**Routing**

|  |
| --- |
| step1: Register routes in app.module.ts  //app.module.ts  import { Routes, RouterModule } from '@angular/router'  import { AppComponent } from './app.component';  import { HomeComponent } from './home/home.component';  import { UsersComponent } from './users/users.component';  **const appRoutes: Routes = [**  **{ path: '', component: HomeComponent },**  **{ path: 'users', component: UserComponent }**    **];**  @NgModule({  declarations: [  AppComponent,  HomeComponent,  UsersComponent    ],  imports: [  BrowserModule,  FormsModule,  **RouterModule.forRoot(appRoutes)**  ],  providers: [ServersService],  bootstrap: [AppComponent]  })  export class AppModule { } |

|  |
| --- |
| 2 : Add placeholder in component  **<router-outlet></router-outlet>**  //app.component.html  <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 href="#">Home</a></li>  <li role="presentation"><a href="#">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">  <router-outlet></router-outlet>  </div>  </div>  </div> |

|  |
| --- |
| 3: To avoid reloading of page , we use routerLink in <a> tag  href in anchor tag <a> will reload the page and all the storages in page will be lost.  use **ROUTERLINK** instead of href, as :-   * **it is much faster** * **and avoids reloading of page**.   <li role="presentation" class="active"><a routerLink="/">Home</a></li>  ------------  <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>  <li role="presentation"><a [routerLink]="['/users']">Users</a></li>  </ul>  </div>  </div>  <!--using routes-->  <div class="row">  <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">  <router-outlet></router-outlet>  </div>  </div>  </div> |

|  |
| --- |
| 4. **routerLinkActive : set active tab**  **routerLinkActive="active"**  <li role="presentation" **routerLinkActive="active"** **[routerLinkActiveOptions]="{exact:true**}"><a routerLink="/">Home</a></li>  **[routerLinkActiveOptions] ]="{exact:true**}">< **: It will match exact path in case of empty path like "" or "/"**  <li role="presentation"routerLinkActive="active"><a [routerLink]="['/users']">Users</a></li> |

|  |  |
| --- | --- |
| 5. **Navigating programmatically:-**  **this.route.navigate(["/servers"]);**  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 route: Router) { }  ngOnInit() {  }  onLoadServers() {  this.route.navigate(["/servers"]);  }  } | |
| **6. Passing dynamic params to route :-**  { path: 'users**/:id'**, component: UserComponent }, // dynamic route param  **: - dynamic param**  { path: 'users/:id/:name', component: UserComponent |

|  |
| --- |
| **7. fetch route params** :- snapshot  **ActivatedRoute** :-Get access to currently loaded route. It is a JavaScript object with lot of metadata  **snapshot** :- Get access to parameter in currently active route.  //app.module.ts  { path: 'users/:id/:name', component: UserComponent  import { Component, OnInit } from '@angular/core';  import { ActivatedRoute } from '@angular/router';  @Component({  selector: 'app-user',  templateUrl: './user.component.html',  styleUrls: ['./user.component.css']  })  export class UserComponent implements OnInit {  user: {id: number, name: string};  constructor(private route: ActivatedRoute) { }  ngOnInit() {  this.user = {  id: **this.route.snapshot.params['id'],**  name: this.route.snapshot.params['name']  }  }  } |

|  |  |
| --- | --- |
| 8. **Fetch route params reactively**  For dynamic route - in case route is changing dynamically , we will **subscribe** route   * snapshot is apt for first initialization but for subsequent changes **params** should be used. Params are observables * observables are a feature added by some 3 rd party package, not by Angular which allows you to easily work with asynchronous tasks. * So an observable is an easy way to subscribe to some event which might happen in the future, to execute some code   when it happens without having to wait for it now.   * Params is such an observable   Syntax:-  this.route.**params**  .**subscribe**(  (params: Params) => {  this.user.id = params['id'];  this.user.name = params['name'];  }  ); | |
| 9. Query params and fragments  <a  [routerLink]="['/servers', server.id]"  **[queryParams]**="{allowEdit: server.id === 3 ? '1' : '0'}"  **fragment**="loading"  href="#"  class="list-group-item"  \*ngFor="let server of servers">  **{{** server.name **}}**  </a> | |

|  |
| --- |
| 10. Child/Nested routes  //app.module.ts   * **Children** property holds all nested routes   const appRoutes: Routes = [  { path: servers, component: ServersComponent, **children**: [  { path: ':id', component: ServerComponent },  { path: ':id/edit', component: EditServerComponent }  ] }  ];   * Child routes need separate outlet, they should be loaded nested * Add <router-outlet></router-outlet> in parent component   C |

|  |
| --- |
| Preserve query parameters this.router.navigate(['edit'], {relativeTo: this.route, **queryParamsHandling**: **'preserve'**}); |

