Using Angular Route Guard For securing routes

Theophilus Omoregbee Follow
Aug 6, 2018 · 5 min read

If you are trying to block some routes from loading based on some permissions or blocking a route based if not authenticated, then you can read along.

We are going to do an example where you need to login to view your dashboard and permission-based routing where if no permission to view a route you see 404 page.

Create an angular app with angular cli

```
ng new authGuard
```

Setup components

Let's create our components

```
ng g c login --spec false -is
ng g c pageNotFound --spec false -is -it
ng g m dashboard
ng g c dashboard/layout --spec false -is -it
ng g c dashboard/home --spec false -is -it
ng g c dashboard/admin --spec false -is -it
```

we have created our sample login component, a `pageNotFound` component which is going to hold our 404 page, then dashboard modules to hold our secured components, a layout component to handle a minimal templating of our dashboard, a home component which is available for everyone signed in like a landing page and admin component only allowed to be viewed by those with admin role and also authenticated.

`-it` is for an inline template and `is` inline style instead of having separate `.html` and `.css` files for each generate component

Create our Routes

create a file `src/app/app.routes.ts` and add the below code

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

export const APP_ROUTES: Routes = [
  { path: 'login', component: LoginComponent },
  { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

app.routes.ts hosted with ♥ by GitHub view raw

app.routes.ts

add the created routes above to our `app.module.ts` like so

```
...
import { RouterModule } from '@angular/router';
import { APP_ROUTES } from './app.routes';
...

imports: [
    BrowserModule,
    RouterModule.forRoot(APP_ROUTES)
  ]
...
```

Let's set up routes for our secured dashboard in `/dashboard/dashboard.routes.ts`

```
import { Routes } from '@angular/router';
import { LayoutComponent } from './layout/layout.component';
import { HomeComponent } from './home/home.component';
import { AdminComponent } from './admin/admin.component';

export const dashboardRoutes: Routes = [
  {
    path: 'dashboard',
    component: LayoutComponent,
    children: [
      { path: '', redirectTo: 'home', pathMatch: 'full' },
      { path: 'home', component: HomeComponent},
      { path: 'admin', component: AdminComponent}
    ]
  }
];
```

dashboard.routes.ts hosted with ♥ by GitHub view raw

dashboard.routes.ts

Open `dashboard.module.ts` and import our routes, like so

```
...
import { RouterModule } from '@angular/router';
import { dashboardRoutes } from './dashboard.routes';
...

imports: [
    RouterModule.forChild(dashboardRoutes)
  ]
...
```

Finally, we attach the dashboard module to our app main module. Our `app.module.ts` should look this

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
import { APP_ROUTES } from './app.routes';
import { DashboardModule } from './dashboard/dashboard.module';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    PageNotFoundComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(APP_ROUTES),
    DashboardModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.modules.ts hosted with ❤ by GitHub view raw

app.module.ts

Flesh out our components

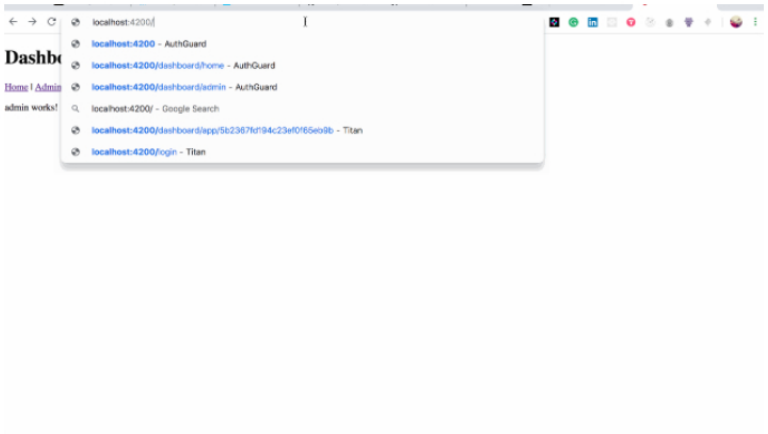
Open `app.component.html` let's add our router outlet

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

and update `layout.component.ts` template to the below snippet

```
<h1>Dashboard Layout</h1>
    <p>
      <a routerLink="home" >Home</a> |
      <a routerLink="admin"> Admin </a>
    </p>

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

Run `ng serve — open,` our app should now be like the below gif



app state

Route Guarding

Let's guard our dashboard to only be accessed by authenticated users.

