PUBLISHED: SEPTEMBER 09 2020

Angular 10 - Role Based Authorization Tutorial with Example

Tutorial built with **Angular 10.1.0**

Other versions available:

- **Angular:** Angular 9 (/post/2020/05/15/angular-9-role-based-authorization-tutorial-with-example), 8 (/post/2019/08/06/angular-8-role-based-authorization-tutorial-with-example), 7 (/post/2018/11/22/angular-7-role-based-authorization-tutorial-with-example)
- **React:** React (/post/2019/02/01/react-role-based-authorization-tutorial-with-example)
- **Vue:** Vue.js (/post/2019/03/08/vuejs-role-based-authorization-tutorial-with-example)

In this tutorial we'll go through an example of how to implement role based authorization / access control in Angular 10. The example builds on a previous tutorial I posted which focuses on JWT authentication (/post/2020/07/09/angular-10-jwt-authentication-example-tutorial), this example has been extended to include role based access control on top of the JWT authentication.

The tutorial example is pretty minimal and contains just 3 pages to demonstrate role based authorization - a login page, a home page and an admin page. There are two roles - a regular user (`Role.User`) that can access the home page, and an admin user (`Role.Admin`) that can access everything (i.e. the home page and admin page).

Styling of the example app is all done with Bootstrap 4.5 CSS, for more info about Bootstrap see https://getbootstrap.com/docs/4.5/getting-started/introduction/ (https://getbootstrap.com/docs/4.5/getting-started/introduction/).

The tutorial code is available on GitHub at https://github.com/cornflourblue/angular-10-role-based-authorization-example (https://github.com/cornflourblue/angular-10-role-based-authorization-example).

/

NOTE: You can also start the app with the Angular CLI command `ng serve --open`. To do this first install the Angular CLI globally on your system with the command `npm install -g @angular/cli`.

For more info on setting up your local Angular dev environment see Angular - Setup Development Environment (/post/2020/06/02/angular-setup-development-environment).

Run the Angular App with an ASP.NET Core 3.1 API

For full details about the example ASP.NET Core API see the post ASP.NET Core 3.1 - Role Based Authorization Tutorial with Example API (/post/2019/10/16/aspnet-core-3-role-based-authorization-tutorial-with-example-api). But to get up and running quickly just follow the below steps.

1. Install the .NET Core SDK from https://www.microsoft.com/net/download/core (https://www.microsoft.com/net/download/core).
2. Download or clone the project source code from https://github.com/cornflourblue/aspnet-core-3-role-based-authorization-api (https://github.com/cornflourblue/aspnet-core-3-role-based-authorization-api)
3. Start the api by running `dotnet run` from the command line in the project root folder (where the WebApi.csproj file is located), you should see the message `Now listening on: http://localhost:4000`.
4. Back in the Angular app, remove or comment out the line below the comment `// provider used to create fake backend` located in the `/src/app/app.module.ts` file, then start the Angular app and it should now be hooked up with the ASP.NET Core role based auth API.

Run the Angular App with a Node.js API

For full details about the example Node.js API see the post Node.js - Role Based Authorization Tutorial with Example API (/post/2018/11/28/nodejs-role-based-authorization-tutorial-with-example-api). But to get up and running quickly just follow the below steps.

1. Install NodeJS and NPM from https://nodejs.org (https://nodejs.org).
2. Download or clone the project source code from https://github.com/cornflourblue/node-role-based-authorization-api (https://github.com/cornflourblue/node-role-based-authorization-api)
3. Start the api by running `npm start` from the command line in the project root folder, you should see the message `Server listening on port 4000`.
4. Back in the Angular app, remove or comment out the line below the comment `// provider used to create fake backend` located in the `/src/app/app.module.ts` file, then start the Angular app and it should now be hooked up with the Node.js role based auth API.

Angular 10 Project Structure

The Angular CLI was used to generate the base project structure with the `ng new <project name>` command, the CLI is also used to build and serve the application. For more info about the Angular CLI see https://angular.io/cli (https://angular.io/cli).

