**Section07 Routing in Angular**

**Lesson01 Apply Angular Routing**

**Notes:-**

**1-anuglar use the routing to replace component instead of another component with single page application**

**(The router link is special directive capture the click on hyperlink and then find the match path and then call the routing target component we want to apply)**

**(The router outlet is also special directive which is used to place the target component into the selector placed)**

**(Using of href to apply routing cause reload the whole page which is not the best way, so we use the router link directive instead of using href)**

**Steps:-**

**1-on the app.module.ts we set the following code as below**

**const appRoutes: Routes = [**

**{path:'',component:HomeComponent},**

**{path:'users',component:UsersComponent},**

**{path:'servers',component:ServerComponent},];**

**@NgModule({**

**declarations: [],**

**imports: [**

**RouterModule.forRoot(appRoutes)**

**],**

**providers: [ServersService],**

**bootstrap: [AppComponent]})**

**export class AppModule { }**

**2-on the app.component.html we set the following code as below**

**<div class="container">**

**<div class="row">**

**<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">**

**<ul class="nav nav-tabs">**

**<li role="presentation" class="active"><a routerLink="/" >Home</a></li>**

**//this is absolute path and works only on absolute path**

**<li role="presentation"><a routerLink="/servers">Servers</a></li>**

**//this is relative path and its also working on absolute path also**

**<li role="presentation"><a [routerLink]="['users']">Users</a></li></ul>**

**</div></div>**

**<div class="row">**

**<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">**

**<!-- this is special directive which is responsible to load the target component -->**

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

**</div></div></div>**

**Lesson02 Understanding Navigation Path**

**Notes:-**

**1-[routeLink] attribute works on both absolute and relative paths, whereas router Link works only on absolute path**

**(Relative path appends only the users to instead of the last section after last slash)**

**(Absolute path replace the path on the base url after the domain)**

//this is relative path

<li role="presentation"><a [routerLink]="['users']">Users</a></li></ul>

**//this is also relative path which means go to the step up and replace after this like localhost:4200/Servers**

<li role="presentation"><a routerLink="../servers">Servers relative</a></li>

**//this is absolute path**

**<li role="presentation"><a [routerLink]="['/users']">Users</a></li></ul>**

**<li role="presentation"><a routerLink="/servers">Servers</a></li>**

**Lesson03 Styling Active Router Links**

**Notes:-**

**1-by default the default path is injected into the whole path links , so you can use the tow property**

**routerLinkActive=”active”: it will apply the active style instead of declare class=”active”**

**[routerLinkActiveOptions]=”{exact:true}”: it will set the active option on the target option**

**<div class="container">**

**<div class="row">**

**<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">**

**<ul class="nav nav-tabs">**

**<li role="presentation" routerLinkActive="active" [routerLinkActiveOptions]="{exact:true}">**

**<a routerLink="/">Home</a></li>**

**<li role="presentation"  routerLinkActive="active">**

**<a routerLink="/servers">Servers</a></li>**

**<li role="presentation"  routerLinkActive="active">**

**<a [routerLink]="['users']">Users</a></li>**

**</ul>**

**</div></div>**

**<div class="row">**

**<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">**

**<!-- this is special directive which is responsible to load the target component -->**

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

**</div></div></div>**

**Lesson04 Navigating Programmatically**

**Notes:-**

**1-we can apply routing on the component level as below**

**<h4>Welcome to Server Manager 4.0</h4>**

**<p>Manage your Servers and Users.</p>**

**<button (click)="onLoadServers()" class="btn btn-primary">Routing</button>**

**//on the code behind we inject the Router on the constructor as below**

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

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

**@Component({**

**selector: 'app-home',**

**templateUrl: './home.component.html',**

**styleUrls: ['./home.component.css']})**

**export class HomeComponent implements OnInit {**

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

**ngOnInit() {}**

**onLoadServers(){**

**//complex calculation**

**//we apply relative path navigation as below**

**this.router.navigate(['servers']);}}**

**Lesson05 using relative path in programing navigation**

**Notes:-**

**1-by default the router-link keeps track of the last navigation path , so the default relative path will make an exception as below**

