Angular Services(9) Routing(11)

https://ide.c9.io/laczor/angular

session_9_services (session_9_services/session_9_services-start)
96 Creating A Service
100 Creating a Data service, + Service injection
101 Communicating between components with services
103-110 Course Projects Analysis **(session_9_services/prj-services-final)**
session_11_routing-start.0
114 Setup Routes
115 Navigating with RouterLinks
116. Navigation Path
117 Styling active links
118 Programatically Navigating
119.Programatically Navigating Relative Path
120-121. Adding parameter ot the URL, and fetching with snapshot
122 Reactive Route Metadata fetching
124 Passing Query Parameters + Fragments
125 Listen to QueryParams, Fragments, Parameters, with subscribe
127. Children Routes
129 Query Params Handling
130 redirecting + wildcards
session_11_routing-start.1
131 Outsourcing Route Configuration
134 Creating Guards
135 CanACtivateChild
137 Can Deactivate
138 Passing Static Data to the Route
139. Resolve Dynamic data with the resolver (fetch data before loading the component)
141Hastag server routing

96 Creating A Service

Why services are useful?

- It can centralize the datastorage, and group functions into one file which can be called from every component.
- Also if we add it to the main ts we generate only 1 instance of this service.

1. Create the service file
logging.services.ts

```
// A  simple service, which has 1 method to console log 1 line out.

export class LoggingService {
```

```
    logStatusChange (status: string){
        console.log('A server status changed, new status: ' + status);
    }
}
```

2. Import it in the **app.module.ts** and by adding it to the `providers` we can tell angular how, to provide it to all of it's components.

```
import { AppComponent } from './app.component';
import { LoggingService } from './logging.services';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
  ],
// Lecture 101 in order to inject services to other services, we have to include the instance t
o the whole app
  providers: [LoggingService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3. In the component level, import the service, (provide it if necessary) + you have to tell angular, when it builds the component you want to include this service in the component.
account.component.ts

```
import { Component,Input} from '@angular/core';
import { LoggingService } from '../logging.services';
@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css'],
```

```
    //providers: [LoggingService]            // only needed if the service is not instanciated
 on the whole app level
})
export class AccountComponent {
  constructor (private loggingService: LoggingService){}
  @Input() account: {name: string, status: string};
  @Input() id: number;

  onSetTo(status: string) {
    this.loggingService.logStatusChange(status);
  }
}
```

100 Creating a Data service, + Service injection

- So since we imported the services into the app.module.ts as providers we can use them and **inject into other service**s with `@Injectable`
- SInce we created an instance in the app.module.ts by adding it into the providers, the accounts.service **will use 1 instance.**
- This service has actually 1. an array of objects, where you can 2. add new status or 3. change existing one.
- Lastly it will use the logging service function of logging the status, since it has been injected

accounts.services.ts

```
// In order to inject 1 service to an other
// 0. Include in the providers in the app.module.ts (import service ts file as well)
// 1. import the services file
// 2. import injectable, and add decorator to the export class
// 3. Inject the service with the constructor
// 4. Since the other service has been injected to be the property of the loggingService, we ca
n access all of it's method and properties



import { LoggingService } from './logging.services';
import {Injectable,EventEmitter} from '@angular/core';
```

```
@Injectable()
export class AccountsService {
  constructor(private loggingService:LoggingService){     // 3. Inject the service with the con
structor
  }
// we are adding an event emitter variable, to the account service, which can be triggered late
r, in ever childcomponent
  statusUpdated = new EventEmitter<string>();
     accounts = [
    { name: 'Master Account',
      status: 'active'
    },
    { name: 'Testaccount',
      status: 'inactive'
    },
    { name: 'Hidden Account',
      status: 'unknown'
    }
  ];
// we are insterting a ojbect, which has name,status property with declared string type
  addAccount(name: string, status :string){
      this.accounts.push({name:name,status:status});
      this.loggingService.logStatusChange('new server has added' +status);
  }
  updateStatus(id:number,status:string){
      this.accounts[id].status = status;
      this.loggingService.logStatusChange(status);
  }
}
```

- In order to use the services, just have to include the `AccountsService` in the **constructor** + **import**ing it, and as an inserted object we can use it.

account.component.ts

