**Basic Commands**

1. ng new app-one : Create new project
2. ng serve : Start project
3. ng g c : create component

**Data binding**

Output data from TS -> HTML :

**String interpolation** {{var\_name}} -> Only works innerHTML

**Property Binding** [property]=”data” -> Change properties/attributes of HTML element

[disabled]=”variable”

React to user event from HTML -> TS :

**Event binding** (event)=”expression”

(click)=”anyMethod()”

(click)=”anyMethod(‘any param’)”

(click)=”anyMethod(‘any param’ , $event)”

(click)=”anyMethod($event)”

In ts file -> anyMethod(event:any){

Console.log(event); // here you will see what type of event

// Change the any to MouseEvent in param

const clicked = event.target as HTMLElement;

this.whereClicked = clicked.innerText;

}

**Two way binding**: [(ngModel)]=”data”

![Graphical user interface, application

Description automatically generated]()

**Find what type of element:**

console.dir(document.querySelector(‘.btn’));

scroll down to \_\_proto

you will see then. EG HTMLInputElement , HTMLButtonElement

**Find Properties and Events of an element**:

* In chrome console -> console.dir(document.querySelector(‘.btn’));
* Or ![Graphical user interface, application, Teams

  Description automatically generated]()
* Or google YOUR\_ELEMENT properties or YOUR\_ELEMENT events

<https://www.w3schools.com/jsref/dom_obj_event.asp>

**Directives**

1. **Structural directives**—change the DOM layout by adding and removing DOM elements.

\*ngIf:

<div **\*[ngIf](https://angular.io/api/common/NgIf)**="hero">This is if condition, value can be boolean</div>

With Else:

<div **\*[ngIf](https://angular.io/api/common/NgIf)**="hero; else localRef">This is if condition, value can be boolean</div>

<ng-template #localRef >

<div>this is else condition, text of #localref can be any</div>

</ ng-template>

NgFor—repeat a node for each item in a list.

<div \*[ngFor](https://angular.io/api/common/NgForOf)="let item of items; let i=index"></div>

1. **Attribute directives**-change the appearance or behavior of an element, component, or another directive.

ngStyle: Set many inline styles simultaneously and dynamically

<div [[ngStyle](https://angular.io/api/common/NgStyle)]="currentStyles">will accept an object</div>

currentStyles: {};

this.currentStyles = {

'font-style': this.canSave ? 'italic' : 'normal',

'font-weight': !this.isUnchanged ? 'bold' : 'normal',

'font-size': this.isSpecial ? '24px' : '12px' };

}

NgClass—adds and removes a set of CSS classes based on condition. Use property binding if only one class needs to be added/removed (like toggle)

Accepts an object. Key is name of the class, value is true or false.

<li [ngClass]="{

'text-success':person.country === 'UK',

'text-primary':person.country === 'USA',

'text-danger':person.country === 'HK'

}">{{ person.name }} ({{ person.country }})

</li>

**Component and Databinding**

**Component Property Binding:**

From **PARANT** to **CHILD** -> **@Input()**

**-> @Input(‘alias’)**

Good doc: https://angular.io/guide/inputs-outputs

**Component Event Binding:**

From **CHILD** to **PARENT** -> **@Output()**

**-> @Output(‘alias’)**

Good doc: <https://angular.io/guide/inputs-outputs>

**View Encapsulation**

**To change angulars default CSS rendering**

**Local Reference in template/html**

**To assign a reference to any html element and use it only in template.**

The only way to get this ref in ts file is from a method param mentioned

In html template.

<input type="text"

class="form-control"

#serverInputElement>

<button

class="btn btn-primary"

(click)="onAddServer(serverInputElement)">Add Server</button>

onAddServer(serverInputElement:HTMLInputElement) {

this.serverCreated.emit(

{name: serverInputElement.value.trim() ,

content: this.newServerContent,

}

);

<https://angular.io/guide/template-reference-variables>

**Access to the Template & DOM with @ViewChild**

**To access template and dom element in ts file**

<input

type="text"

class="form-control"

#contentInputElement>

@ViewChild('contentInputElement' , {static : true}) contentInputElement:ElementRef;

onAddServer(serverInputElement:HTMLInputElement) {

this.serverCreated.emit(

{name: serverInputElement.value.trim() ,

content: this.contentInputElement.nativeElement.value

}

);

