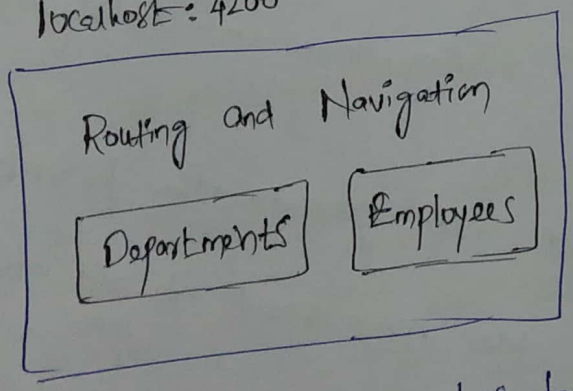
23. Routing and Navigation :-

→ Any Angular Application we build have multiple Components & each Component has its own view.

→ we need to navigate between the Components as and when user performs the action.

& we make use of Angular's Router.

Example How to navigate between two different views based on button click.



If we click on Departments, it has to display list of Departments.

If is the case with Employees

Steps for the above requirements are

(1) Generate the project with routing option using Angular CLI

(2) Generate the Department List and Employee List Component.

(3) Configure the routes. ie navigate the Routes from URL. allows us to

(4) Add buttons & use directives to Navigate to the Configured Routes

Step (1) ng new routing-demo --routing

Now we might not generate the project always with Routing option & we may want to add the Routing to our existing application as well.

→ what the Routing Application does and Replicate the same code in the existing application.

Case ①: include the base tag in the index.html file ie

src → index.html

<base href="/">

This is required so that the Application knows how to construct the URL while navigating

Case ②: in the app folder Create file ie

app-routing.module.ts

This file Contains routing Module for our Application & this is the file where we Configure the different Routes

Case (3) finally import app.routing module in app module ts.

& add it to the imports Array

Step (2) Generate the Department list and Employee list Components that we will be navigating to

On the terminal navigate inside the project folder

ng g c department-list -it -is

ng employee-list

Step (3) Configure the Routes.

we do it in the app Routing Module

In app-routing.module.ts

```
Const routes : Routes = [];
```

/

It is Strongly typed to Route from the Router Package

Here we define all possible Routes for our Application.

Each route is nothing but object.

The Object Contains a path which is reflected in the URL and the Component that has to be rendered when we are navigating to the Corresponding path.

```
Const routes : Routes = [
{ path : 'departments', Component : DepartmentListComponent },
{ path : 'employees', Component : EmployeeListComponent }
];
```

→ But by doing so we are duplicating the Import Statements in both app module & app-routing Module.

Solution is Create an Array of all the Routing Components in app-routing Module and Export it & Import in the App Module i.e

in app-routing.module.ts

Export Const routingComponents = [DepositlistComponent, EmployeelistComponents]

& in App.module.ts

Import it

& Remove the Routing in the declarations &

Any time if we want to add the Routing Components add it in the app-routing.module.ts..

→ How are the we specifying where these Components have to be displayed ie using Router Outlet directive

(in the app-component.html we have

<router-outlet></router-outlet> ie

Routed view goes here

Step④ Add the buttons to navigate from one Component to Another Component

in the app.Component.html

<nav> routerLink, routerLinkActive

<a> Department

<a> Employees

</nav>

→ To make routing possible for these anchor tags, we use two special directives from the Router package

①

1. routerLink directive which specifies the path to which we want to navigate to

② routerLinkActive, specifies one or more CSS classes that will be applied when corresponding RouterLink is active

24. Wild Card Routing

→ If the user tries to Navigate through the Route that is not Configured, the Application throws an error.

→ In the Application localhost:4200/test

then it throws an error

→ A better way to handle the Invalid URL is using Wild Card Routes

By Making use of Wild Card Routes, we can navigate the user to 404 page Not found Component if the url doesnot Match to any of the configured Routes

Step① Create PageNotFoundComponent

Step② Add the WildCardRouting

In the App Routing Module, at the end in the Routes, add another Object ie

```
Const routes : Routes = [
  { path : 'departments', Component : DepartmentListComponent },
  { path : 'employees', Component : EmployeeListComponent },
  { path : '**', Component : PageNotFoundComponent }
];
```

and also add it to the list of Components

```
Export Class AppRoutingModule { }

Export Const routingComponents =
  [ DepartmentListComponent, EmployeeListComponent,
    PageNotFoundComponent ]
```

Note: WildCard Route should be the last Point in the Configuration

The problem with wildcard route is ie after Configuring wild card Route, localhost:4200; then navigation wont work ie because url doesnot match any of the Configured Routes.