The app and code structure of the tutorial mostly follows the best practice recommendations in the official Angular Style Guide (https://angular.io/guide/styleguide), with a few of my own tweaks here and there.

Each feature has it's own folder (home, admin & login), other shared/common code such as services, models, helpers etc are placed in folders prefixed with an underscore `_` to easily differentiate them and group them together at the top of the folder structure.

The `index.ts` files in each folder are barrel files that group the exported modules from a folder together so they can be imported using the folder path rather than the full module path, and to enable importing multiple modules in a single import (e.g. `import { AuthenticationService, UserService } from '../_services'`).

Path aliases `@app` and `@environments` have been configured in tsconfig.base.json that map to the `/src/app` and `/src/environments` directories. This allows imports to be relative to the app and environments folders by prefixing import paths with aliases instead of having to use long relative paths (e.g. `import MyComponent from '../../../MyComponent'`).

Here are the main project files that contain the application logic, I left out some files that were generated by Angular CLI `ng new` command that I didn't change.

- src
  - app
    - _helpers
      - auth.guard.ts
      - error.interceptor.ts
      - fake-backend.ts
      - jwt.interceptor.ts
      - index.ts
    - _models
      - role.ts
      - user.ts
      - index.ts
    - _services
      - authentication.service.ts
      - user.service.ts
      - index.ts
    - admin
      - admin.component.html
      - admin.component.ts
      - index.ts
    - home

/

            - home.component.html
            - home.component.ts
            - index.ts
        - login
            - login.component.html
            - login.component.ts
            - index.ts
        - app-routing.module.ts
        - app.component.html
        - app.component.ts
        - app.module.ts
    - environments
        - environment.prod.ts
        - environment.ts
    - index.html
    - main.ts
    - polyfills.ts
    - styles.less
- package.json
- tsconfig.base.json

Auth Guard

Path: /src/app/_helpers/auth.guard.ts

The auth guard is an angular route guard that's used to prevent unauthenticated or unauthorized users from accessing restricted routes, it does this by implementing the `CanActivate` interface which allows the guard to decide if a route can be activated with the `canActivate()` method. If the method returns `true` the route is activated (allowed to proceed), otherwise if the method returns `false` the route is blocked.

The auth guard uses the authentication service to check if the user is logged in, if they are logged in it checks if their role is authorized to access the requested route. If they are logged in and authorized the `canActivate()` method returns `true` , otherwise it returns `false` and redirects the user to the login page.

Angular route guards are attached to routes in the router config, this auth guard is used in app-routing.module.ts to protect the home page and admin page routes.

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

import { AuthenticationService } from '@app/_services';

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) { }

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const user = this.authenticationService.userValue;
        if (user) {
            // check if route is restricted by role
            if (route.data.roles && route.data.roles.indexOf(user.role) === -1) {
                // role not authorised so redirect to home page
                this.router.navigate(['/']);
                return false;
            }

            // authorised so return true
            return true;
        }

        // not logged in so redirect to login page with the return url
        this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
        return false;
    }
}
```

Back to top

Error Interceptor

Path: /src/app/_helpers/error.interceptor.ts

The Error Interceptor intercepts http responses from the api to check if there were any errors. If there is a `401 Unauthorized` or `403 Forbidden` response the user is automatically logged out of the application, all other errors are re-thrown up to the calling service so an alert with the error can be displayed on the screen.

It's implemented using the Angular `HttpInterceptor` interface included in the `HttpClientModule`, by implementing the HttpInterceptor interface you can create a custom interceptor to catch all error responses from the api in a single location.

Http interceptors are added to the request pipeline in the providers section of the app.module.ts file.

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

import { AuthenticationService } from '@app/_services';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
    constructor(private authenticationService: AuthenticationService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        return next.handle(request).pipe(catchError(err => {
            if ([401, 403].indexOf(err.status) !== -1) {
                // auto logout if 401 Unauthorized or 403 Forbidden response returned from api
                this.authenticationService.logout();
            }

            const error = err.error.message || err.statusText;
            return throwError(error);
        }))
    }
}
```