```
import { Component,Input} from '@angular/core';
import { AccountsService } from '../accounts.services';

@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css'],
  // providers: [LoggingService]
})
export class AccountComponent {
  constructor (private accountsService: AccountsService){}
  @Input() account: {name: string, status: string};
  @Input() id: number;

  onSetTo(status: string) {
    this.accountsService.updateStatus(this.id,status);
    this.accountsService.statusUpdated.emit(status);
  }
}
```

101 Communicating between components with services

1. For the shared service we declare an `EventEmitter`, (**statusUpdated = new EventEmitter<string>();**), which can be used to pass around data.
accounts.services.ts

```
import { LoggingService } from './logging.services';
import {Injectable,EventEmitter} from '@angular/core';
@Injectable()
export class AccountsService {
  constructor(private loggingService:LoggingService){}
// we are adding an event emitter variable, to the account service, which can be triggered late
r, in ever childcomponent
```

```
  statusUpdated = new EventEmitter<string>();
}
```

2. We are using the services emitter object to emit data
account.component.ts

```
import { Component,Input} from '@angular/core';
import { AccountsService } from '../accounts.services';
@Component({
  selector: 'app-account',  templateUrl: './account.component.html',  styleUrls: ['./account.co
mponent.css'],
})
export class AccountComponent {
  constructor (private accountsService: AccountsService){}
  @Input() account: {name: string, status: string};
  onSetTo(status: string) {
    this.accountsService.statusUpdated.emit(status);
  }
}
```

3. We can use the component's constructor event, to start listening to that emitted event and get the data.
new-account.component.ts

```
import { Component } from '@angular/core';
import { AccountsService } from '../accounts.services';
@Component({
  selector: 'app-new-account',  templateUrl: './new-account.component.html',  styleUrls: ['./ne
w-account.component.css'],
})
export class NewAccountComponent {
  constructor ( private accountsService: AccountsService)
    {  this.accountsService.statusUpdated.subscribe(
    (status: string) => console.log("Emitted then recieved status is : "+status)
    )}
  onCreateAccount(accountName: string, accountStatus: string) {
```

```
    this.accountsService.addAccount(accountName,accountStatus);
  }
}
```

103-110 Course Projects Analysis

(prj-services-final)
shopping.list.service.ts

```
/* 0. Has an array, of ingredients
  1. getIngredients() will make a copy of the array
  2. addIngredient() will add 1 ingredient to the array
  3. addIngredients() will use the ES6 ... seperator operator to seperately add an array of ing
redients.
  4.ingredientsChanged is a property which can be emitted
  This property will emmity an array of Ingredients object models, which will be waited at the
 shopping list componenet for the looping
*/
import { Ingredient } from '../shared/ingredient.model';
import { EventEmitter } from '@angular/core';

export class ShoppingListService {
  ingredientsChanged = new EventEmitter<Ingredient[]>();
  private ingredients: Ingredient[] = [
    new Ingredient('Apples', 5),
    new Ingredient('Tomatoes', 10),
  ];
  getIngredients() {
    return this.ingredients.slice();
  }
  addIngredient(ingredient: Ingredient) {
    this.ingredients.push(ingredient);
    this.ingredientsChanged.emit(this.ingredients.slice());
```

```
  }
  addIngredients(ingredients: Ingredient[]) {
    this.ingredients.push(...ingredients);
    this.ingredientsChanged.emit(this.ingredients.slice());
  }
}
```

recipe.service.ts

```
/* 0. Using Recipe model + Ingredient Model + ShoppingListService
*  1. Have an array of recipes, with two funfcitons
*  2. getRecipes () will return a copy of the array of Recipes
*  3. addIngredientsToShoppingList () will pass the ingredients to the shopping list datastorag
e service
*  4.recipeSelected is a proeprty which will be emitted
*   recipe-item will emit it's property of recipe ---> and it will be listened (subscribed by r
ecipes.component)
*/

import {
  EventEmitter,                 //Means there will be a property which can be emitted
  Injectable                    //There is an other service injected  into this service's conts
tructor
} from '@angular/core';

import { Recipe } from './recipe.model';
import { Ingredient } from '../shared/ingredient.model';
import { ShoppingListService } from '../shopping-list/shopping-list.service';