**ng-content**

**Angular Lifecycle Hook**

Graphical user interface, text, application

Description automatically generated

**Custom Directive (Better way to access DOM)**

<p appHighlight [defaultColor]="'green'" [highlightColor]="'blue'">this is custom directive</p>

@Directive({

selector: '[appHighlight]'

})

export class HighlightDirective implements OnInit{

@Input() defaultColor:string = "transparent";

@Input() highlightColor:string = "blue";

constructor(private elRef:ElementRef, private renderer:Renderer2) { }

ngOnInit() {

this.renderer.setStyle(this.elRef.nativeElement , 'background-color' , this.defaultColor);

}

@HostListener('mouseenter') mouseEnter(event: Event){

this.renderer.setStyle(this.elRef.nativeElement , 'background-color' , this.highlightColor);

}

@HostListener('mouseleave') mouseleave(event: Event){

this.renderer.setStyle(this.elRef.nativeElement , 'background-color' , this.defaultColor);

}

}

**Model**

Create a model : ng generate class servers/server/server --type=model

Then declare props in constructor

export class Server {

constructor(public id: number, public name: string, public status: string){}

}

**Services**

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class AccountsService {

accounts = [

{

name: 'Master Account',

status: 'active'

},

{

name: 'Testaccount',

status: 'inactive'

},

{

name: 'Hidden Account',

status: 'unknown'

}

];

constructor() { }

createAccount(name:string , status:string){

this.accounts.push({name: name , status: status});

}

updateAccount(id: number , status: string){

this.accounts[id].status = status;

}

}

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

import { AccountsService } from '../services/accounts.service';

import { LoggingService } from '../services/logging.service';

@Component({

selector: 'app-new-account',

templateUrl: './new-account.component.html',

styleUrls: ['./new-account.component.css']

})

export class NewAccountComponent {

constructor(private accountsService: AccountsService){}

onCreateAccount(accountName: string, accountStatus: string) {

this.accountsService.createAccount(accountName , accountStatus);

}

}

<div class="form-group">

<label>Account Name</label>

<input

type="text"

class="form-control"

#accountName>

</div>

<div class="form-group">

<select class="form-control" #status>

<option value="active">Active</option>

<option value="inactive">Inactive</option>

<option value="hidden">Hidden</option>

</select>

</div>

<button

class="btn btn-primary"

(click)="onCreateAccount(accountName.value, status.value)">

Add Account

</button>

Another Example of passing data between components

Service :

import { EventEmitter, Injectable } from '@angular/core';

import { Recipe } from './recipe.model';

@Injectable({

providedIn: 'root'

})

export class RecipeService {

private recipes: Recipe[] = [

new Recipe('Detox Moroccan-Spiced Chickpea Glow Bowl' , 'Sweats, thick socks, and a big glow bowl loaded with Moroccan-spiced chickpeas' , 'https://pinchofyum.com/wp-content/uploads/Moroccan-Chickpea-Bowls-4.jpg'),

new Recipe('Spicy Shrimp Tacos with Garlic Cilantro Lime Slaw' , 'Spice-loaded shrimp tucked in between smashed avocado and a cabbage slaw that is heavy with a homemade creamy lime sauce' , 'https://pinchofyum.com/wp-content/uploads/Shrimp-Tacos-with-Slaw.jpg'),

new Recipe('The Best Protein Pancakes' , 'With peanut butter and chocolate chips and a little bit of maple syrup' , 'https://pinchofyum.com/wp-content/uploads/Protein-Pancakes-1-4.jpg')

];

selectedRecipe = new EventEmitter<Recipe>();

constructor() { }

getAllRecipes():Recipe[]{

return [...this.recipes];

}

}

A component that emits/triggers some change

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

import { Recipe } from '../../recipe.model';

import { RecipeService } from '../../recipe.service';

@Component({

selector: 'app-recipe-item',

templateUrl: './recipe-item.component.html',

styleUrls: ['./recipe-item.component.css']

})

export class RecipeItemComponent implements OnInit {

@Input() recipe:Recipe; // ignore this line

constructor(private recipeService: RecipeService) { }

ngOnInit(): void {

}

onRecipeClicked(){

this.recipeService.selectedRecipe.emit(this.recipe);

}

}

A component that’s listing/subscribing to any trigger/emit

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

import { Recipe } from '../recipe.model';

import { RecipeService } from '../recipe.service';

@Component({

selector: 'app-recipe-details',

templateUrl: './recipe-details.component.html',

styleUrls: ['./recipe-details.component.css']

})

export class RecipeDetailsComponent implements OnInit {

selectRecipe: Recipe;

constructor(private recipeService:RecipeService) { }

ngOnInit(): void {

this.recipeService.selectedRecipe.subscribe(

(recipe : Recipe) => {

this.selectRecipe = recipe;

}

);

}

}

**Routing**

Absolute path : If it starts with / . eg. “/users” , “/”

Relative path : Without / . eg. “users” , “./users” , “../users”

Navigation from template is always relative to current component where router link present.