Back to top

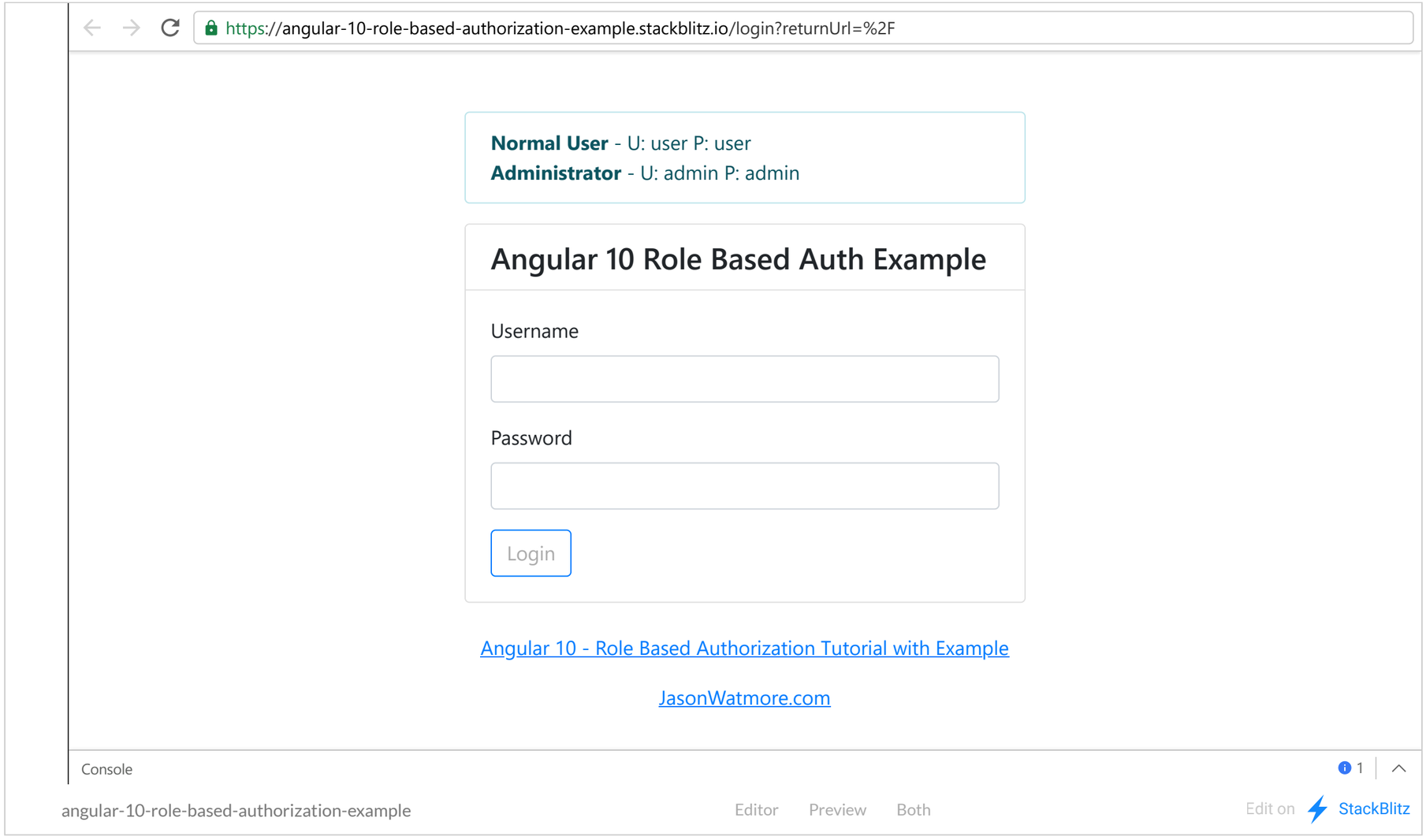
Fake Backend Provider

Path: /src/app/_helpers/fake-backend.ts

In order to run and test the Angular application without a real backend API, the example uses a fake backend that intercepts the HTTP requests from the Angular app and send back "fake" responses. This is done by a class that implements the Angular `HttpInterceptor` interface, for more information on Angular HTTP Interceptors see https://angular.io/api/common/http/HttpInterceptor (https://angular.io/api/common/http/HttpInterceptor) or this article (https://medium.com/@ryanchenkie_40935/angular-authentication-using-the-http-client-and-http-interceptors-2f9d1540eb8).

The fake backend provider contains a hardcoded collection of users and provides fake implementations for the api endpoints "authenticate", "get user by id", and "get all users", these would be handled by a real api and database in a production application.

Here it is in action:



(See on StackBlitz at https://stackblitz.com/edit/angular-10-role-based-authorization-example (https://stackblitz.com/edit/angular-10-role-based-authorization-example))

Run the Angular 10 Role Based Authorization Example Locally

1. Install NodeJS and NPM from https://nodejs.org (https://nodejs.org).
2. Download or clone the tutorial project source code from https://github.com/cornflourblue/angular-10-role-based-authorization-example (https://github.com/cornflourblue/angular-10-role-based-authorization-example)
3. Install all required npm packages by running `npm install` or `npm i` from the command line in the project root folder (where the package.json is located).
4. Start the application by running `npm start` from the command line in the project root folder, this will build the application and automatically launch it in the browser on the URL http://localhost:4200.

The "authenticate" route is used for logging in to the application and is publicly accessible, the "get user by id" route is restricted to authenticated users in any role, however regular users can only access their own user record while admin users can access any user record. The "get all users" route is restricted to admin users only.

If the request doesn't match any of the faked routes it is passed through as a real HTTP request to the backend API.