@Injectable()
export class RecipeService {
  recipeSelected = new EventEmitter<Recipe>();
```

```
  private recipes: Recipe[] = [
    new Recipe(
      'Tasty Schnitzel',
      'A super-tasty Schnitzel - just awesome!',
      'https://upload.wikimedia.org/wikipedia/commons/7/72/Schnitzel.JPG',
      [
        new Ingredient('Meat', 1),
        new Ingredient('French Fries', 20)
      ]),
    new Recipe('Big Fat Burger',
      'What else you need to say?',
      'https://upload.wikimedia.org/wikipedia/commons/b/be/Burger_King_Angus_Bacon_%26_Cheese_Steak_Burger.jpg',
      [
        new Ingredient('Buns', 2),
        new Ingredient('Meat', 1)
      ])
  ];
  constructor(private slService: ShoppingListService) {}
  getRecipes() {
    return this.recipes.slice();
  }
  addIngredientsToShoppingList(ingredients: Ingredient[]) {
    this.slService.addIngredients(ingredients);
  }
}
```

114 Setup Routes

1. import { `Routes` , `RouterModule` } from '@angular/router';
2. Define roots ath the appRoutes (url, component to be loaded)
3. Register roots with `RouterModule` -- > **RouterModule.forRoot(appRoutes)**
4. Add the `<router-outlet>` selector to the html, so angular will know where and what to load

app.module.ts

```
import { Routes, RouterModule } from '@angular/router';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';


const appRoutes: Routes = [
    {path:'',component: HomeComponent },      // Will mean the root url + it will load the home
 component
    {path:'users',component:UsersComponent },
    {path:'servers',component:ServersComponent},
  ];


@NgModule({
  declarations: [
      PageNotFoundComponent
  ],
  imports: [
    RouterModule.forRoot(appRoutes)              // IMportant to register the routes!
  ],
  providers: [ServersService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

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

115 Navigating with RouterLinks

- routerLink='/' href="#" -- > Will navigate to the corresponding route

- [routerLink]="['/users']" href="#" -- > Will navigate to the corresponding route with more flexible options to define the route
- example. [routerLink] = "['/users',10,'Anna']" -- > will navigate --> root/users/10/Anna

app.component.html

```
<li role="presentation" class="active">
          <a routerLink='/' href="#">Home</a></li>
        <li role="presentation">
          <a routerLink='/servers' href="#">Servers</a></li>
        <li role="presentation">
          <a [routerLink]="['/users']" href="#">Users</a></li>
```

116. Navigation Path

- //relative/absolute path///
- routerLink='/url' ---> root/url --> absolute path
- routerLink='url' ---> curreURL of the component/url --> relative path
- routerLink='../url' ---> currentURL of the component-1 layer/url --> relative path, with moving 1 layer up compared to the currentURL of the component

117 Styling active links

- routerLinkActive ="active" --> will assign the active css class if the router link is activated
- [routerLinkActiveOptions]="{exact:true}"> --> will assign the active css class if the router link is perfectly aligned with the defined url

```
<li role="presentation" class="active"
            routerLinkActive ="active"
            [routerLinkActiveOptions ]="{exact:true}">
          <a routerLink='/' href="#">Home</a></li>
        <li role="presentation"
            routerLinkActive ="active">
          <a routerLink='/servers' href="#">Servers</a></li>
        <li role="presentation"
            routerLinkActive ="active">
```

```
          <a [routerLink]="['/users']" href="#">Users</a></li>
```

118 Programatically Navigating

1. add a button, with a onLoadServers function calling

home.component.html

```
<button class='btn btn-primary' (click)='onLoadServers(1)'> Load Servers</button>
```

home.component.ts

// 1. IMport the router + inject it
// 2. Using navigate function of the router, and pass the url, as an element of an array, not with string type []

```
import { Component, OnInit } from '@angular/core';
import {Router} from '@angular/router';                          //1. IMport the router


@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  constructor(private router:Router) { }                         //2. + inject it
  ngOnInit() {
  }
// IMportant: [URL] is an array of different properties,
  onLoadServers(){
    this.router.navigate(['/servers']);
  }
}
```

