

# ROUTE GUARDS IN ANGULAR

## Prerequisites:

Angular component, services, http service, routing and navigation

## What are Route Guards?

Angular's route guards are **interfaces** which can tell the router **whether or not it should allow navigation to a requested route**. They make this **decision by** looking for a true or false return value from a class which implements the given guard interface.

## Uses of Route Guards

- Allow access to certain parts of the application to specific users.
- Validating the route parameters before navigating to the route.
- Fetching some data before you display the component.
- To Confirm the navigational operation.
- Asking whether to save before moving away from a view.

There are five different types of guards and each of them is called in a particular sequence. The router's behavior is modified differently depending on which guard is used. The guards are:

CanActivate	Checks to see if a user can visit a route.
CanActivateChild	Checks to see if a user can visit a routes children.
CanDeactivate	Checks to see if a user can exit a route.
CanLoad	Checks to see if a user can route to a module that lazy loaded.
Resolve	Performs route data retrieval before route activation.

## CanActivate Guard

This guard decides if a route can be activated (or component gets rendered). We use this guard, when we want to check on some condition, before activating the component or showing it to the user. This allows us to cancel the navigation.

Eg. One of the use cases of this guard is to check if the user has logged in to the system. If user has not logged in, then the guard can redirect him to login page.

The CanActivate interface looks like this,

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

The route and state parameters are accessed by ActivatedRouteSnapshot and RouterStateSnapshot.

The guard must return true/false or a UrlTree. The return value can be in the form of observable or a promise or a simple boolean value.

## CanActivateChild Guard

The CanActivateChild guard is very similar to CanActivate guard. We apply this guard to the parent route. The Angular invokes this guard whenever the user tries to navigate to any of its child route.

Eg. When we are using url with request parameters like: <http://localhost:4200/users/1>

CanActivateChild guard is applied to user id 1 details page(child of Users component).

CanActivateChild Interface looks like this,

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

## CanDeactivate Guard

This guard is called whenever we navigate away from the route before the current component gets deactivated.

Eg. When user enters data in a form and leaves the page for another route without submitting, a warning pops up informing the changes done cannot be save and giving option whether to navigate away or not.

CanDeactivate Interface looks like this,

```
interface CanDeactivate<T> {  
  canDeactivate(component: T,  
    currentRoute: ActivatedRouteSnapshot,  
    currentState: RouterStateSnapshot,  
    nextState?: RouterStateSnapshot): Observable<boolean | UrlTree> |  
  Promise<boolean | UrlTree> | boolean | UrlTree  
}
```

## CanLoad Guard

The CanLoad Guard prevents the loading of the Lazy Loaded Module. We generally use this guard when we do not want to unauthorized user to navigate to any of the routes of the module and also prevents them from even seeing the source code of the module.

The Angular provides CanActivate Guard, which prevents unauthorized user from accessing the route. But it does not stop the module from being downloaded. The user can use the chrome developer console to see the source code. The CanLoad Guard prevents the module from being downloaded.

## Resolve Guard

This guard delays the activation of the route until some tasks are complete. You can use the guard to pre-fetch the data from the backend API, before activating the route.

Eg. When we are getting data from a remote repository and display in next page, if we go to next page it will be loading (which is not a good user experience).

But if we apply resolve guard to that route, unless it loads all the data it won't go to the next page and will be in the current page.

## How to create a guard?

### Step1: Create a guard (This is applicable to 4 types of guards).

**Command:** `ng generate guard guard-name`

If we use the above command, the Command Line Interface (CLI) will show options to select the type of guard we want to implement.

```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT
2: node
(node:1160) ExperimentalWarning: Conditional exports is an experimental feature. This feature could change at any time
PS C:\Users\m1060521\angular projects\route-guards-pract> ng g guard authenticate
? Which interfaces would you like to implement? (Press <space> to select, <a> to toggle all, <i> to invert selection)
>(*) CanActivate
   ( ) CanActivateChild
   ( ) CanDeactivate
   ( ) CanLoad
(node:4816) ExperimentalWarning: Conditional exports is an experimental feature. This feature could change at any time
```

(OR)

Directly mention the interface to be implemented.