Here Application needs a default Route

{ path: '' Component: EmployeeList Component }

ie we are now Handling empty path scenario

But the Prefered Solution is to Redirect the Route to the Existing department Route ie

instead of Specifying the Component, we Specify the path it has to Redirect to.

for that we make use of redirectTo Attribute

{ path : '', redirectTo : '/departments', pathMatch : 'prefix' }

(or)

pathMatch : 'full' }

"It won't work for us"

↙

All the Routes will work as expected

In case of Prefix

→ If in case on any of the link, Routing won't work because Any url will be treated as empty & Redirects to the Department list is configured in the Wild Card Routing.

25. Route Parameters

Scenario:

In the departmentList Component we will have list of Departments

Department List

1) Angular
2) Node
3) MongoDB
4) Ruby
5) BootStrap

Department Detail

→ If we click on Department Navigate to the Department Detail Component passing id of the department as Route Parameter.

→ In the Department detail Component we will read the id parameter and display the id in the Department Detail Component.

In the departmentList Component

```
departments = [
  { "id": 1, "name": "Angular" },
  ---
  { "id": 5, "name": "Bootstrap" }
]
```

In the View we use

```
<ul>
  <li *ngFor="let department of departments">
    <span class="badge">{{department.id}}</span>
      {{department.name}}
  </li>
</ul>
```

→ Here if we click on any of the department it should Route to Department Detail Component by the id along with id.

Create Department Detail List Component and

→ Once it is Created, Add the Component in the App Routing Module

Step 1 :- in the App Routing Module.ts file, in the Constant add the new Object

{ path : 'departments/:id' Component : DepartmentDetailComponent }

Step 2 Navigate to this Route when we Click on department 'ie

<li (click) = 'onSelect(department)'>

--

// To navigate from Code We make use of Router Service

onSelect (department)

{

this.router.navigate

}

Constructor (private router : Router)

{

}

The argument to the navigate() is link parameters array i.e it needs all the information to construct the URL

i.e Pass two parameters to the array.

navigate (['/departments', department.id])

→ now the URL will navigate to Department Detail Component & in the URL 'id' is displayed

Step 8 To display the 'id' in the Department Detail Component, and read the 'id', we make use of Activated Route Service

In the Department Detail Component

Import the ActivatedRoute from angular/router

In the Constructor

```
Constructor(private route : ActivatedRoute)
{
}

ngOnInit()
{
 let id = parseInt(this.route.snapshot.paramMap.get('id'));
 this.departmentId = id;
}
```

ie from the current Route snapshot get the 'id parameter from the URL & assign it to the local variable 'id

```
public departmentId;
```

In the View

```
<h3> Department with id {{departmentId}} </h3>
```

26. ParamMap Observable

In the last Session we used the Snapshot of the Activated Route to read the Route parameter.

But this Approach has a drawback

→ In this last Scenario we always navigate from Department list Component to Department Detail Component.

Example lets add the Previous and Next button in the Department Detail Component

to go back and forth between different Departments

→ In the Component

```
<a (click)="goPrevious()">Previous</a>
<a (onClick)="goNext()">Next</a>
```

→ To Navigate from Code, we make use of Router Component

ie Constructor (private router : Router)

{

}

goPrevious()

{

let previousId = this.previousId - 1;

this.router.navigate(['/department', previousId]);

}

}

// is with goNext

→ After Here if we click on Next, the url will be updated but the view did not update

This is the draw back of Snapshot Approach ie when we are navigating from one Component and back to the same Component, Snapshot approach will not work.