119. Programatically Navigating Relative Path

// 1. Importing ROuter+ ActivatedRouter, router is for, the navigate function, ActiavtedRotuer, is to tell angualar, a lot of metadata of the current component router data
// 2. using router.navigate relativeTo parameter, by parsing the current root to it
// 3. From now on the root url here is the component current url

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

servers.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ServersService } from './servers.service';                    //Custom created service
 for cross componenet communicatons
import {Router, ActivatedRoute} from '@angular/router';                //Router is to navigate,
 Activated Route is to tell metadata
@Component({
  selector: 'app-servers',
  templateUrl: './servers.component.html',
  styleUrls: ['./servers.component.css']
})
export class ServersComponent implements OnInit {
  private 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});                //Router navig
ates, this.route is telling the current url
```

```
  }
}
```

120-121. Adding parameter ot the URL, and fetching with snapshot

Will indicate the anything entered after /users/anything, the parameter identification will be param.id
For the first initilazition of the component it is fine to use the **snapshot property**, but it **is not reactive**!

app.module.ts

```
    {path:'users/:id/:name',component:UsersComponent },
```

user.component.ts

```
// 1. Importing Activated Route, since it has the snapshot.params property, which will tell us
 the used parameters. + injecting it
// 2. defining an empty user object, which properties id,name will be changed with the passed p
arameters from the URL, the with string interpolation, the properties of the component will be
 displayed.
// 3. Since the parameters might change, (asycn code), we can listen to the parameters change b
y subscribing to it with this.route.params.subscribe, and adding a callback function, to it
// so when it changes, it will pass the recieved parameters as arguments, and will update the p
roperties of the components wiht the recieved parameters
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) { }
//We are using the ActivatedRoute metadata, and fetching the querparameter values with snapshot
  ngOnInit() {
    this.user = {
      id:   this.route.snapshot.params['id'],
      name:  this.route.snapshot.params['name'],
    };
}
```

122 Reactive Route Metadata fetching

- Params is an observable which is used to contiously listen to the changes in the ActivatedRoute parameters and by subscribing to it, we can modify the component's parameters
Note: parameters subscribtion will be automatic, when the component will be destroyed, it will **clear the subscription** as well.

```
import {ActivatedRoute,Params} from '@angular/router';
export class UserComponent implements OnInit {
  user: {id: number, name: string};

  constructor(private route: ActivatedRoute) { }

//Good for first initialization
  ngOnInit() {
    this.user = {
      id:   this.route.snapshot.params['id'],
      name:  this.route.snapshot.params['name'],
    };
//Good for continuesly checking
    this.route.params.subscribe(
      (params:Params) => {
          this.user.id = params['id'];
          this.user.name = params['name'];
      }
```

```
      );
  }
}
```

124 Passing Query Parameters + Fragments

1. We are creating a new route at app-module.ts
2. Passing manually the queryparameters + fragment(s)
3. Doing the same programatcally

app.module.ts 1

```
{path:'servers/:i/edit',component:ServersComponent}
```

Servers.component.ts 2

```
<a
      [routerLink] = "['/servers', server.id]"
      [queryParams] = "{allowEdit: server.id === 3 ? '1':'0'}"
      [fragment] = "[]"
              href="#"
      </a>
```

home.component.html (adding a method)

```
<button class='btn btn-primary' (click)='onLoadServers(1)'> Load Servers</button>
```

home.component.ts 3

```
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() {
  }
// IMportant: [URL] is an array of different properties, queryParams,fragment is an insertable
 javascript object, with assignable properties
  onLoadServers(id:number){
    this.router.navigate(['/servers', id, 'edit'], {queryParams: {allowEdit: '1'}, fragment:'lo
ading'});
  }
}
```

125 Listen to QueryParams, Fragments, Parameters, with subscribe

- Listen with snapshot of activated route oor to subscribe to Params observable