**Command:** `ng generate guard guard-name --implements interface-name`

```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT
2: powershell
any time
PS C:\Users\m1060521\angular projects\route-guards-pract> ng g guard guards/Admin --implements CanActivateChild
```

After selection of the interface, 2 files will get generated.

```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT
2: node
(node:1160) ExperimentalWarning: Conditional exports is an experimental feature. This feature could change at any time
PS C:\Users\m1060521\angular projects\route-guards-pract> ng g guard authenticate
? Which interfaces would you like to implement? (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Which interfaces would you like to implement? CanActivate
CREATE src/app/authenticate.guard.spec.ts (371 bytes)
CREATE src/app/authenticate.guard.ts (465 bytes)
PS C:\Users\m1060521\angular projects\route-guards-pract>
```

The authenticate.guard.ts will look like this (with @Injectable annotation and class implementing CanActivate interface),

```
TS app.module.ts TS user.component.ts TS users.service.ts <> user.component.html TS authenticate.guard.ts X <> users 🔍 📄
route-guards-pract > src > app > TS authenticate.guard.ts > ...
1 import { Injectable } from '@angular/core';
2 import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class AuthenticateGuard implements CanActivate {
9   canActivate(
10     route: ActivatedRouteSnapshot,
11     state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
12     return true;
13   }
14 }
15
16
```

For **5<sup>th</sup> Guard : Resolve guard**, the process is little different as there is no option to create resolve guard in CLI (Command Line Interface). First, we **create a service and implement Resolve interface and override resolve()** to create resolve guard.

```
1 import { Injectable } from '@angular/core';
2 import { Resolve } from '@angular/router';
3 import { UsersService } from '../users.service';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class ResolveGuard implements Resolve<any>{
9
10   constructor(private service:UsersService){}
11
12   resolve(){
13     return this.service.getAllUsers();
14   }
15
16 }
17
```

**Step2: Register as a service in root module (This is applicable to all types of guards).**

The authenticate.guard.ts is generally a **service which implements the route guard interface**, which can be **injected like any other services**. Hence it **must be registered in the providers property of app.module**.

```
providers: [
  UsersService,
  AuthenticateGuard,
  AdminGuard,
  UnsavedChangesGuard,
  ResolveGuard
],
```

### Step3: Associate the guard with corresponding route

Now we must apply the guard for the route for which we have to protect.

```
const routes:Routes=[
  {path:"users",component:UsersComponent,
    canActivate:[AuthenticateGuard]
  },
```

The **canActivate** property is given the value with our AuthenticationGuard class.

Since canActivate property takes array of values, we can implement multiple guards before the activation of the particular route.

If it is **canActivateChild** guard,

```
{path:"user/:id",component:UserComponent,
  canActivateChild |: [AdminGuard],
  children: [
    // {path:"", redirectTo:"address", pathMatch:"full"},
    {path:"address",component:AddressComponent},
    {path:"company",component:CompanyComponent}
  ]
},
```

Parent route: user/:id

Children routes: address and company

The AdminGuard class is applied to both the children of routes.

We should be careful when applying canActivateChild guard, if any default child route for redirection is given, then the guard is applied to the parent route.

For **canDeactivate** guard, it is similar to canActivate,

```
{path:"addUser", component:AdduserComponent, canDeactivate:[UnsavedChangesGuard]],
```

For **resolve** guard,

```
{path:"users",component:UsersComponent,  
  canActivate:[AuthenticateGuard],  
  resolve:{  
    data:ResolveGuard  
  }  
},
```

Unlike the other guards, we have to use resolve keyword and value as object containing key and value pair.

Data is the key and ResolveGuard which we created is the value.

#### Step4: Implement the logic to guard the routes

Whenever we choose a route, the control flow will go to the guard component and check for permission to enter the route.