**<button class="btn btn-primary" [routerLink]="['users']">Navigate Relative User Page</button>**

**So either use the absolute path or using ../users to go backstep to the parent section**

**<button class="btn btn-primary" [routerLink]="['/users']">Navigate Absolute User Page</button>**

**<button class="btn btn-primary" [routerLink]="['../users']">Navigate Relative User Page</button>**

**2-by using Router on the code behind its working fine in absolute and relative path because it does not keep track of last navigation path**

**(By default it will assign to absolute path which is root path localhost:4200/)**

**<button class="btn btn-primary" (click)="onReload()">Reload Page</button>**

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

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

**import { ServersService } from '../../services/servers-service.service';**

**@Component({**

**selector: 'app-servers',**

**templateUrl: './servers.component.html',**

**styleUrls: ['./servers.component.css']})**

**export class ServersComponent implements OnInit {**

**public servers: {id: number, name: string, status: string}[] = [];**

**constructor(private serversService: ServersService,private router:Router) { }**

**ngOnInit() {this.servers = this.serversService.getServers();}**

**onReload(){**

**this.router.navigate(['servers']);}}**

**(We can assign relative path to which relateiveTo target route)**

**(it will make error because it will append localhost:4200/users/servers**

<button class="btn btn-primary" (click)="onReload()">Reload Page</button>

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';

import { ServersService } from '../../services/servers-service.service';

@Component({

selector: 'app-servers',

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

styleUrls: ['./servers.component.css']})

export class ServersComponent implements OnInit {

public servers: {id: number, name: string, status: string}[] = [];

constructor(private serversService: ServersService,private router:Router,private route:ActivatedRoute) { }

ngOnInit() {

this.servers = this.serversService.getServers();}

onReload(){

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

**Lesson06 Passing parameters to Routes**

**Notes:-**

**1-observable is feature that allow work to easily works with sync task**

**2-the route-link if you navigate to the same component it will not update the new values because the instance is not initialized**

**(So we can use activated route with it to apply a sync operation and fix the issue)**

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

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

**import { UserModel } from 'src/app/models/user';**

**import { UsersService } from 'src/app/services/users.service';**

**@Component({**

**selector: 'app-user',**

**templateUrl: './user.component.html',**

**styleUrls: ['./user.component.css'],})**

**export class UserComponent implements OnInit {**

**user: UserModel;**

**userService: UsersService;**

**userId: number;**

**//with AcitvatedRoute injected we can acccess to parameters passed**

**constructor(**

**private route: ActivatedRoute,**

**private usersService: UsersService) {**

**//when ever recieved new value update the userId and then refelect the changes on the template**

**this.route.paramMap.subscribe((params) => {**

**if (params.get('id') != null) {**

**this.userId = Number(params.get('id'));}});**

**this.userService = usersService;}**

**ngOnInit() {**

**this.user = this.userService.getUser(this.userId);}}**

**//on your template set the following code**

**<p>{{user.id}}</p>**

**<p>{{user.name}}</p>**

**<a [routerLink]="['/user/',user.id]">Click Me</a>**

**Lesson07 Important Note about Route observables**

**Notes:-**

**1-by default angular has do the unsubscribe the subscription for you when navigate outside the component you are automatically, but in another scenario on custom subscription you have to unsubscribe the subscription by yourself as below**