```
import { Component, OnInit } from '@angular/core';
import { ServersService } from '../servers.service';                    //Used ath navigation

import { ActivatedRoute,Params } from '@angular/router';                //Listening to metadata
 with Activated Route + Params package

@Component({
  selector: 'app-edit-server',
  templateUrl: './edit-server.component.html',
  styleUrls: ['./edit-server.component.css']
})
export class EditServerComponent implements OnInit {
  server: {id: number, name: string, status: string};
  serverName = '';
```

```
  serverStatus = '';
  allowEdit = false;

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

  ngOnInit() {
//Static subscrive with snapshot
  console.log(this.route.snapshot.queryParams);
    console.log(this.route.snapshot.fragment);
//Dymamic subscrie with param
    this.route.queryParams.subscribe(
      (queryParams:Params)=>{
       this.allowEdit = queryParams['allowEdit'] === '1' ? true : false ;
      });  // + addding some callback as we have done with the user.ts module
   // internal service worker
    this.server = this.serversService.getServer(1);
    this.serverName = this.server.name;
    this.serverStatus = this.server.status;
  }

  onUpdateServer() {
    this.serversService.updateServer(this.server.id, {name: this.serverName, status: this.serve
rStatus});
  }
}
```

127. Children Routes

1. We define children routes in the app.module ts
2. We add <router-outlet> to the server.component.html (Since the childrout, will be rendered, in the parent root's component.)

1. we make `children` routes, for the servers->Serverscomponent route

Important to note, **/** is not needed, **nor the parent url**
app.module.ts

```
    {path:'servers',component:ServersComponent , children: [
        {path:':id',component:ServerComponent },
        {path:':id/edit',component:EditServerComponent },
      ]}
```

--server.component.html---
Adding a <router-outlet></router-outlet> tag, in the serverscomponent element

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

129 Query Params Handling

server.component.ts

- this.router.navigate(['edit'], {relativeTo: this.route, queryParamsHandling :'preserve'}); --->relative path + preserve the existing parameters
- this.router.navigate(['edit'], {relativeTo: this.route, queryParamsHandling :'merge'}); ---> merge with the new one, overwrite them if it is necessary

130 redirecting + wildcards

- 1. Creating a page-not-found component, then creating a wildcard path in the app.module.ts

2. Using `redirectTo`

-

app.module.ts

```
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' }
  ];
```

- 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" :

Note

- (((not only just recognizing the begining of the path, but the full you need to include pathMatch:'full'))))

```
{ path: '', redirectTo: '/somewhere-else', pathMatch: 'full' }
```

131 Outsourcing Route Configuration

app-routing.module.ts

```
// 1. Import NgModule, to create custom module,
//+ Route, Routermodel as well //Routes - for defining routing paths, RouterModule, to import a
ll of the defined routes + exports it
//2. Import all of the created components, in the routing section
// 3. Using the @NgModule, decorator, we can import the data defined in the appRoutes (array of
 defined routes), to the Routermodule, which later can be exported to this typescript calls
// 4. Lastly it has to be exported with AppRoutingModule
```

```
import {NgModule} from '@angular/core';                                                    //
 To create custom module
import {Routes,RouterModule} from '@angular/router';                                       //R
outes - for defining routing paths, RouterModule, to import all of the defined routes + exports
 it


import {HomeComponent} from './home/home.component';
import {UsersComponent} from './users/users.component';
import {UserComponent} from './users/user/user.component';
import {ServersComponent} from './servers/servers.component';
import {ServerComponent} from './servers/server/server.component';
import {EditServerComponent} from './servers/edit-server/edit-server.component';
import {PageNotFoundComponent} from './page-not-found/page-not-found.component';
import {AuthGuard} from './auth-guard.service';
import {CanDeactivateGuard} from './servers/edit-server/can-deactivate-guard.service';
import {ErrorPageComponent} from './error-page/error-page.component';
import {ServerResolver} from './servers/server/server-resolver.service';


const appRoutes: Routes = [
    {path:'',component: HomeComponent },
    {path:'users',component:UsersComponent, children: [
        {path:':id/:name',component:UserComponent }