In this example, instead of writing the logic in the guards itself, we have included one more service named Authentication.service.ts

So based on the outputs from the service layer, the guards will decide whether access to particular route should be provided or not.

```
3  @Injectable({  
4    providedIn: 'root'  
5  })  
6  export class AuthenticationService {  
7  
8    constructor() { }  
9  
10   isUserLoggedIn(){  
11     // can have our logic here  
12     return true;  
13   }  
14  
15   isAdminRole(){  
16     // can have our logic here  
17     return false;  
18   }  
19 }
```

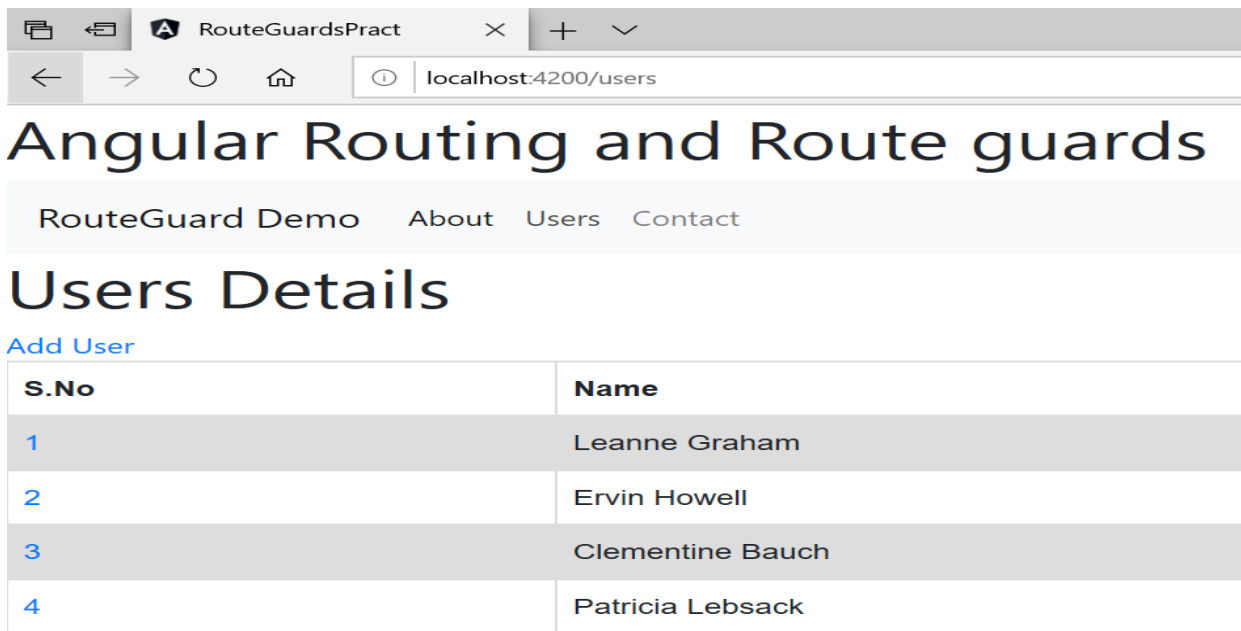
## CanActivate

Here the CanActivate guard uses isUserLoggedIn() method, to give permission to the UsersComponent

```
@Injectable({
  providedIn: 'root'
})
export class AuthenticateGuard implements CanActivate {

  constructor(private service:AuthenticationService){}
  canActivate(){
    // logic
    if(this.service.isUserLoggedIn()){
      return true;
    }
    else{
      window.alert('Permission denied')
      return false;
    }
  }
}
```

If isUserLoggedIn() returns true, there is no problem in accessing the route.



