Click here to go to main page
   <a routerLink="/employees">Home</a>
   </div>
   </div>
   ```

8. When first the page loads, show the default page. To achieve this we create an empty path as follows

   ```
   {path:'' ,redirectTo:'employees',pathMatch:'full'}
   ```

   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.

9. Create a component as follows
   ```
   ng g c profile
   ```

10. Add a route for view profile
    ```
    {path:'employees/:id', component:ProfileComponent
    ```

11. Add a button View Profile in the employees list along side edit and delete..
    ```
    <button class="btn btn-primary" (click)="viewProfile(emp.eid)">Profile</button>
    ```

12. Clicking on view should update the url with the current id of the employee whose profile we need to view. Since this will be dynamic we cannot use routerLink to

route for this component. Complete the viewProfile function in employees list component as follows:

```
viewProfile(id:number)
  {

        this.router.navigate(['employees', id])

        // Above is absolute routing. If the /employees url changes this code will
break. Hence uncomment below line and comment above line to add relative route
        //this.router.navigate([id], {relativeTo:this.route})
  }
```

13. To get access of id in the ProfileComponent, inject Router , ActivatedRoute and employee http service in the constructor, update ngOnInit to access the value of path parameter passed and the corresponding html of profile component as follows:

```
employee:Employee;
constructor(private route:ActivatedRoute, private es :EmphttpService,
    private router:Router) {
    this.employee = {eid:0,ename:'',phone:'',email:'',address:{country:''}}
 }

// localhost:4200/employees/1
  ngOnInit(): void {
    this.route.params.subscribe(data => {
      console.log(data.id)
      this.es.getEmployeeById(data.id)
      .subscribe(emp => this.employee = emp);
    })
  }

<div class="container">
<p> name : {{employee.ename}}</p>
<p> country : {{employee.address.country}}</p>
</div>
```

14. TASK : Likewise complete the code for edit component.

15. Above passes id as path parameter. Lets see how to pass query parameters. Lets add a back button in profile component that will take to employee list component. The employee list component needs to know the id of the employee whose profile was currently visited to highlight it as visited.

16. Update profile component ts and html as follows:

```
<button (click)="back(employee.eid)" style="margin-right:10px;">Back</button>
```

```
back(eid)
  {
      // this add ; in the url
      //this.router.navigate(['../',{id:eid}],{relativeTo:this.route})

      // to pass query parameter with ? syntax
      this.router.navigate(['../'],{relativeTo:this.route, queryParams:{id:eid}})
  }
```

17. Update employees list component and html to highlight the employee component for the id just selected

```
      selid:any = 0;

      ngOnInit(): void {
    //this.route.params.subscribe(params => this.selid = params.id)

this.route.queryParams.subscribe(params => this.selid = params.id)

    this.empservice.getAllEmployees()
    .subscribe(resp => {
     console.log('fetched employees')
     console.log(resp);
      this.employees = resp
    })
  }

isSelected(empid:any):boolean{
    console.log(empid === this.selid)
    return empid === this.selid;
  }

<button class="btn btn-primary" (click)="viewProfile(emp.eid)"
[class.border]="isSelected(emp.eid)
>Profile</button>

.border{

    color: green;
}
```

18. To create nested routes, create component:
    ng g c profiledetail

19. Update the routes for nested route for profile component

```
      {path:'employees/:id', component:ProfileComponent, canActivate:[AuthService],
          children:[
            {path:'detail', component:ProfiledetailComponent}
         ]},
```

20. Add below in profile component html

```
<div class="container">
<p> name : {{employee.ename}}</p>
<p> country : {{employee.address.country}}</p>
<button (click)="back(employee.eid)" style="margin-right:10px;">Back</button>
<button (click)="displayAddress()">View Address</button>

</div>
<router-outlet></router-outlet>
```

21. Add below in profile component ts

```
displayAddress()
  {
    this.router.navigate(['detail'],{relativeTo:this.route})
  }
```

22. Update the profile detail component and html to read the id of current employee from parent component in the url and access to display the details:

```
constructor(private route:ActivatedRoute) { }
eid:number = 0;
  ngOnInit(): void {
    this.route.parent?.params.subscribe(param => this.eid = parseInt(param.id))
  }

<div class="container" style="margin-top: 20px;">
<h1>Employee Complete Address</h1>
<p>{{eid}}</p>
</div>
```

ROUTE GUARDS

1. In traditional server side applications the application would check permissions on the server and return a 403 error page if the user didn't have permissions, or perhaps redirect them to a login/register page if they were not signed up.
   403 is a HTTP error code specifically this one means Permission Denied
   We want to have the same functionality in our client side SPA, and with Router Guards we can.
   With Router Guards we can prevent users from accessing areas that they're not allowed to access, or, we can ask them for confirmation when leaving a certain area.

2. Different types of guards based on specific use cases.
   Maybe the user must login (authenticate) first.
   Perhaps the user has logged in but is not authorized to navigate to the target component.
   We might ask the user if it's OK to discard pending changes rather than save them.
   There are four different types of Guards:
   2.1. **CanActivate**
   Checks to see if a user can visit a route.

   2.2. **CanActivateChild**
        Checks to see if a user can visit a routes children.
   2.3. **CanDeactivate**
        Checks to see if a user can exit a route.
   2.4. **Resolve**
        Performs route data retrieval before route activation.
   2.5. **CanLoad**
        Checks to see if a user can route to a module that lazy loaded.

3. To creat a CanActivateGuard, execute below command from with the **service folder** and select CanActivate as the default
   ```
   ng g g auth
   ```

4. Update the constructor and canActivate method of this class as follows:
   ```
   constructor(private router:Router, private us:UserService) { }

     canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean |
   UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {

    console.log('auth guard')
       if(sessionStorage.getItem('email') != null)
       return true;
       this.router.navigate(['login'])
       return true;
     }
   ```

5. Lets guard the profile component from unauthorized access. Update routes as follows:

   ```
   {path:'employees/:id', component:ProfileComponent, canActivate:[AuthService],
       children:[
         {path:'info', component:ProfileinfoComponent}
   ]},
   ```