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

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

**import { Subscription } from 'rxjs';**

**import { UserModel } from 'src/app/models/user';**

**import { UsersService } from 'src/app/services/users.service';**

**@Component({**

**selector: 'app-user',**

**templateUrl: './user.component.html',**

**styleUrls: ['./user.component.css'],})**

**export class UserComponent implements OnInit,OnDestroy {**

**user: UserModel;**

**userService: UsersService;**

**userId: number;**

**paramsSub:Subscription;**

**//with AcitvatedRoute injected we can acccess to parameters passed**

**constructor(**

**private route: ActivatedRoute,**

**private usersService: UsersService) {**

**//when ever recieved new value update the userId and then refelect the changes on the template**

**this.paramsSub = this.route.paramMap.subscribe((params) => {**

**if (params.get('id') != null) {**

**this.userId = Number(params.get('id'));}});**

**this.userService = usersService;}**

**ngOnInit() {**

**this.user = this.userService.getUser(this.userId);}**

**//it will unsubscribe the subscription as below**

**ngOnDestroy(): void {**

**this.paramsSub.unsubscribe();}}**

**Lesson08 Different ways to pass parameters**

**Notes:-**

**1-there are three different way to pass parameters**

**A-through route Link**

**B-through query prams: which pass JavaScript object**

**C-through fragmentation: which detect the fragment name target**

**onLoadServer(id:number){**

**//complex calculation**

**this.router.navigate(['servers',id,'edit'],{queryParams:{allowEdit:'1'},fragment:'loading'});}**

**<a [routerLink]="['/servers',5,'edit']"**

**[queryParams]="{allowEdit:'1'}"**

**[fragment]="'loading'"**

**href="#" class="list-group-item" \*ngFor="let server of servers">**

**{{ server.name }}**

**</a>**

**2-the way of receiving query prams or fragments**

**//to receive the query prams or fragment as below**

**ngOnInit() {**

**//it will make issue that it will update when component is initialized not after it**

**console.log(`query params :${this.route.snapshot.queryParams}`);**

**console.log(`query params :${this.route.snapshot.fragment}`);**

**//it will reflect the changes when component initialized or after it**

**this.route.queryParams.subscribe((params) => {**

**console.log(params);});**

**//it will reflect the changes when component initialized or after it**

**this.route.fragment.subscribe((fragments) => {**

**console.log(fragments);});}**

**3-on the servers component html**

**<div class="list-group">**

**<a [routerLink]="['/servers',server.id]" [fragment]="'loading'" href="#"**

**class="list-group-item" \*ngFor="let server of servers">{{ server.name }}</a>**

**</div>**

**//on the server component code behind**

**export class ServerComponent implements OnInit {**

**server: ServerModel;**

**serverId:number= 1;**

**constructor(private serversService: ServersService,private route:ActivatedRoute){**

**//on the server component code behind we apply subscribe to receive params**

**this.route.params.subscribe((params) => {**

**if(params.id != null){this.serverId = Number(params['id']);}})}**

**ngOnInit() {**

**this.server = this.serversService.getServer(this.serverId);}}**

**Lesson09 Setting up Nested Routes**

**Notes:-**

**1-there are scenarios that the component show or hide other components by clicking on different buttons so you need to apply router-outlet on the parent component not the app.component.html also on the parent component and apply nested routes as below**