Navigation from ts file: From ts file its always relative to root. You have to configure relative to which component using {relativeTo: this.route} to make it relative

First Register route in module.ts file (later we will use separate ts):

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

const appRoutes : Routes = [

{path : '' , component: HomeComponent},

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

{path : 'users/:id/:name' , component: UserComponent},

{path : 'servers' , component: ServersComponent}

];

@NgModule({

declarations: [

Add in imports array in same file to register the const route

imports: [

BrowserModule,

FormsModule,

RouterModule.forRoot(appRoutes)

],

Then let angular know where to place/render/show this route in html

<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>

Navigating through nav bar links from HTML and making active using [routerLink] and [routerLinkActive]

<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>

Navigating through TS file programmatically

<button (click)="onClick()">Go to Servers</button>

Using Absolute path. Navigate method always point to from root (.com) unless you configure it.

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() {

}

onClick(){

// Any complex steps to perform before navigation

this.router.navigate(['/servers']);

}

}

Here is how to configure. Using Relative Path. Most of the time use this to Reload page functionality

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

import { ActivatedRoute, 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 , private route: ActivatedRoute) { }

ngOnInit() {

}

onClick(){

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

}

}

Pass data from URL

Add your path in routing module

{path : 'users/:id/:name' , component: UserComponent},

Grab this param in TS file

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']

};

}

}

Output the user properties in html

<p>User with ID {{user.id}} loaded.</p>

<p>User name is {{user.name}}</p>

Now to load this component with different data passed from URL from same component, we need to take reactive approach. Observable is used here to Observe any changes in URL from same component. Because of observable we are using ngDestroy.

<p>User with ID {{user.id}} loaded.</p>

<p>User name is {{user.name}}</p>

<a [routerLink]="['/users' , 10 , 'MD']">Load MD</a>

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

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

import { Subscription } from 'rxjs';

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit , OnDestroy {

user: {id: number, name: string};

paramSubscription: Subscription;

constructor(private route: ActivatedRoute) { }

ngOnDestroy(): void {

this.paramSubscription.unsubscribe();

}

ngOnInit() {

this.user = {

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name']

};

this.paramSubscription = this.route.params.subscribe(

(params : Params) => {

this.user = {

id: params['id'],

name: params['name']

};

}

);

}

}

Query Parameter -> anything after ? is query param

Fragment -> Anything in url with #

Add query param and fragment in HTML link

<a

class="list-group-item"

\*ngFor="let server of servers , let i=index"

[routerLink]="['/servers' , i , 'edit']"

[queryParams]="{allowEdit:1 , secondParam: true}"

[fragment]="'loading'">

{{ server.name }}

</a>

This will show like this



Add query params from TS file

<button (click)="loadServer(1)">Load server 1</button>

loadServer(id:number){

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

}

Access query params and fragments from TS file

ngOnInit() {

// will only work first time this component loads

console.log(this.route.snapshot.queryParams);

console.log(this.route.snapshot.fragment);

//will work reactvly. Makesure to unsubscribe in OnDestroy

this.route.queryParams.subscribe();

this.route.fragment.subscribe();

}

Child Routes: Child/nested routes can be used to render component side by side after clicking

const appRoutes : Routes = [

{path : '' , component: HomeComponent},

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

{path : ':id/:name' , component: UserComponent},

]},

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

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

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

]}

];

<div class="row">

<div class="col-xs-12 col-sm-4">

<div class="list-group">

<a

class="list-group-item"

\*ngFor="let user of users"