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpResponse, HttpHandler, HttpEvent, HttpInterceptor, HTTP_INTERCEPTORS } from '@angular/common/http';
import { Observable, of, throwError } from 'rxjs';
import { delay, materialize, dematerialize } from 'rxjs/operators';

import { Role } from '@app/_models';

const users = [
    { id: 1, username: 'admin', password: 'admin', firstName: 'Admin', lastName: 'User', role: Role.Admin },
    { id: 2, username: 'user', password: 'user', firstName: 'Normal', lastName: 'User', role: Role.User }
];

@Injectable()
export class FakeBackendInterceptor implements HttpInterceptor {
    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const { url, method, headers, body } = request;

        return handleRoute();

        function handleRoute() {
            switch (true) {
                case url.endsWith('/users/authenticate') && method === 'POST':
                    return authenticate();
                case url.endsWith('/users') && method === 'GET':
                    return getUsers();
                case url.match(/\/users\/\d+$/) && method === 'GET':
                    return getUserById();
                default:
                    // pass through any requests not handled above
                    return next.handle(request);
            }

        }

        // route functions

        function authenticate() {
            const { username, password } = body;
            const user = users.find(x => x.username === username && x.password === password);
            if (!user) return error('Username or password is incorrect');
            return ok({
                id: user.id,
                username: user.username,
                firstName: user.firstName,
                lastName: user.lastName,
                role: user.role,
                token: `fake-jwt-token.${user.id}`
```

```
            });
        }

        function getUsers() {
            if (!isAdmin()) return unauthorized();
            return ok(users);
        }

        function getUserById() {
            if (!isLoggedIn()) return unauthorized();

            // only admins can access other user records
            if (!isAdmin() && currentUser().id !== idFromUrl()) return unauthorized();

            const user = users.find(x => x.id === idFromUrl());
            return ok(user);
        }