RouteGuard Demo About Users Contact

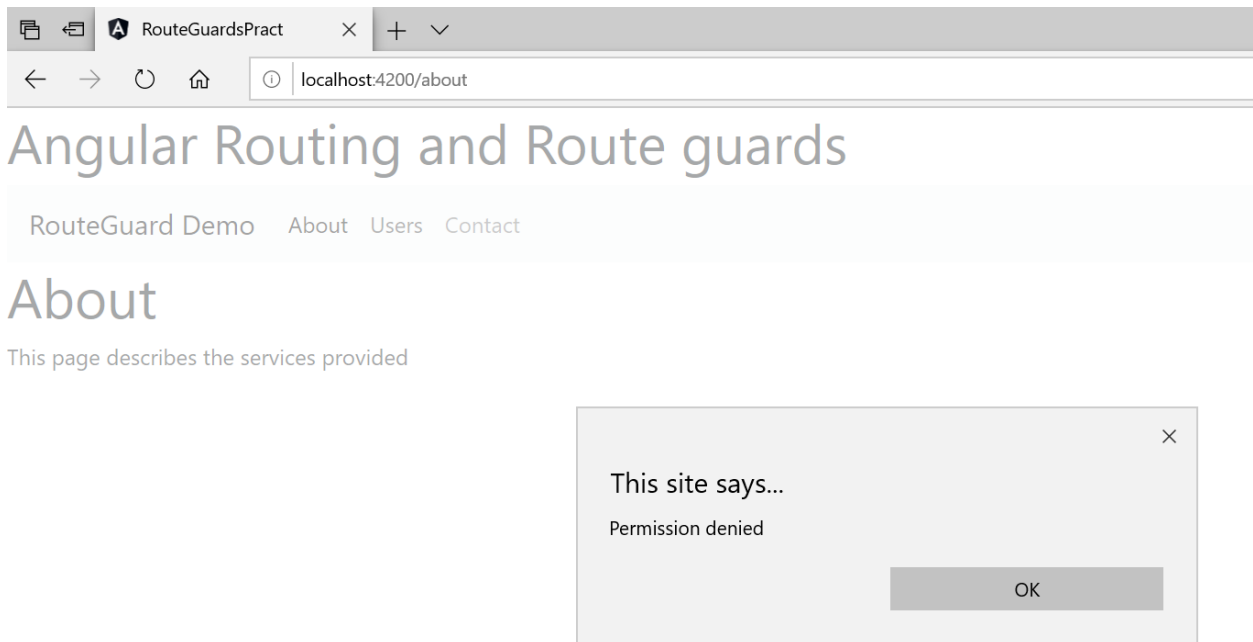
## Users Details

[Add User](#)

S.No	Name
1	Leanne Graham
2	Ervin Howell
3	Clementine Bauch
4	Patricia Lebsack



If `isUserLoggedIn()` returns false, a window alert with message “Permission denied” is thrown.