```
ng g s guards/authGuard --spec false
```

A service should be generated, Open `guards/auth-guard.service.ts` let's get down to securing our dashboard using JWT strategy(as a sample auth method, you can use any preferred auth method)

```
1  import { AuthService } from './../services/auth.service';
2  import { Injectable } from '@angular/core';
3  import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router, Route } from '@angula
4  import { Observable } from 'rxjs';
5
6  @Injectable()
7  export class AuthGuard implements CanActivate {
8
9
10   constructor(private _authService: AuthService, private _router: Router) {
11   }
12
13   canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> | P
14     if (this._authService.isAuthenticated()) {
15         return true;
16     }
17
18     // navigate to login page
19     this._router.navigate(['/login']);
20     // you can save redirect url so after authing we can move them back to the page they request
21     return false;
22   }
23
24 }
```

auth-guard.service.ts hosted with ❤ by GitHub view raw

auth-guard.service.ts

We implemented the `canActivate` interface, which is going to determine if the component to be entered or not. Observed we are making use of `AuthService` which is a simple implementation to check if there's a token in local storage and also check if the token is valid or not.

From our above code, if `canActivate` returns `true` the component is rendered else we redirect the user back to `login` component. At this point, you can do more like saving the current URL the user is trying to open and after authing, they can be redirected to the saved URL.

> *`canActivate` method can also be asynchronous when the checking is calling another service like sending a confirmation token to your backend server, which can either be a `Promise<boolean>` or `Observable<boolean>`.*

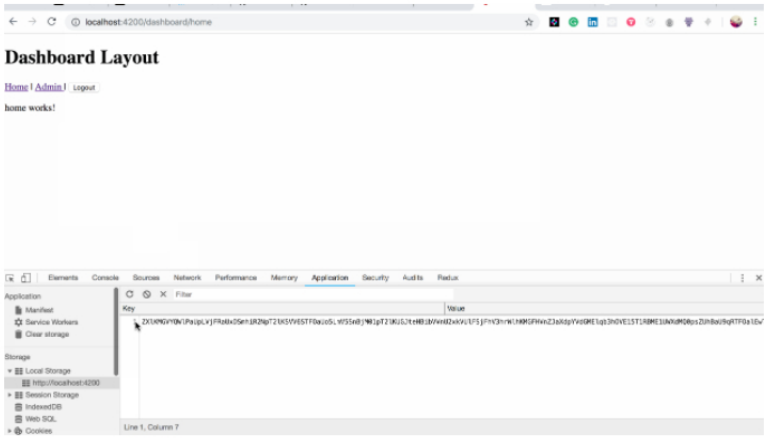
Let's use the route in our dashboard, open `dashboard.routes.ts` and add the below code

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

export const dashboardRoutes: Routes = [
  {
    ...
    component: LayoutComponent,
    canActivate: [AuthGuard],
    ...
  }
];
```

We added an extra field to our dashboard route `canActivate` which is going to read the `canActivate` method from our `AuthGuard`. Observe that it's taking an array which means you can create multiple guards for different purposes, like which type of users can view the dashboard as a whole after authentication. Guards are processed in the order they were placed in the `canActivate` array field.

Lastly, we need to provide our `AuthGuard` in our `dashboard.module.ts`

```
...
import { AuthGuard } from "../guards/auth-guard.service";
...

imports: [
...
],
providers: [AuthGuard],
...
```



Dashboard secured

We are almost there.

Role-based guarding

Since we are using JWT auth strategy, we are going to decode our token and then use the decoded user object to check if a role is allowed to access

and then use the decoded user object to check if a role is allowed to access a route or not.

> *We don't just hide the menu link to the secured route, we also need to guard it against users who are familiar with the URL to secured routes. Don't forget to also add backend role checker for any action that's going to be taken from a secured routes too.*

Top highlight

```
ng g s guards/roleGuard --spec false
```

Open `role-guard.service.ts` and paste the below code

```
import { AuthService } from './../services/auth.service';
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router, Route } from '@angula
import { Observable } from 'rxjs';

@Injectable()
export class RoleGuard implements CanActivate {


  constructor(private _authService: AuthService, private _router: Router) {
  }

  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> | P
    const user = this._authService.decode();

    if (user.Role === next.data.role) {
      return true;
    }

    // navigate to not found page
    this._router.navigate(['/404']);
    return false;
  }

}
```

role-guard.service.ts hosted with ❤ by GitHub view raw

role-guard.service.ts

We are making use of the `ActivatedRouteSnapshot` data, which is for passing extra information to a route during route setup. We check if the role added to the route is the same as the user role, if so we move on, else we take the user to the `pageNotFound` component.