        // helper functions

        function ok(body) {
            return of(new HttpResponse({ status: 200, body }))
                .pipe(delay(500)); // delay observable to simulate server api call
        }

        function unauthorized() {
            return throwError({ status: 401, error: { message: 'unauthorized' } })
                .pipe(materialize(), delay(500), dematerialize()); // call materialize and dematerialize to ensure delay even if an error is
        }

        function error(message) {
            return throwError({ status: 400, error: { message } })
                .pipe(materialize(), delay(500), dematerialize());
        }

        function isLoggedIn() {
            const authHeader = headers.get('Authorization') || '';
            return authHeader.startsWith('Bearer fake-jwt-token');
        }

        function isAdmin() {
            return isLoggedIn() && currentUser().role === Role.Admin;
        }

        function currentUser() {
            if (!isLoggedIn()) return;
            const id = parseInt(headers.get('Authorization').split('.')[1]);
            return users.find(x => x.id === id);
```

```
        }

        function idFromUrl() {
            const urlParts = url.split('/');
            return parseInt(urlParts[urlParts.length - 1]);
        }
    }
}

export const fakeBackendProvider = {
    // use fake backend in place of Http service for backend-less development
    provide: HTTP_INTERCEPTORS,
    useClass: FakeBackendInterceptor,
    multi: true
};
```

Back to top

JWT Interceptor

Path: /src/app/_helpers/jwt.interceptor.ts

The JWT Interceptor intercepts http requests from the application to add a JWT auth token to the Authorization header if the user is logged in and the request is to the application api url (`environment.apiUrl`).

It's implemented using the `HttpInterceptor` interface included in the `HttpClientModule` , by implementing the HttpInterceptor interface you can create a custom interceptor to modify http requests before they get sent to the server.

Http interceptors are added to the request pipeline in the providers section of the app.module.ts file.

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable } from 'rxjs';

import { environment } from '@environments/environment';
import { AuthenticationService } from '@app/_services';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {
    constructor(private authenticationService: AuthenticationService) { }

    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        // add auth header with jwt if user is logged in and request is to api url
        const user = this.authenticationService.userValue;
        const isLoggedIn = user && user.token;
        const isApiUrl = request.url.startsWith(environment.apiUrl);
        if (isLoggedIn && isApiUrl) {
            request = request.clone({
                setHeaders: {
                    Authorization: `Bearer ${user.token}`
                }
            });
        }

        return next.handle(request);
    }
}
```

Back to top

Role Model

Path: /src/app/_models/role.ts

The role model contains an enum that defines the roles that are supported by the application.

```
export enum Role {
    User = 'User',
    Admin = 'Admin'
}
```

Back to top

User Model

/

Path: /src/app/_models/user.ts

The user model is a small class that defines the properties of a user, including their `role` and jwt auth `token?` (the trailing question mark `?` makes the property optional in TypeScript).

```
import { Role } from "./role";

export class User {
    id: number;
    firstName: string;
    lastName: string;
    username: string;
    role: Role;
    token?: string;
}
```

Back to top

Authentication Service

Path: /src/app/_services/authentication.service.ts

The authentication service is used to login & logout of the Angular app, it notifies other components when a user logs in & out, and provides access the currently logged in user.

RxJS Subjects and Observables are used to store the current user object and notify other components when the user logs in and out of the app. Angular components can `subscribe()` to the public `user: Observable` property to be notified of changes, and notifications are sent when the `this.userSubject.next()` method is called in the `login()` and `logout()` methods, passing the argument to each subscriber. The RxJS `BehaviorSubject` is a special type of Subject that keeps hold of the current value and emits it to any new subscribers as soon as they subscribe, while regular Subjects don't store the current value and only emit values that are published after a subscription is created. For more info on component communication with RxJS see Angular 10 - Communicating Between Components with Observable & Subject (/post/2020/07/06/angular-10-communicating-between-components-with-observable-subject).

The `login()` method sends the user credentials to the API via an HTTP POST request for authentication. If successful the user object including a JWT auth token are stored in localStorage to keep the user logged in between page refreshes. The user object is then published to all subscribers with the call to `this.userSubject.next(user);`.