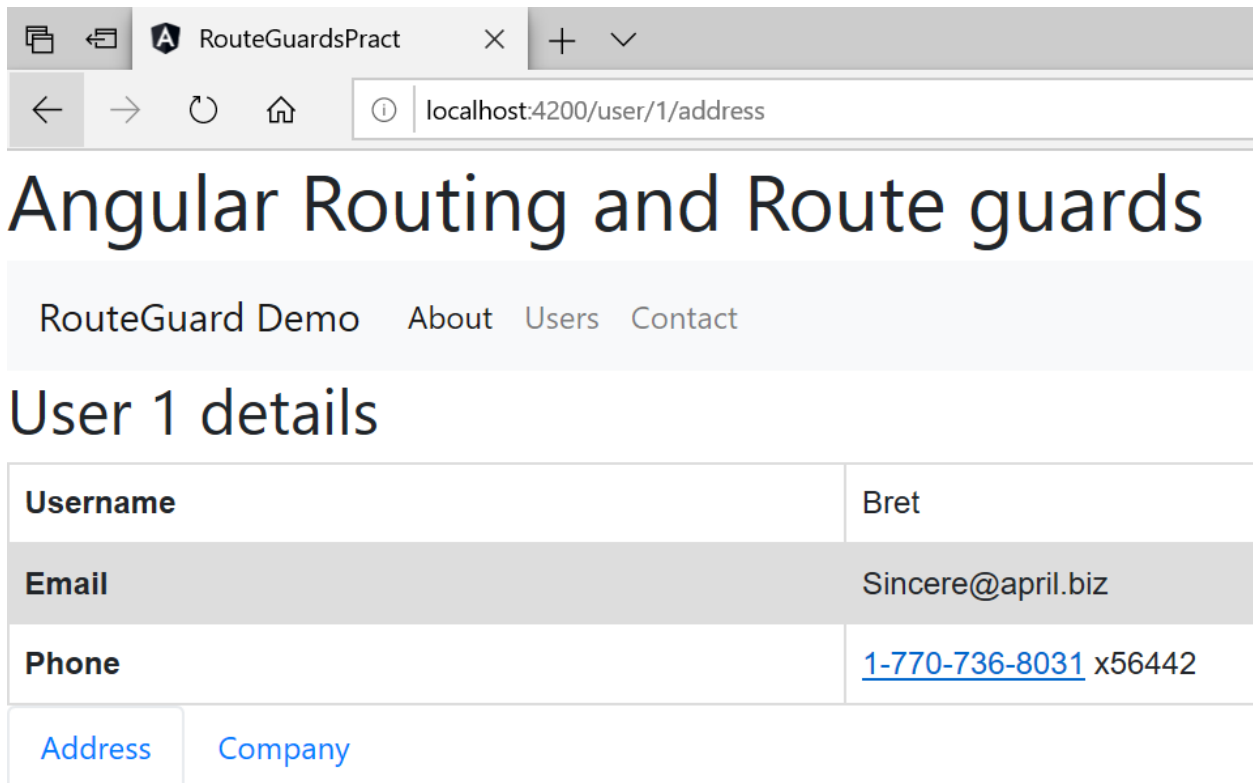


## CanActivateChild

Here the `CanActivateChild` guard uses `isAdminRole()` method, to give permission to the children of User Component,

```
6  @Injectable({
7    providedIn: 'root'
8  })
9  export class AdminGuard implements CanActivateChild {
10     constructor( private service:AuthenticationService){
11
12     }
13     canActivateChild() {
14         if(this.service.isAdminRole())
15             return true;
16         else{
17             window.alert("Only admin can view the address or company")
18             return false
19         }
20     }
21 }
22 }
```

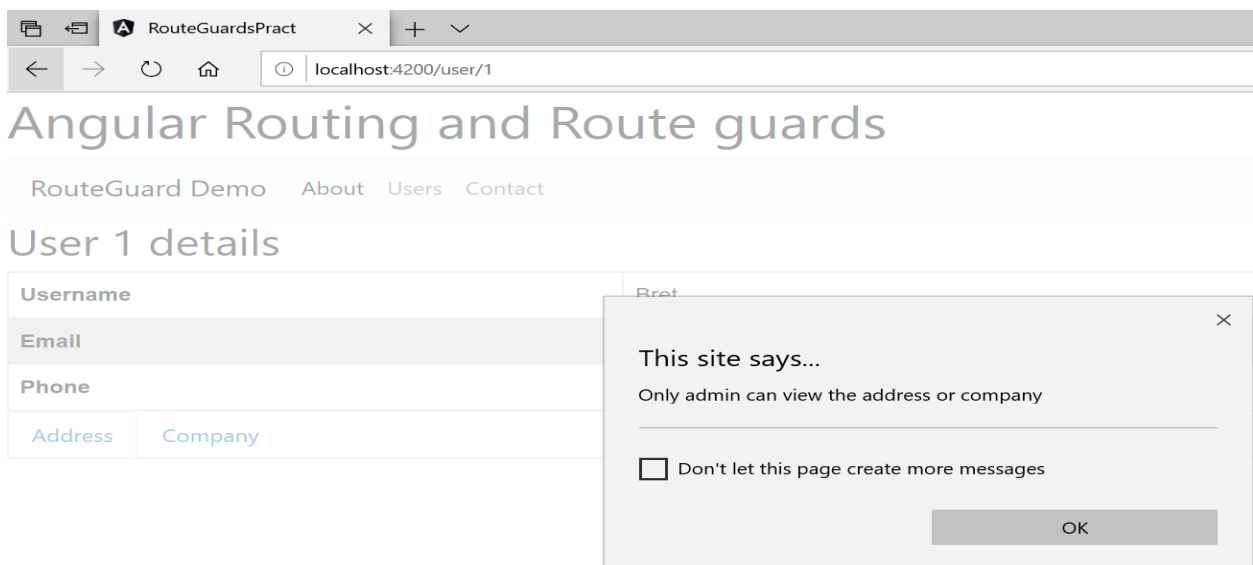
If `isAdminRole()` returns true, there is no problem in accessing the child routes address and company.



Username	Bret
Email	Sincere@april.biz
Phone	<a href="tel:1-770-736-8031">1-770-736-8031</a> x56442
Address	Company

address works!

If `isAdminRole()` returns false, a window alert with message “Only admin can view the address or company” is thrown.



Username	Bret
Email	Sincere@april.biz
Phone	<a href="tel:1-770-736-8031">1-770-736-8031</a> x56442
Address	Company

This site says...