      ] },
    {path:'servers',
    // canActivate: [AuthGuard],
    canActivateChild:[AuthGuard],
    component:ServersComponent,
    children: [
        {path:':id',component:ServerComponent, resolve: {server: ServerResolver}},
        {path:':id/edit',component:EditServerComponent, canDeactivate:[CanDeactivateGuard] },
      ]},
```

```
    // {path: 'not-found', component:PageNotFoundComponent },
    {path: 'not-found', component:ErrorPageComponent, data: {message:"Page-not Found!"} },
    {path: '**', redirectTo:'/not-found' }
  ];
@NgModule({
    imports:[
        RouterModule.forRoot(appRoutes)
        ],
    exports:[RouterModule]
})
export class AppRoutingModule{


}
```

// 5. THis custom created module has to be imported in the app.module.ts file, @NgModule imőprts section
// 6. since we have declared, previously the components of the angular, we do not have to declared them again in this module.

app.module.ts

```
import {AppRoutingModule} from './app-routing.module';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    AppRoutingModule              // IMportant to register the routes!
  ],
```

134 Creating Guards

1. Creating an auth.service file, to return a boolean property with simulating the server runtime

2. Creating the auth-guard service file, which is an exportable class of service.
3. Inserting into the app.module.ts so the services can be used at app level
4. Importing the AuthGuard to the routing ts file, and using the built in function of router

1.**auth.service.ts**
// This is a created, service file, which main purpose, is to determine & return it's only property, the user is authenticated or not.
// 1. isAuthenticated funciton is a but special, it is actually creating a new instance of a Promise, which will save the state of the function, if it is succeded, or not.
// So in this example what we are doing, is to wait 0.8 seconds and resolve, return the loggedIn property of this AuthService app (mimicing http request)

```
export class AuthService {
  loggedIn = false;

  isAuthenticated() {
    const promise = new Promise(
      (resolve, reject) => {
        setTimeout(() => {
          resolve(this.loggedIn);
        }, 800);
      }
    );
    return promise;
  }
  login() {
    this.loggedIn = true;
  }
  logout() {
    this.loggedIn = false;
  }
}
```

2.. **auth-guard.ts**

```
// Actually, this service implements the CanActivate interface, which has the following require
ment:
```

```
// 1. has to have a (route, state )argument whith the defined type
// 2. has to return a Observable,Promise,Boolan value
// 3. Since we are returning a promise with this code: this.authService.isAuthenticated(), the
 property loggedIn of boolean is wrapped in tht promise
// 4. Then the recieved value from this code, will be evaluted, and
//  Either a boolean value will be returned, or  with router.navigation, we will move to a next
 page, not for the requested one
// 5. Key dependecies, Authservice, which will return the value, has to be injectable, all of t
he argument types, implementation interface, has ti be imported as well

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

import {Observable} from 'rxjs/Observable';
import {Injectable} from '@angular/core';
import {AuthService} from './auth.service';

@Injectable()
export class AuthGuard implements CanActivate,CanActivateChild {

    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(['/']); // takes you to the root page.
                  return false;
                }
              }
            );
          }
}
```

3. **app.module.ts**

```
import {AuthGuard} from './auth-guard.service'
import {AuthService} from './auth.service'


 providers: [ AuthService, AuthGuard],
```

4. **app-routing.module.ts**

```
    {path:'servers', canActivate: [AuthGuard],  component:ServersComponent,
    children: [
        {path:':id',component:ServerComponent, resolve: {server: ServerResolver}},
        {path:':id/edit',component:EditServerComponent, canDeactivate:[CanDeactivateGuard] },
      ]},
```

135 CanACtivateChild

1. basically the same structure as for the CanActivate, but **CanActivateChild will be applied to the childRoutes of the mainRoute.**
It is actually calling the main CanActivate method
2. Inserting into the Routing ts
auth-guard.ts