**A-on the app.module.ts we set the following nested routes**

**const appRoutes: Routes = [**

**{path:'users',component:UsersComponent,children:[**

**{path:':id',component:UserComponent},]},**

**{path:'servers',component:ServersComponent,children:[**

**{path:':id',component:ServerComponent},**

**{path:':id/edit',component:EditServerComponent},]},**

**{path:'',component:HomeComponent}];**

**B-On the servers.components.html**

**<div class="row">**

**<div class="col-xs-12 col-sm-4">**

**<div class="list-group">**

**<a [routerLink]="['/servers/',server.id]" href="#" class="list-group-item"**

**\*ngFor="let server of servers">**

**{{ server.name }}**

**</a>**

**</div></div>**

**<div class="col-xs-12 col-sm-4">**

**//we set router-outlet as below to show / hide edit , details components**

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

**</div></div>**

**C-on the users.component.html**

**<div class="row">**

**<div class="col-xs-12 col-sm-4">**

**<div class="list-group"  \*ngFor="let user of users">**

**<a href="#" class="list-group-item">{{ user.name }}</a>**

**<button class="btn btn-primary" [routerLink]="['/users/',user.id]">**

**Navigate Absolute User Page</button>**

**</div></div>**

**<div class="col-xs-12 col-sm-4">**

**//we set router-outlet as below to show / hide edit , details components**

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

**</div></div>**

**Lesson10 using Query Parameters**

**Notes:-**

**1-on the server.component.html**

**<a [routerLink]="['/servers/',server.id]" [queryParams]="{allowEdit:server.id == 3 ? '1' : '0'}" href="#" class="list-group-item"\*ngFor="let server of servers">{{ server.name }}</a>**

**2-on the server component.html**

**<h5>{{ server.name }}</h5>**

**<p>Server status is {{ server.status }}</p>**

**<button class="btn btn-primary" (click)="onEdit()">Edit Server</button>**

**//on code behind**

**ngOnInit() {**

**this.route.params.subscribe((params) => {**

**if (params.id != null) {**

**this.serverId = Number(params['id']);**

**this.server = this.serversService.getServer(this.serverId);}});**

**this.route.queryParams.subscribe((params) => {**

**this.allowEdit = params['allowEdit'];});}**

**onEdit(){**

**this.router.navigate(['/servers',this.server.id,'edit'],{queryParams:{allowEdit: this.allowEdit}});}**

**3-on edit-server.html**

**<h4 \*ngIf="!allowEdit">You are not allowed </h4>**

**<div \*ngIf="allowEdit">**

**<div  class="form-group">**

**<label for="name">Server Name</label>**

**<input**

**type="text"**

**id="name"**

**class="form-control"**

**[(ngModel)]="serverName">**

**</div>**

**<div class="form-group">**

**<label for="status">Server Status</label>**

**<select**

**id="status"**

**class="form-control"**

**[(ngModel)]="serverStatus">**

**<option value="online">Online</option>**

**<option value="offline">Offline</option>**

**</select>**

**</div>**

**<button**

**class="btn btn-primary"**

**(click)="onUpdateServer()">Update Server</button>**

**</div>**

**ngOnInit() {**

**this.server = this.serversService.getServer(1);**

**this.serverName = this.server.name;**

**this.serverStatus = this.server.status;**

**//it will make issue that it will update when component is initialized not after it**

**console.log(`query params :${this.route.snapshot.queryParams}`);**

**console.log(`query params :${this.route.snapshot.fragment}`);**

**this.route.queryParams.subscribe((params) => {**

**this.allowEdit = params['allowEdit'] == 0 ? true : false;**

**console.log(params);});**

**this.route.fragment.subscribe((fragments) => {**

**console.log(fragments);});}**

**Lesson11 Wild Card in Angular**

**Notes: -**

**1-with using wild card routing, you can set routing to the not found page when the routing not match any of routing list**