The `constructor()` of the service initializes the `userSubject` with the user object from localStorage which enables the user to stay logged in between page refreshes or after the browser is closed. The public `user` property is then set to `this.userSubject.asObservable();` which allows other components to subscribe to the `user` Observable but doesn't allow them to publish to the `userSubject`, this is so logging in and out of the app can only be done via the authentication service.

The `userValue` getter provides a quick and easy way to get the value of the currently logged in user without having to subscribe to the `user` Observable.

The `logout()` method removes the current user object from local storage and publishes `null` to the `userSubject` to notify all subscribers that the user has logged out.

NOTE: If you don't like the idea of storing the current user details in local storage, all you need to do is change the 3 references to `localStorage` in this file. Other options are session storage, cookies, or you could simply not store the user details in the browser, although be aware that with this last option the user will be automatically logged out if they refresh the page.

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { environment } from '@environments/environment';
import { User } from '@app/_models';

@Injectable({ providedIn: 'root' })
export class AuthenticationService {
    private userSubject: BehaviorSubject<User>;
    public user: Observable<User>;

    constructor(
        private router: Router,
        private http: HttpClient
    ) {
        this.userSubject = new BehaviorSubject<User>(JSON.parse(localStorage.getItem('user')));
        this.user = this.userSubject.asObservable();
    }

    public get userValue(): User {
        return this.userSubject.value;
    }

    login(username: string, password: string) {
        return this.http.post<any>(`${environment.apiUrl}/users/authenticate`, { username, password })
            .pipe(map(user => {
                // store user details and jwt token in local storage to keep user logged in between page refreshes
                localStorage.setItem('user', JSON.stringify(user));
                this.userSubject.next(user);
                return user;
            }));
    }

    logout() {
        // remove user from local storage to log user out
        localStorage.removeItem('user');
        this.userSubject.next(null);
        this.router.navigate(['/login']);
    }
}
```

Back to top

User Service

Path: /src/app/_services/user.service.ts

The user service contains just a couple of methods for retrieving user data from the api, it acts as the interface between the Angular application and the backend api.

I included the user service to demonstrate accessing secure api endpoints with the http authorization header set after logging in to the application, the auth header is set with a JWT token in the JWT Interceptor. The secure endpoints in the example are mocked routes implemented in the fake backend provider.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { environment } from '@environments/environment';
import { User } from '@app/_models';

@Injectable({ providedIn: 'root' })
export class UserService {
    constructor(private http: HttpClient) { }

    getAll() {
        return this.http.get<User[]>(`${environment.apiUrl}/users`);
    }

    getById(id: number) {
        return this.http.get<User>(`${environment.apiUrl}/users/${id}`);
    }
}
```

Back to top

Admin Component Template

Path: /src/app/admin/admin.component.html

The admin component template contains html and angular 10 template syntax for displaying a list of all users retrieved from a secure api endpoint.

```
<div class="card mt-4">
    <h4 class="card-header">Admin</h4>
    <div class="card-body">
        <p>This page can be accessed <u>only by administrators</u>.</p>
        <p class="mb-1">All users from secure (admin only) api end point:</p>
        <div *ngIf="loading" class="spinner-border spinner-border-sm"></div>
        <ul *ngIf="users">
            <li *ngFor="let user of users">{{user.firstName}} {{user.lastName}}</li>
        </ul>
    </div>
</div>
```

Back to top

Admin Component

Path: /src/app/admin/admin.component.ts

The admin component calls the user service to get all users from a secure api endpoint and assign them to the local `users` property which is accessible from the admin component template above.

```
import { Component, OnInit } from '@angular/core';
import { first } from 'rxjs/operators';

import { User } from '@app/_models';
import { UserService } from '@app/_services';

@Component({ templateUrl: 'admin.component.html' })
export class AdminComponent implements OnInit {
    loading = false;
    users: User[] = [];

    constructor(private userService: UserService) { }

    ngOnInit() {
        this.loading = true;
        this.userService.getAll().pipe(first()).subscribe(users => {
            this.loading = false;
            this.users = users;
        });
    }
}
```