Only admin can view the address or company

☐ Don't let this page create more messages

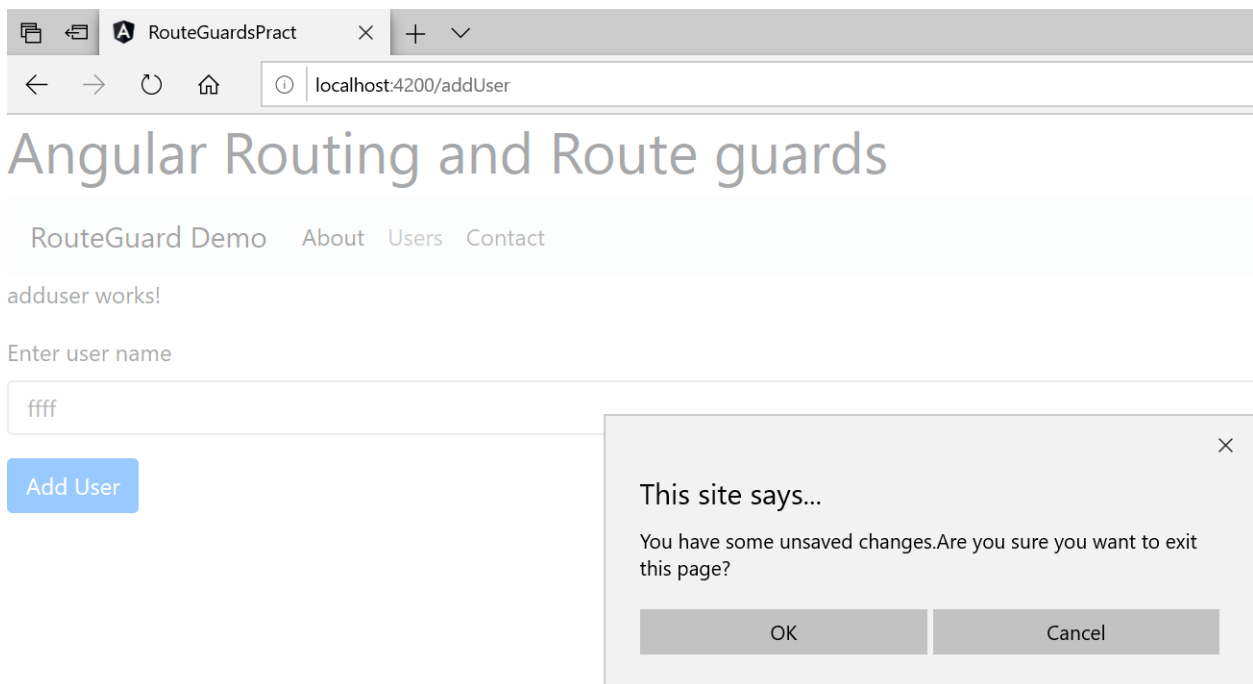
OK

## CanDeactivate

Here the CanDeactivate guard uses username.dirty property of formControl to give permission to navigate away from the addUser route.

```
6  @Injectable({
7    providedIn: 'root'
8  })
9  export class UnsavedChangesGuard implements CanDeactivate<AdduserComponent> {
10    canDeactivate(component:AdduserComponent) {
11      if(component.userName.dirty){
12        return window.confirm("You have some unsaved changes."
13          + "Are you sure you want to exit this page?");
14      }
15      return true;
16    }
17  }
18 }
19
```

When we enter data and leave without submitting form, it throws a confirming message “You have some unsaved changes. Are you sure you want to exit this page?” If we give OK, we can navigate away. If we give CANCEL, it will stay on the same page.



## Resolve

Here in the Resolve guard we are calling UsersService method to load data from a remote server. Initially this getAllUsers() method is called in Users Component. So while viewing data it directly went to display page, took some time to load the data and display it on the screen.

But now we are using resolve guard, the users page won't display until data is loaded from the remote server, thus giving a better user experience.

```
@Injectable({
  providedIn: 'root'
})
export class ResolveGuard implements Resolve<any>{

  constructor(private service:UsersService){}

  resolve(){
    return this.service.getAllUsers();
  }
}
```

Users service class:

```
11  @Injectable({
12    providedIn: 'root'
13  })
14  export class UsersService {
15    private _url="https://jsonplaceholder.typicode.com/users";
16    constructor(private http:HttpClient) { }
17
18    getAllUsers(){
19      return this.http.get(this._url)
20      .pipe(catchError(this.errorHandler));
21    }
22
23    errorHandler(error:HttpErrorResponse){
24      return throwError(error.message || "server error");
25    }
26
27    getUser(id:number){
28      return this.http.get(this._url+'/'+id)
29    }
30
31
32    getAddress(user){
33      return this.http.get(this._url+'/'+user.id+'/'+user.address)
34      .pipe(catchError(this.errorHandler));
35    }
36  }
```