**const appRoutes: Routes = [**

**{path:'',component:HomeComponent},**

**{path:'users',component:UsersComponent,children:[**

**{path:':id',component:UserComponent},]},**

**{path:'servers',component:ServersComponent,children:[**

**{path:':id',component:ServerComponent},**

**{path:':id/edit',component:EditServerComponent},]},**

**//make sure that the wildcard routing is the last route**

**{path:'\*\*',component:NotFoundComponent}];**

**2-create new component as below: - ng g c not-found**

**Lesson12 Outsourcing routing**

**Notes: -**

**1-if you want to apply outsourcing routing in separate file you can using app-routing. Module as below**

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

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

**import { HomeComponent } from './components/home/home.component';**

**import { NotFoundComponent } from './components/not-found/not-found.component';**

**import { EditServerComponent } from './components/servers/edit-server/edit-server.component';**

**import { ServerComponent } from './components/servers/server/server.component';**

**import { ServersComponent } from './components/servers/servers.component';**

**import { UserComponent } from './components/users/user/user.component';**

**import { UsersComponent } from './components/users/users.component';**

**const routes: Routes = [**

**{path:'',component:HomeComponent},**

**{path:'users',component:UsersComponent,children:[**

**{path:':id',component:UserComponent},]},**

**{path:'servers',component:ServersComponent,children:[**

**{path:':id',component:ServerComponent},**

**{path:':id/edit',component:EditServerComponent}]},**

**//make sure that the wildcard routing is the last route**

**{path:'\*\*',component:NotFoundComponent}];**

**@NgModule({**

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

**//export means that make this class content accessiable to the outsourcing using**

**exports: [RouterModule]})**

**export class AppRoutingModule { }**

**2-on the app.module.ts we using app-routing.module.ts on import section as below**

**imports: [**

**BrowserModule,**

**AppRoutingModule,**

**FormsModule,**

**AppRoutingModule],**

**Lesson13 Introduction to Guard**

**Notes: -**

**1-with using Guard, you can apply checks of routing valid to access before loading or it**

**Steps: -**

**1-we will folder called guards and we will create two classes as below**

**Auth.service.ts**

**export class AuthService {**

**loggedIn = false**

**login(){return this.loggedIn = true;}**

**logout(){return this.loggedIn = false;}}**

**auth-guard.service.ts**

**import { Injectable } from "@angular/core";**

**import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot, UrlTree } from "@angular/router";**

**import { Observable } from "rxjs";**

**import { AuthService } from "./auth-service";**

**@Injectable()**

**export class AuthGuard implements CanActivate{**

**constructor(private authService:AuthService,private router:Router){}**

**canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):**

**boolean  | Observable<boolean> | Promise<boolean>  {**

**const promise = new Promise<boolean>(async (resolve, reject) => {**

**setTimeout(() =>{**

**var result = this.authService.logout();**

**if(!result){**

**console.log('nvaigate to home url');**

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

**resolve(result);}, 800);});**

**return promise;}}**

**3-we can target to the routing we want to apply angular guard classes as below**

**(It will apply AuthGuard to the parent and child components)**

**{path:'servers',component:ServersComponent,canActivate:[AuthGuard],children:[**

**{path:':id',component:ServerComponent},**

**{path:':id/edit',component:EditServerComponent},]},**

**4-we have to inject the both classes on the app.module.ts as below**

**providers: [ServersService,AuthGuard,AuthService],**

**Lesson14 Preventing Child (nested) routes with canActivateChild**

**Notes: -**

**1-to apply auth guard on child component only not parent you can implemtn CanActivateChild interface and using as below**

**import { Injectable } from "@angular/core";**

**import { ActivatedRouteSnapshot, CanActivate, CanActivateChild, Router, RouterStateSnapshot, UrlTree } from "@angular/router";**