[routerLink] = "['/users' , user.id , user.name]">

{{ user.name }}

</a>

</div>

</div>

<div class="col-xs-12 col-sm-4">

<router-outlet></router-outlet>

<!-- <app-user></app-user> -->

</div>

</div>

To preserve query params from intial component to next component

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

onEdit(){

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

}

Wildcard routes and redirection

{path : 'not-found' , component: PageNotFoundComponent},

{path : '\*\*' , redirectTo: "/not-found"}

Optional

export class PageNotFoundComponent implements OnInit {

constructor(private router: Router) { }

ngOnInit(): void {

setTimeout(

() => {

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

},1000

);

}

}

Outsourcing route configuration

If you already don’t have the module when creating the project, do this:

ng generate module app-routing --flat --module=app

Then in the routing file add these

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

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

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

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

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

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

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

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

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

const appRoutes : Routes = [

{path : '' , component: HomeComponent},

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

{path : ':id/:name' , component: UserComponent},

]},

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

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

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

]},

{path : 'not-found' , component: PageNotFoundComponent},

{path : '\*\*' , redirectTo: "/not-found"}

];

@NgModule({

imports: [

RouterModule.forRoot(appRoutes)

],

exports: [RouterModule]

})

export class AppRoutingModule { }

Then in the app module just import this

imports: [

BrowserModule,

FormsModule,

AppRoutingModule

],

Guards

Create a fake auth service : ng g s auth

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class AuthService {

loggedIn = false;

constructor() { }

isAutheticated(){

const promise = new Promise(

(resolve , reject) => {

setTimeout(

() => {

resolve(this.loggedIn);

}, 2000);

}

);

console.log(promise);

return promise;

}

login(){

this.loggedIn = true;

}

logout(){

this.loggedIn = false;

}

Create a auth-guard service : ng g s auth-guard

Then implement CanActivate and fix import

import { Injectable } from '@angular/core';

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

import { Observable } from 'rxjs';

import { AuthService } from './auth.service';

@Injectable({

providedIn: 'root'

})

export class AuthGuardService implements CanActivate{

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

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

return this.authService.isAutheticated().then(

(authenticated: boolean) => {

if(authenticated){

return true;

}else{

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

return false;

}

}

);

}

}

Modify routing module and add guard where you want to protect. This canActivate property will protect both parent and child routes.

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

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

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

]},

To only project child routes:

Implement CanActiveChild interface in auth-guard service and import fix

Inside the method just call the canActivate method.

import { Injectable } from '@angular/core';

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

import { Observable } from 'rxjs';

import { AuthService } from './auth.service';

@Injectable({

providedIn: 'root'

})

export class AuthGuardService implements CanActivate , CanActivateChild{

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

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

return this.authService.isAutheticated().then(

(authenticated: boolean) => {

if(authenticated){

return true;

}else{

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

return false;

}

}

);

}

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

return this.canActivate(childRoute , state);

}

}

Then in the routing module, instead of canActivate , use canActivateChild. This will protect ONLY the child NOT parent

{path : 'servers' ,

//canActivate:[AuthGuardService] ,

canActivateChild:[AuthGuardService],

component: ServersComponent , children:[

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

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

]},

canDeactivate: this will be called right before component gets destroyed and navigate away. This can be used if a user wants to navigate away without any unsaved changes.

In this example edit server component will implement canDeactivate. So if user wants to navigate away without clicking on update button it will alert for permission

First create a guard :

ng g guard deactivate --implements CanDeactivate

Then edit this file with adding an interface like below and use that interface inside the guard class. You can copy below guard

import { Injectable } from '@angular/core';

import { CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';

import { Observable } from 'rxjs';

export interface CanComponentDeactivate {

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

}

@Injectable({

providedIn: 'root'

})

export class DeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

canDeactivate(

component: CanComponentDeactivate,

currentRoute: ActivatedRouteSnapshot,

currentState: RouterStateSnapshot,

nextState?: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

return component.canDeactivate();

}

}

Then add canDeactivate to the routing module

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

Now implement CanComponentDeactivate interface in our desired component. In our case edit server component.