```
 import {
    CanActivate,
    CanActivateChild,
```

```
    ActivatedRouteSnapshot,
    RouterStateSnapshot,
    Router
} from '@angular/router';

import {Observable} from 'rxjs/Observable';
import {Injectable} from '@angular/core';
import {AuthService} from './auth.service';

@Injectable()
export class AuthGuard implements CanActivate,CanActivateChild {

    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(['/']); // takes you to the root page.
                   return false;
                 }
               }
             );
          }
    canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
                Observable<boolean> | Promise<boolean> | boolean {
                    return this.canActivate(route,state);
           }
```

```
}
```

2.**app-routing.module.ts**

```
    {path:'servers', canActivate: [AuthGuard],canActivateChild: [AuthGuard],  component:Servers
Component,
    children: [
        {path:':id',component:ServerComponent, resolve: {server: ServerResolver}},
        {path:':id/edit',component:EditServerComponent, canDeactivate:[CanDeactivateGuard] },
      ]},
```

137 Can Deactivate

1.create **can-deactivate-guard.ts**
// So here we are creating an interface, which is a contract. Every class which implements it has t o have a **canDeactivate ()**method which will return a Observable/promise/boolean, this is the only criteria
// next, we are creating a CanDeactiavedGuard, which is a class, which implements the canDeactivate generic interface, which defined type is our custom created interface
// The reason for it is to ensure, the if a component, is using the custom interface, (has a canDeactivate function, with the specific returning values), can use this guard.
// Can deactivate method, will take several arguments
// - component, is referring to the component where we are using it
// - currentRoute is the component's current route
// - CurrentState is the component's current url, queryparams,fragment etc.
// - nextState is an optional argument ?, with the next routerparameters.
// Lastly, we are calling the component's canDeactivate own function, where we are currently on.

```
import { Observable } from 'rxjs/Observable';
import { CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';


//CanDeactivate method needsparameters (component,currentRoute,currentState,NextState)


//Every class which implements it has t o have a canDeactivate ()method which will return a Obs
ervable/promise/boolean, this is the only criteria
export interface CanComponentDeactivate {
  canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;
```

```
}
export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {
  canDeactivate(component: CanComponentDeactivate,                              //The actual comp
onent which are using the applied interface
                currentRoute: ActivatedRouteSnapshot,
                currentState: RouterStateSnapshot,
                nextState?: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | bool
ean {
    return component.canDeactivate();
  }
}
```

2.import the guard service ot **app.module.ts**

```
import {CanDeactivateGuard} from './servers/edit-server/can-deactivate-guard.service';
@NgModule({
  declarations: [  ],
  providers: [CanDeactivateGuard],
})
```

3, import the guard to the **routing.ts**

```
import {CanDeactivateGuard} from './servers/edit-server/can-deactivate-guard.service';
const appRoutes: Routes = [

    {path:'servers',   canActivateChild:[AuthGuard],   component:ServersComponent,    children:
 [
        {path:':id',component:ServerComponent, resolve: {server: ServerResolver}},
        {path:':id/edit',component:EditServerComponent, canDeactivate:[CanDeactivateGuard] },
      ]},
  ];
@NgModule({
    imports:[
```

```
        RouterModule.forRoot(appRoutes)
        ],
    exports:[RouterModule]
})
export class AppRoutingModule{


}
```

4. creating the canDeactiave method in the e**dit-server.component**
// 1. Adding an interface, to this custom created class, which will force use to have a canDeactivate class, which will return a Observable,Promiise, Boolean
// 2. Writing the candeactivate function
// will return Observable,Promise,Boolean as the canComponenetDeactivated interfaces forces, check the properties for changes and returning a confrim options

// In summary, canDeactiave is a built in angular interface, which can be called in the Routes path, have several arguments, will keeptrack if the user is trying to leave the current component
// if yes, it will know all of the details about the component, and will call the canDeactivate function of the component, and with our custom interface we are forcoing the injected component to have a candeactivate function
// our custom function is not mandatory, but good practice to force them to have, so the canDeactivate angular interface could work.

