An Angular allows us to protect routes with guards. Route guards in Angular are a way of protecting navigation by allowing or denying a route based on the condition we supply. The most commonly used angular route guard types are CanActivate and CanDeactivate.

Route guards give us a lot of flexibility in the kinds of checks we want to add before a route opens or closes. We can generate angular guard in our project as

$ng g guard my-new-guard guard-type

The guard type can be canActivate, canActivateChild, canDeactivate and canLoad.

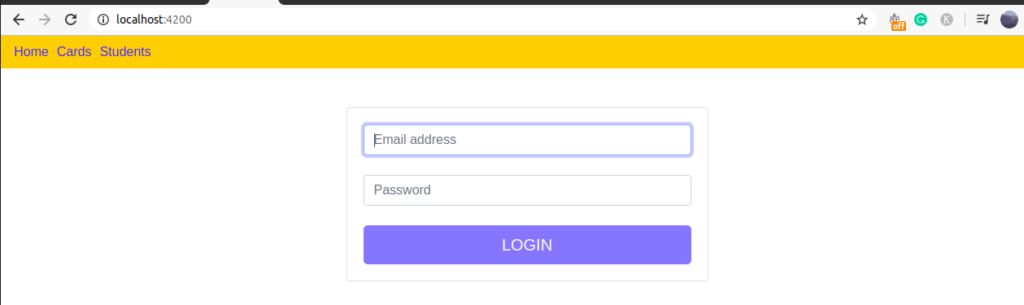
**Different types of angular route guard**

Based on user requirement, we can implement different guard. Angular provide following different type of router guards as

1. **CanActivate:** Checks to see if we can visit a route based on condition.
2. **CanActivateChild:**Checks to see if we can visit route children based on condition.
3. **CanDeactivate:**Prevent the user from navigating away from the current content based on condition.
4. **canLoad:**Used to determine whether the user can navigate to a lazy loaded module prior to loading it
5. **Resolve:**Delay navigation until a task is complete before route activate.

## Angular auth guard and an example using canActivate

Let’s take a look at what Angular’s route guards are and how to use them to help with authentication in your Angular apps.  
**The CanActivate: Can the user navigate to a route.?**  
We can have a guard class that implements canActivate() method, we can inject this class to the router configuration. The Angular guard can be used to prevent a route from being activated. This is useful to navigate the user away to some resource if the user isn’t allowed to navigate to a route. This is not only limited to authorized users only, but we can also implement different logic that returns boolean value to control the activation of the route.



We will demonstrate an example to control the accessing some of our components like cards and students from unauthorized users. The links like card and student can access only once the user has logged in.

### Step 1: Create an angular project and add required component

We need three components, one for the login component to allow a user to login. The other two components to demonstrate the auth guard to control the navigation route. We also auth service for login validation.

$ng new authApp

$ng generate component login

$ng generate component cards

$ng generate component students

$ng generate service auth

### Step 2: Edit app.component.hmtl to include router-outlet directive and navigation link

We already have discuss on router-outlet and routerLink directive in previous [router tutorial.](https://edupala.com/a-complete-guide-to-routing-in-angular/).

### Step 3: Edit auth.service.ts file to check user authentication.

We will set static value for username and password to demonstrate the auth guard demo.

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

@Injectable({

providedIn: 'root'

})

export class AuthService {

isAuthenticate = false;

login(user: string, password: string): boolean {

if (user === 'admin' && password === 'admin') {

this.isAuthenticate = true;

return this.isAuthenticate;

}

this.isAuthenticate = false;

return this.isAuthenticate;

}

### Step 4: Create an auth guard and implement canActivate interface.

Guards for route activation are classes that define a method called canActivate, which have optional ActivatedRouteSnapshot and RouterStateSnapshot arguments as resolvers.

$ng g guard auth CanActivate

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

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

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

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService) { }

canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot)

: Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

if (this.authService.isAuthenticate) {

return true;

} else {

return false;

}

}

The canActivate() method can return three different types. We have a return value of observable or promise or just boolean. As we might have some guards which execute some synchronous code on the client-side, or we might have some asynchronous code to wait for a remote server and both are possible with canActivate(),

### Step 5: Create/edit app.routing.module.ts and configure the auth guard on the router.

If your application didn’t have app.routing.module.ts, then we can manually create our new module and hook it up to the main AppModule. Angular CLI command to generate route module for us, by running:

**ng generate module app-routes –flat –module=app**

In order to check if a component can be activated we add a guard class to the key canActivate in our router configuration.

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

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

import { CardsComponent } from './cards/cards.component';

import { LoginComponent } from './login/login.component';

import { StudentsComponent } from './students/students.component';

import { AuthGuard } from './auth.guard';

const routes: Routes = [

{ path: '', component: LoginComponent },

{ path: 'cards', component: CardsComponent, canActivate: [AuthGuard] },

{ path: 'students', component: StudentsComponent, canActivate: [AuthGuard] }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

We also need to register our angular routing in app.module.ts file as

import { FormsModule } from '@angular/forms';

import { BrowserModule } from '@angular/platform-browser';

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

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

import { AppComponent } from './app.component';

import { LoginComponent } from './login/login.component';

import { StudentsComponent } from './students/students.component';

@NgModule({

declarations: [

AppComponent,

LoginComponent,

StudentsComponent

],

imports: [

BrowserModule,

FormsModule,

AppRoutingModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

We only want to give access to cards and students component if the user is logged in. You can read more on the Angular canActivate(), guard, please check the official Angular [document.](https://angular.io/api/router/CanActivate)