Back to top

Home Component Template

Path: /src/app/home/home.component.html

The home component template contains html and angular 10 template syntax for displaying a simple welcome message and the current user record that is fetched from a secure api endpoint.

```
<div class="card mt-4">
    <h4 class="card-header">Home</h4>
    <div class="card-body">
        <p>You're logged in with Angular 10 & JWT!!</p>
        <p>Your role is: <strong>{{user.role}}</strong>.</p>
        <p>This page can be accessed by <u>all authenticated users</u>.</p>
        <p class="mb-1">Current user from secure api end point:</p>
        <div *ngIf="loading" class="spinner-border spinner-border-sm"></div>
        <ul *ngIf="userFromApi">
            <li>{{userFromApi.firstName}} {{userFromApi.lastName}}</li>
        </ul>
    </div>
</div>
```

Back to top

Home Component

Path: /src/app/home/home.component.ts

The home component gets the current user from the authentication service and then gets the current user from the api with the user service. We only really needed to get the user from the authentication service, but I included getting it from the user service as well to demonstrate fetching data from a secure api endpoint.

```
import { Component } from '@angular/core';
import { first } from 'rxjs/operators';

import { User } from '@app/_models';
import { UserService, AuthenticationService } from '@app/_services';

@Component({ templateUrl: 'home.component.html' })
export class HomeComponent {
    loading = false;
    user: User;
    userFromApi: User;

    constructor(
        private userService: UserService,
        private authenticationService: AuthenticationService
    ) {
        this.user = this.authenticationService.userValue;
    }

    ngOnInit() {
        this.loading = true;
        this.userService.getById(this.user.id).pipe(first()).subscribe(user => {
            this.loading = false;
            this.userFromApi = user;
        });
    }
}
```

Back to top

Login Component Template

Path: /src/app/login/login.component.html

The login component template contains a login form with username and password fields. It displays validation messages for invalid fields when the submit button is clicked. The form submit event is bound to the `onSubmit()` method of the login component.

The component uses reactive form validation to validate the input fields, for more information about angular reactive form validation see Angular 10 - Reactive Forms Validation Example (/post/2020/07/07/angular-10-reactive-forms-validation-example).

```
<div class="col-md-6 offset-md-3 mt-5">
    <div class="alert alert-info">
        <strong>Normal User</strong> - U: user P: user<br />
        <strong>Administrator</strong> - U: admin P: admin
    </div>
    <div class="card">
        <h4 class="card-header">Angular 10 Role Based Auth Example</h4>
        <div class="card-body">
            <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
                <div class="form-group">
                    <label for="username">Username</label>
                    <input type="text" formControlName="username" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.username.err
                    <div *ngIf="submitted && f.username.errors" class="invalid-feedback">
                        <div *ngIf="f.username.errors.required">Username is required</div>
                    </div>
                </div>
                <div class="form-group">
                    <label for="password">Password</label>
                    <input type="password" formControlName="password" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password
                    <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
                        <div *ngIf="f.password.errors.required">Password is required</div>
                    </div>
                </div>
                <button [disabled]="loading" class="btn btn-primary">
                    <span *ngIf="loading" class="spinner-border spinner-border-sm mr-1"></span>
                    Login
                </button>
                <div *ngIf="error" class="alert alert-danger mt-3 mb-0">{{error}}</div>
            </form>
        </div>
    </div>
</div>
```

Back to top

Login Component

Path: /src/app/login/login.component.ts

The login component uses the authentication service to login to the application. If the user is already logged in they are automatically redirected to the home page.

The `loginForm: FormGroup` object defines the form controls and validators, and is used to access data entered into the form. The FormGroup is part of the Angular Reactive Forms module and is bound to the login template above with the formGroup directive (`[formGroup]="loginForm"`).