```
import { Component, OnInit } from '@angular/core';
import { ServersService } from '../servers.service';
import { ActivatedRoute,Params,Router } from '@angular/router';
import { CanComponentDeactivate} from './can-deactivate-guard.service';                //íThis
 is the service what we have created
import { Observable } from 'rxjs/Observable';


@Component({
  selector: 'app-edit-server',
  templateUrl: './edit-server.component.html',
  styleUrls: ['./edit-server.component.css']
})
export class EditServerComponent implements OnInit, CanComponentDeactivate {        //For the s
ervice the interface has to be used as well
  server: {id: number, name: string, status: string};
```

```
  serverName = '';
  serverStatus = '';
  allowEdit = false;
  changesSaved = false;

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

  ngOnInit() {
    this.route.queryParams.subscribe(
      (queryParams:Params)=>{
       this.allowEdit = queryParams['allowEdit'] === '1' ? true : false ;
      });  // + addding some callback as we have done with the user.ts module

    this.server = this.serversService.getServer(+this.route.snapshot.params['id']);      // + i
s needed to convert string value from it this.route.snapshot.params['id']
    this.serverName = this.server.name;
    this.serverStatus = this.server.status;
  }

  onUpdateServer() {
    this.serversService.updateServer(this.server.id, {name: this.serverName, status: this.serve
rStatus});
    this.changesSaved = true;
    this.router.navigate(['../'], {relativeTo: this.route });
  }

// will return Observable,Promise,Boolean as the canComponenetDeactivated interfaces forces, ch
eck the properties for changes and returning a confrim options, in any case, this will terurn t
rue or false, based on the componenet's variables
  canDeactivate(): Observable<boolean> | Promise<boolean> | 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;
    }
  }
}
```

138 Passing Static Data to the Route

1. Parse some data in the **app-routing.module.ts**

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

2. Listen to it in the **ErrorPageComponent.ts**

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Data } 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() {
    // this.errorMessage = this.route.snapshot.data['message'];
```

```
    this.route.data.subscribe(
      (data: Data) => {
        this.errorMessage = data['message'];
      }
    );
  }
}
```

139. Resolve Dynamic data with the resolver (fetch data before loading the component)

1. Create the resolver service
   - Import other service + defining a custom inter that componenet which is using this service has to have a property of server
   - we are fetching the route parameters from ActivatedRouteSnapshot and passing to the service which will return data in the interface defined object **Server**

server-resolver.service.ts

```
import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import { Injectable } from '@angular/core';                          //In order to tuse other
 services
import { ServersService } from '../servers.service';                //Implementation of custom
 service for data fetching from a global instance
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> | Prom
ise<Server> | Server {
    return this.serversService.getServer(+route.params['id']);
  }
}
```

2. Register it in the **app.module.ts**

```
import {ServerResolver} from './servers/server/server-resolver.service';


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

3. Insert ot the **app-routing.ts**

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

4. Insert it into the **server.component.ts**

```
 import { Component, OnInit } from '@angular/core';
import { ServersService } from '../servers.service';
import {ActivatedRoute, Params,Router,Data } from '@angular/router';


@Component({
  selector: 'app-server',
  templateUrl: './server.component.html',
  styleUrls: ['./server.component.css']
})
export class ServerComponent implements OnInit {
  server: {id: number, name: string, status: string};


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


  ngOnInit() {
```

```
// So here what is happening, when the URL is passed with the routing, at the path level, the r
esolver function is called, which will get the server data, using the passed param id
// so the only thing what we have to do, is to continously check the the data for updates, and
 change the property of this server.
    this.route.data.subscribe(
      (data:Data)=>
      this.server = data['server']
      );
//tHIS WE ARE NOT JUST ISMPLY COLLECTION THE DATA USING SERVICES AFTER THE COMPONENET HAS BEEN
 LOADED, BUT WE ARE RECIEving ready data.
    // const id= +this.route.snapshot.params['id'];
    // this.server = this.serversService.getServer(id);
    // this.route.params.subscribe(
    //   (params:Params) => {
    //     this.server = this.serversService.getServer(+params['id']);

    //   }
      // );
  }

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

141Hastag server routing

-- app.module.ts ---

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

Will include a **root/#/url,** and the server will only load the root file, instead of the index html file. even at 404