**import { promise } from "protractor";**

**import { Observable } from "rxjs";**

**import { AuthService } from "./auth-service";**

**@Injectable()**

**export class AuthGuard implements CanActivate, CanActivateChild{**

**constructor(private authService:AuthService,private router:Router){}**

**canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {**

**return this.canActivate(childRoute,state);}**

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

**const promise = new Promise<boolean>(async (resolve, reject) => {**

**setTimeout(() =>{**

**var result = this.authService.logout();**

**if(!result){**

**console.log('nvaigate to home url');**

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

**resolve(result);}, 800);});**

**return promise;}}**

**//on app.module.ts we use canActivateChild instead of canActivate**

**const routes: Routes = [**

**{path:'',component:HomeComponent},**

**{path:'users',component:UsersComponent,children:[**

**{path:':id',component:UserComponent},]},**

**//,canActivate:[AuthGuard]**

**{path:'servers',component:ServersComponent,canActivateChild:[AuthGuard],children:[**

**{path:':id',component:ServerComponent},**

**{path:':id/edit',component:EditServerComponent},]},**

**//make sure that the wildcard routing is the last route**

**{path:'\*\*',component:NotFoundComponent}];**

**Lesson15 using Fake Auth Service**

**Notes: -**

**1-on the** [**home.component.html**](http://home.component.html) **page we apply the two events as below**

**<button (click)="activate()" class="btn btn-primary">Activate Route</button>**

**<button (click)="deActivate()" class="btn btn-secondary">DeActivate Route</button>**

**export class HomeComponent implements OnInit {**

**constructor(private router :Router,private authService:AuthService) { }**

**ngOnInit() {}**

**onLoadServer(id:number){**

**//complex calculation**

**this.router.navigate(['servers',id,'edit'],{queryParams:{allowEdit:'1'},fragment:'loading'});}**

**activate(){this.authService.login();}**

**deActivate(){this.authService.logout();}}**

**Lesson16 Controlling Navigation with Can Deactivate**

**Notes: -**

**1-in the previous section of angular guard , we use can activate on separate class called AUTH Guard in order to inject into the angular routing rules**

**(This check happens before navigate to the target routing and nothing logic written on the component level as special handling)**

**2-to achieve scenario like we have form and the user change input field and click to another menu option and make check permission based on business logic written on the component level we have to use CanDeactivate that execute after the CanActivate and before navigation as below**

**Steps: -**

**1-create class called can-component-deactivate.ts and set the following code as below**

**import { ActivatedRouteSnapshot, CanDeactivate, RouterStateSnapshot } from "@angular/router";**

**import { Observable } from "rxjs";**

**export interface CanComponentDeactivate {**

**canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;}**

**//in this place we can place logic check betwen componnent and this class that implement canDeactivate**

**export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {**

**canDeactivate(**

**component: CanComponentDeactivate,**

**currentRoute: ActivatedRouteSnapshot,**

**currentState: RouterStateSnapshot,**

**nextState: RouterStateSnapshot**

**): Observable<boolean> | Promise<boolean> | boolean {**

**return component.canDeactivate();}}**

**2-on the app-routing.modules.ts we set the CanDeactivateGuard on the target route we want to check as below**

**//,canActivate:[AuthGuard]**

**{path:'servers',component:ServersComponent,canActivateChild:[AuthGuard],children:[**

**{path:':id',component:ServerComponent},**

**{path:':id/edit',component:EditServerComponent,canDeactivate:[CanDeactivateGuard]}]},**

**//make sure that the wildcard routing is the last route**

**{path:'\*\*',component:NotFoundComponent}**

**3-on app.module we inject the CanDeactivateGuard as below**

**providers: [ServersService,AuthGuard,AuthService,CanDeactivateGuard],**

**4-on the edit-server.component.ts we set the following code by adding the changesSaved flag that indicate if the user click on button submit or not it will show confirm message dialog as below**