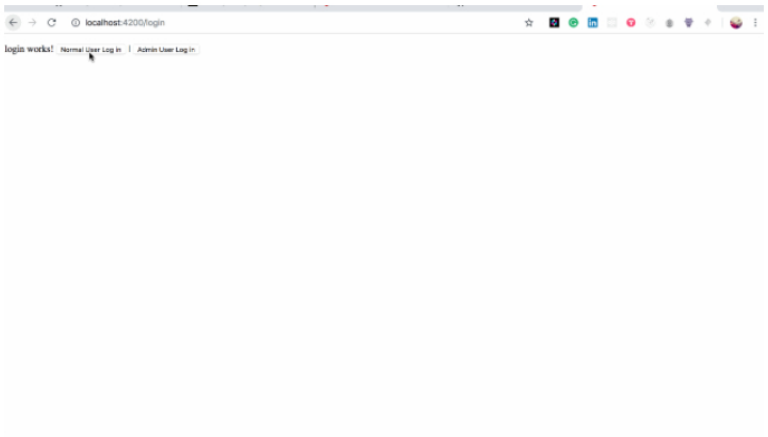
Let's apply our route guard now. Open up our routes configuration in `dashboard.routes.ts` let's modify the admin route

```
...
import { RoleGuard } from '../guards/role-guard.service';
...

children:[
   ...,
  {
    path: 'admin',
    component: AdminComponent,
    canActivate: [RoleGuard],
    data: {role: 'Admin'}
  },
  ...

];
```

Don't forget to provide our guard like the way we did for `AuthGuard` in our

`dashboard.module.ts` .

Let's save and run our app now



We've successfully created secured routes using Angular's guard, which was basically the use of `canActivate` method. There are other guard methods like `canActivateChild`, `CanLoad`, e.t.c. Check Angular Docs for more.

Complete Source code

theo4u/AuthGuard
AuthGuard - Example repo for guarding routes post
github.com

To prefetch data for your component which is guarded, check out the below link

The right way to prefetch data for your angular components/pages
Ever wondered why you load your component and now do API call? What you might be familiar with is, showing a loader…
codeburst.io

codeburst.io

> Subscribe to *CodeBurst's* once-weekly **Email Blast**, Follow *CodeBurst* on **Twitter**, view **The 2018 Web Developer Roadmap**, and **Learn Full Stack Web Development**.

Thanks to Ozioma Ogbe and Peter Mbanugo.

JavaScript Angularjs Router Navigation Single Page Applications

1.2K 14

WRITTEN BY
Theophilus Omoregbee

Follow

Dev at Coherent Path

codeburst

Bursts of code to power through your day. Web Development articles, tutorials, and news.

Follow

More From Medium

React vs. Angular : Which Is Better for Web Development?

BK Puneioi I Owe

How Do You Get Undone?

Michael Dotson

Tutorial: Build a Toggle Button Using ReactJS Inverse Data Flow in ES6

Stephanie Zou in The Startup

How to allow a child component to access and modify a parent's state in React

Vahid Dejwakh

Storybook Docs for new frameworks

Michael Shilman in Storybook

Using Hasura's GraphQL Engine with ReactJS and authentication using auth0.

Gagan Ganapathy

WebAssembly and .NET

GrapeCity Developer Solutions in GrapeCity

How to Use Face ID With React Native or Expo

Nicolas Dommanget-Muller in The Startup

Learn more.

Medium is an open platform where 170 million readers come to find insightful and dynamic thinking. Here, expert and undiscovered voices alike dive into the heart of any topic and bring new ideas to the surface. Learn more

Make Medium yours.

Follow the writers, publications, and topics that matter to you, and you'll see them on your homepage and in your inbox. Explore

Write a story on Medium.

If you have a story to tell, knowledge to share, or a perspective to offer — welcome home. It's easy and free to post your thinking on any topic. Start a blog

Medium

About Write Help Legal