Reason is Angular will check the Component is loaded or not

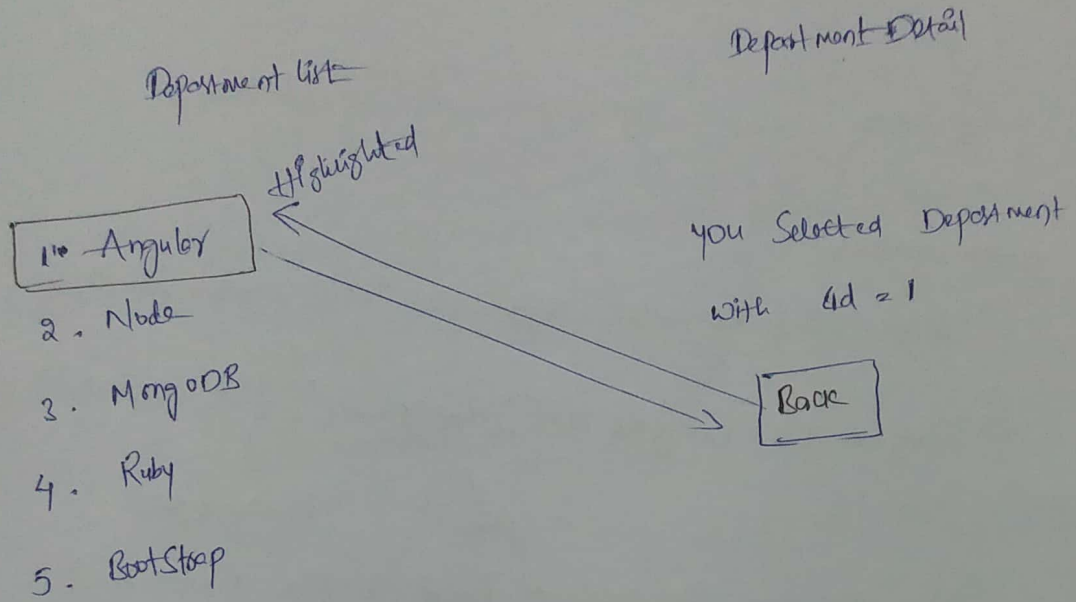
ie Here ngonit() wont be called

To overcome this we use paramMap observable.

```
ngonit()
{
this.route.paramMap.subscribe(params: paramMap));
                    => let id = parseInt(params.get('id'));
                       this.departmentId = id;
};
```

This returns the observable and we only get the data when we subscribe

27 : Optional Route Parameter



→ In the Department detail Component, we need to add Navigate button & when we click on back button, the previously Selected department will be highlighted.

and

example : This can be Acheived using optional route parameter

In the Department Detail Component we need to add back button, assign the handler to the button click

```
public selectedId;
<button (click)="gotoDepartment()"></button>

gotoDepartment() {
    let selectedId = this.department ? this.department.id : null;
    this.router.navigate(['/departments', {id: selectedId}]);
}
```

path — key value pair

→ once we have selected, we can navigate back to the Department list view, but this time passing it as an optional parameter in Link parameters Array.

→ Here id is optional because its existence doesn't effect our view, it can be used to apply some logic to the view.

→ Here we are going to read the optional parameter, compare with department id, if it matches then highlight the selected department.

→ To read the optional parameter inject the Activated Route & inject it

in the ngOnInit()

```
{
this.route.paramMap.subscribe(params: ParamMap) => {
    let id = parseInt(params.get('id'));
    this.selectedId = id;
});
}
```

→ Now we have selected Id, that comes as an optional parameter

Now lets compare department id with selected id

```
isSelected(department)
{
    return department.id == this.selectedId;
}
```

returns true if department id is equal to selected id.

[class.selected] = "isSelected(department)"

→ optional parameter doesnot need a place holder while Configuring the Route

→ we can pass any no of optional parameters in the link Array

28. Relative Navigation

→ So far in the Application, we are using absolute paths for Navigation, Can be identified by using /

→ In the Department list Component we are using '/departments' to navigate to the DepartmentDetail Component.

→ In the Department Detail Component, we are using /departments to navigate back to the Department List Component

→ When we use absolute path, then there will be small disadvantage is lack of flexibility using Routes.

→ If we want to Change the Url in the Route Module, we need to change in the app-routing Module.. & in its effected Components.

To add a little more flexibility to the routing module, we use Relative Navigation.

Example

```
onSelect(department)
{
this.router.navigate([department.id], {relativeTo: this.route});
}
```

→ In the future if we want to change the url, change in app-routing module, then it works fine.

29: Child Routes

→ In an Angular Applications, some routes may only be viewed within other routes, In such Scenario we create child routes.

→ Example we have Department Detail Component & in that we want to show Overview & Contact detail Component.

→ Overview & Contact info are features of each Department & these should be viewable only when we navigate to Department Detail Component. & Can be achieved by using Child Routes

→ generate two new Components ie overview and Contactinfo Component

In app Routing Module, change in DepartmentDetail Component

{

path : 'departments/:id'

Component : DepartmentDetailComponent,

children : [

{ path : 'overview', Component : DepartmentOverviewComponent },

{ path : 'contact', Component : DepartmentContactComponent }

]

→ add them to the Routing Components array

and

→ where exactly we need to display the child Components is specified by using <router-outlet>

→ we use buttons to Navigate

<p>

<button (click) = "showOverview()" > ShowOverview </button>

<button (click) = "showContact()" > ShowContact </button>

→ showOverview() {

this.router.navigate(['overview'], {relativeTo: this.route});

}

my showContact() {

}