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

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

**import { iif, Observable } from 'rxjs';**

**import { CanComponentDeactivate } from 'src/app/guards/can-component-deactivate';**

**import { ServerModel } from 'src/app/models/server';**

**import { ServersService } from '../../../services/servers-service.service';**

**@Component({**

**selector: 'app-edit-server',**

**templateUrl: './edit-server.component.html',**

**styleUrls: ['./edit-server.component.css'],})**

**export class EditServerComponent implements OnInit , CanComponentDeactivate{**

**server: ServerModel;**

**serverName = '';**

**serverStatus = '';**

**allowEdit:boolean = false;**

**changesSaved:boolean = false;**

**constructor(private serversService: ServersService,private route: ActivatedRoute,**

**private router:Router) {}**

**ngOnInit() {**

**this.route.params.subscribe((params) => {**

**var id = +params['id'];**

**this.server = this.serversService.getServer(id);**

**this.serverName = this.server.name;**

**this.serverStatus = this.server.status;});**

**//it will make issue that it will update when component is initialized not after it**

**console.log(`query params :${this.route.snapshot.queryParams}`);**

**console.log(`query params :${this.route.snapshot.fragment}`);**

**this.route.queryParams.subscribe((params) => {**

**this.allowEdit = params['allowEdit'] == 0 ? true : false;**

**console.log(params);});**

**this.route.fragment.subscribe((fragments) => {**

**console.log(fragments);});}**

**onUpdateServer() {**

**this.serversService.updateServer(this.server.id, {**

**name: this.serverName,**

**status: this.serverStatus,});**

**this.changesSaved = true;**

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

**canDeactivate(): boolean | Observable<boolean> | Promise<boolean>{**

**if(!this.allowEdit){return true;}**

**if((this.serverName !== this.server.name || this.serverStatus !== this.server.status) &&**

**!this.changesSaved){**

**return confirm('Do you want to discard the changes?');}**

**else{return true;}}}**

**Lesson17 Passing static data into Angular Guard**

**Notes: -**

**We can pass static data in angular routing configuration and using special listener:-**

**1-create new component called ErrorPageComponent and set the following code**

**<h4>{{errorMessage}}</h4>**

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

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

**@Component({**

**selector: 'app-error-page',**

**templateUrl: './error-page.component.html',**

**styleUrls: ['./error-page.component.css']})**

**export class ErrorPageComponent implements OnInit {**

**errorMessage:string = "";**

**constructor(private route:ActivatedRoute) { }**

**ngOnInit(): void {**

**//this used to recieve static data in the routing defined**

**this.route.data.subscribe((params) => {**

**this.errorMessage = params["message"];})}}**

**2-on the app-routing.module.ts we set the statis data besides the routing as below**

**{path:'not-found',component:ErrorPageComponent,data:{message:'Page Not found!'}},**

**Lesson18 Resolving Dynamic data on Routing**

**Notes: -**

**1-we will be using resolver which is special service like on Activate but not control on accept / reject routing, but with controlling the rendering of the page**

**(instead of using subscribe on each component with handling incoming data as params , query params , and then call service , you can do all these steps on resolvers as below)**

**import { Injectable } from "@angular/core";**

**import { ActivatedRouteSnapshot, Resolve, RouterStateSnapshot } from "@angular/router";**

**import { Observable } from "rxjs";**

**import { ServerModel } from "../models/server";**

**import { ServersService } from "../services/servers-service.service";**

**@Injectable()**

**//we create custom resolver which return to us custom data of type ServerModel**

**export class ServerResolver implements Resolve<ServerModel> {**

**constructor(private serversService:ServersService){}**

**//this resolver return object of ServerModel so we handle the params incoming and calling service //only using it on the router**

**resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): ServerModel |**

**Observable<ServerModel> | Promise<ServerModel> {**

**return this.serversService.getServer(+route.params['id']);}}**

**2-on the app-routing.module.ts we apply using of the resolver as below**

**//,canActivate:[AuthGuard]**

**{path:'servers',component:ServersComponent,canActivateChild:[AuthGuard],children:[**

**//we apply resolver by set the format of the response format from the resolver**

**{path:':id',component:ServerComponent,resolve:{server: ServerResolver}},**

**{path:':id/edit',component:EditServerComponent,canDeactivate:[CanDeactivateGuard]}]}**

**On server.component.ts we using data route listener as below**

**//we receive the response Server Model coming from the resolver as below**

**ngOnInit() {**

**this.route.data.subscribe((data:Data) => {**

**this.server = data['server'];})}**

**3-on app.mdoule we inject the resolver as below**

**providers: [ServersService,AuthGuard,AuthService,CanDeactivateGuard,ServerResolver],**

**Lesson19 Understanding Location Strategies**

**Notes: -**

**1-in case of old browser that not accept normal html page , we can solve this by alternative way with hash on app-routing.module.ts**

**imports: [RouterModule.forRoot(routes,{useHash:true})],**