Here is the entire edit server component

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

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

import { Observable } from 'rxjs';

import { CanComponentDeactivate } from 'src/app/deactivate.guard';

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

@Component({

selector: 'app-edit-server',

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

styleUrls: ['./edit-server.component.css']

})

export class EditServerComponent implements OnInit , OnDestroy , CanComponentDeactivate {

server: {id: number, name: string, status: string};

serverName = '';

serverStatus = '';

allowEdit = false;

changesSaved = false;

constructor(private serversService: ServersService ,

private route:ActivatedRoute,

private router:Router) { }

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

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;

}

}

onUpdateServer() {

this.serversService.updateServer(this.server.id, {name: this.serverName, status: this.serverStatus});

this.changesSaved = true;

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

}

ngOnDestroy(): void {

console.log('destroy called');

}

ngOnInit() {

// will only work first time this component loads

this.allowEdit = this.route.snapshot.queryParams['allowEdit']==='1' ? true :false;

this.server = this.serversService.getServer(Number(this.route.snapshot.params['id']));

this.serverName = this.server.name;

this.serverStatus = this.server.status;

}

}

If you want to see the html here it is

<h1 \*ngIf="!allowEdit">You are not allowed to Edit this server</h1>

<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>

Pass static data from routing module

Routing module:

//{path : 'not-found' , component: PageNotFoundComponent},

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

Hitml:

<h2>{{errorMessage}}</h2>

Ts file:

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.errorMessage = this.route.snapshot.data['message'];

}

}

Resolver: This runs before a specific route is rendered just like canActivate. Difference between these two is , canActivate takes care of guarding the route / component whether it sould render or not, but resolver is always going to run and preload desired data (mostly http). Usually all data get/fetch logic happens here. So instead of getting the http data from component onInit method, we will pass the data from routing module.

This is what happens : canActivate -> Resolver -> URL shows in browser -> onInit

So if you need to preload some data before showing the url in browser, you will need resolver

Create a resolver : ng g resolver servers/server --skip-tests

After this change implement from Resolve<Boolean> to Resolve<Server>. Here Server is a model means this resolver will return data which is a type of Server.

Also in the return section of resolve() method, change return type from Observable<boolean> to Observable<Server> | Promise<Server> | Server

Then create a constructor and inject your service in constructor which is responsible to make the http call and get data. In this example , ServerService. This class has a syncronouse method to get a server by passing an id

Servers.service.ts file:

getServer(id: number) {

const server = this.servers.find(

(s) => {

return s.id === id;

}

);

return server;

}

Then in the resolve() method, call and return this service method and pass route param, in this case the id. Make sure to cast it to number.

Resolve file:

import { Injectable } from '@angular/core';

import {

Router, Resolve,

RouterStateSnapshot,

ActivatedRouteSnapshot

} from '@angular/router';

import { Observable, of } from 'rxjs';

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

import { Server } from './server.model';

@Injectable({

providedIn: 'root'

})

export class ServerResolver implements Resolve<Server> {

constructor(private serversService:ServersService){}

resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<Server> | Promise<Server> | Server{

return this.serversService.getServer(Number(route.params['id']));

}

}

Then add this resolver to desired component path in routing module

{path : 'servers' ,

//canActivate:[AuthGuardService] ,

canActivateChild:[AuthGuardService],

component: ServersComponent , children:[

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

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

]},

Notice here the key is server which will hold the return from the resolver. You can change the name depending on your situation

Now inside the onInit of your component you can use this data like below:

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

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

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

import { Server } from './server.model';

@Component({

selector: 'app-server',

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

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

})

export class ServerComponent implements OnInit {

server: Server;

constructor(private serversService: ServersService,

private router:Router,

private route:ActivatedRoute) { }

ngOnInit() {

this.route.data.subscribe(

(data: Data) => {

this.server = data['server'];

}

);

// const id:number = Number(this.route.snapshot.params['id']);

// this.server = this.serversService.getServer(Number(this.route.snapshot.params['id']));

// this.route.params.subscribe(

// (params :Params) => {

// this.server = this.serversService.getServer(Number(params['id']));

// }

// );

}

onEdit(){

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

}

}

Important note: Usually webservers will handle routes and gives back html pages based on requested route. In angular case this will not work because angular only has index.html file. For example, <http://something.company.com/users> , webservers will think that there is a html file called users.html, it will try to find it but is not present, so webserver will send not found. In angulars case, we will have to configure the webserver to always return index.html even for 404. We will show that in deployment section. If incase you are using old webservers where you cant configure this, you can use hash routing mode. Just add these in routing module :

@NgModule({

imports: [

RouterModule.forRoot(appRoutes , {useHash: true})

],

exports: [RouterModule]

})

export class AppRoutingModule { }

now the url should look like this using #:



But configuring the webserver is always preferred.

**Observables**

Diagram

Description automatically generated