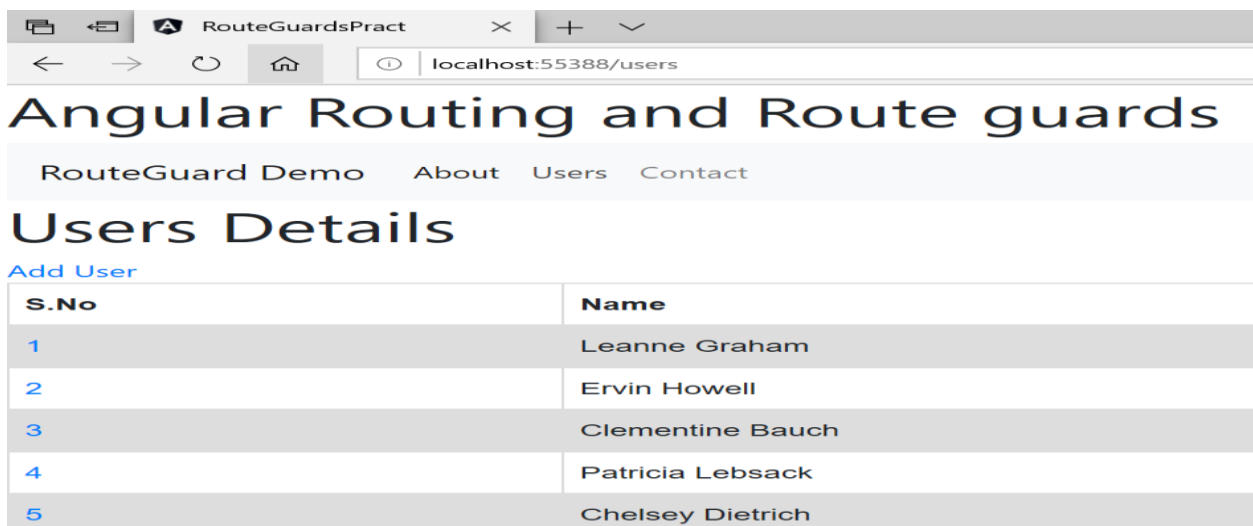
And in Users Component, we retrieve the data through snapshot.

```
route-guards-pract > src > app > users > TS users.component.ts > UsersComponent > ngOnInit  
1 import { Component, OnInit } from '@angular/core';  
2 import { ActivatedRoute, Router } from '@angular/router';  
3 import {UsersService}from '../users.service'  
4 @Component({  
5   selector: 'app-users',  
6   templateUrl: './users.component.html',  
7   styleUrls: ['./users.component.css']  
8 })  
9 export class UsersComponent implements OnInit {  
10   public users;  
11   public errorMsg:any;  
12   constructor(private _userService:UsersService, private activatedRoute:ActivatedRoute) { }  
13  
14   ngOnInit(): void {  
15     this.users=this.activatedRoute.snapshot.data['data'];  
16   }  
17  
18 }  
19
```

Here in the line: **this.activatedRoute.snapshot.data[ 'data' ]**

The key “**data**” which we used for associating the route is used.

This page is viewed after the data is loaded from the remote repository.



S.No	Name
1	Leanne Graham
2	Ervin Howell
3	Clementine Bauch
4	Patricia Lebsack
5	Chelsey Dietrich

## **CanLoad**

As mentioned earlier, it is similar to CanActivate guard. It can prevent unauthorized access, but in addition to that this guard will not allow the browser to lazy load the module.

Please explore more on this guard.

## **REFERENCES:**

1. [https://www.youtube.com/playlist?list=PLC8OkhrVTHNHi6Do0etu\\_wMvAyjdjukQp](https://www.youtube.com/playlist?list=PLC8OkhrVTHNHi6Do0etu_wMvAyjdjukQp)
2. <https://www.digitalocean.com/community/tutorials/angular-route-guards>

**Document By**

**Lavanya S (M1060521)**