|  |
| --- |
| Redirecing and wild card routes { path: 'not-found', component: ErrorPageComponent, data: {message: 'Page not found!'} },  { path: '\*\*', **redirectTo**: '/not-found' }   * Routes are parsed from top to bottom * Generic routes like '\*\*' should be placed in last |
| Redirection Path Matching By default, Angular matches paths by prefix. That means, that the following route will match both /recipes  and just /  { path: '', redirectTo: '/somewhere-else' }  Actually, Angular will give you an error here, because that's a common gotcha: This route will now **ALWAYS** redirect you! Why?  Since the default matching strategy is "prefix" , Angular checks if the path you entered in the URL does **start with the path** specified in the route. Of course every path starts with ''  (Important: That's no whitespace, it's simply "nothing").  To fix this behavior, you need to change the matching strategy to "full" :  { path: '', redirectTo: '/somewhere-else'**, pathMatch: 'full'** }  Now, you only get redirected, if the full path is ''  (so only if you got NO other content in your path in this example). |

|  |  |
| --- | --- |
| 11. Route Guards   * Functionality, logic, code which is executed before a route is loaded or once you want to leave a route * A feature built into Angular router, running some code before the component is called.   Protecting routes with **canActivate**  **Step1. Add auth-guard.service.ts file:-**  **//auth-guard.service.ts**  @Injectable()  export class AuthGuard implements **CanActivate**{  constructor(private authService: AuthService, private router: Router) {}  canActivate(route: ActivatedRouteSnapshot,  state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {  return this.authService.isAuthenticated()  .then(  (authenticated: boolean) => {  if (authenticated) {  return true;  } else {  this.router.navigate(['/']);  }  }  );  }  **Step2: Define which route needs to be protected by guard:-**  **//app.routing.module.ts**   * Add **canActivate** property in route paths:-   path: 'servers', **canActivate: [AuthGuard]**, component: ServersComponent Protecting child routes(Nested) with canActivateChild Guard  * Needs to implement **CanActivateChild interface** and define canActivateChild Method * Since, it has same logic as canActivate method, we can simply call it   **//1. auth-guard.service.ts**  **canActivateChild**(route: ActivatedRouteSnapshot,  state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean  {  return this.canActivate(route, state);  }  **//2. app.routing.module.ts**  path: 'servers', **canActivateChild: [AuthGuard]**, component: ServersComponent, children: [  { path: ':id', component: ServerComponent, },  { path: ':id/edit', component: EditServerComponent} Protecting from accidentally navigating away :- canDeactivate Guard 1. Add file: // **can-deactivate-guard.service.ts**  import { Observable } from 'rxjs/Observable';  import { CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';  export interface CanComponentDeactivate {  canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;  }  export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {  **canDeactivate**(component: CanComponentDeactivate,  currentRoute: ActivatedRouteSnapshot,  currentState: RouterStateSnapshot,  nextState?: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {  return component.canDeactivate();  }  }  2. Add canDeactivae property in route config  **//2. app.routing.module.ts**  { path: ':id/edit', component: EditServerComponent, **canDeactivate: [CanDeactivateGuard]** } | |
| Passing static data to route  * Use **Data** property in route config   { path: 'not-found', component: ErrorPageComponent, **data**: {message: 'Page not found!'} },  To fetch data message:-  this.errorMessage = this.route.snapshot.**data**['message']; | | |
| Dynamic data with RESOLVE Guard Resolver:- allow to run some code before route is rendered  Will always render component in the end put it will do some pre-loading like it will fetch some data the component will then need later on  Step1. Add resolver.ts file, in which implement RESOLVE interface  **//1 resolver.module.ts**  import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';  import { Observable } from 'rxjs/Observable';  import { Injectable } from '@angular/core';  import { ServersService } from '../servers.service';  interface Server {  id: number;  name: string;  status: string;  }  @Injectable()  export class **ServerResolver** implements **Resolve**<Server> {  constructor(private serversService: ServersService) {}  **resolve**(route**: ActivatedRouteSnapshot, state: RouterStateSnapshot**): Observable<Server> | Promise<Server> | Server {  return this.serversService.getServer(+route.params['id']);  }  }  **//2. app.routing.module.ts**  { path: ':id', component: ServerComponent, **resolve: {server: ServerResolver}** },   * This **ServerResolver will return a object which will be stored in server variable**   **//3. use this data in your component**  ngOnInit() {  this.route.**data**  .subscribe(  (data: Data) => {  this.server = data['**server'**];  }  ); | | | |
| Summary (Route Guards) :-  1. **CanActivate** 2. **CanActivateChild** 3. **CanDeactivate** 4. **Resolve** | | | | |