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AuthenticationService } from '@app/_services';

@Component({ templateUrl: 'login.component.html' })
export class LoginComponent implements OnInit {
    loginForm: FormGroup;
    loading = false;
    submitted = false;
    error = '';

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private authenticationService: AuthenticationService
    ) {
        // redirect to home if already logged in
        if (this.authenticationService.userValue) {
            this.router.navigate(['/']);
        }
    }

    ngOnInit() {
        this.loginForm = this.formBuilder.group({
            username: ['', Validators.required],
            password: ['', Validators.required]
        });
    }

    // convenience getter for easy access to form fields
    get f() { return this.loginForm.controls; }

    onSubmit() {
        this.submitted = true;

        // stop here if form is invalid
        if (this.loginForm.invalid) {
            return;
        }

        this.loading = true;
        this.authenticationService.login(this.f.username.value, this.f.password.value)
            .pipe(first())
```

```
            .subscribe({
                next: () => {
                    // get return url from query parameters or default to home page
                    const returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
                    this.router.navigateByUrl(returnUrl);
                },
                error: error => {
                    this.error = error;
                    this.loading = false;
                }
            });
    }
}
```

Back to top

App Routing Module

Path: /src/app/app-routing.module.ts

Routing for the Angular app is configured as an array of `Routes` , each component is mapped to a path so the Angular Router knows which component to display based on the URL in the browser address bar. The home and admin routes are secured by passing the AuthGuard to the `canActivate` property of the route. The admin route also sets the `roles` data property to `[Role.Admin]` so only admin users can access it.

The `Routes` array is passed to the `RouterModule.forRoot()` method which creates a routing module with all of the app routes configured, and also includes all of the Angular Router providers and directives such as the `<router-outlet></router-outlet>` directive. For more information on Angular Routing and Navigation see https://angular.io/guide/router (https://angular.io/guide/router).

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

import { HomeComponent } from './home';
import { AdminComponent } from './admin';
import { LoginComponent } from './login';
import { AuthGuard } from './_helpers';
import { Role } from './_models';

const routes: Routes = [
    {
        path: '',
        component: HomeComponent,
        canActivate: [AuthGuard]
    },
    {
        path: 'admin',
        component: AdminComponent,
        canActivate: [AuthGuard],
        data: { roles: [Role.Admin] }
    },
    {
        path: 'login',
        component: LoginComponent
    },

    // otherwise redirect to home
    { path: '**', redirectTo: '' }
];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

Back to top

App Component Template

Path: /src/app/app.component.html

The app component template is the root component template of the application, it contains the main nav bar which is only displayed for authenticated users, and a router-outlet directive for displaying the contents of each view based on the current route / path.

The nav bar contains links for the home page, the admin page and a logout link. The admin link is only displayed for users in the `Admin` role by using the `isAdmin` getter in an `*ngIf` directive. The logout link calls the `logout()` method of the app component on click.

```
<!-- nav -->
<nav class="navbar navbar-expand navbar-dark bg-dark" *ngIf="user">
    <div class="navbar-nav">
        <a class="nav-item nav-link" routerLink="/">Home</a>
        <a class="nav-item nav-link" routerLink="/admin" *ngIf="isAdmin">Admin</a>
        <a class="nav-item nav-link" (click)="logout()">Logout</a>
    </div>
</nav>

<!-- main app container -->
<div class="container">
    <router-outlet></router-outlet>
</div>
```

Back to top

App Component

Path: /src/app/app.component.ts

The app component is the root component of the application, it defines the root tag of the app as `<app></app>` with the selector property of the `@Component()` decorator.

It subscribes to the `user` observable in the authentication service so it can reactively show/hide the main navigation bar when the user logs in/out of the application. I didn't worry about unsubscribing from the observable here because it's the root component of the application, the only time the component will be destroyed is when the application is closed which would destroy any subscriptions as well.

The app component contains a `logout()` method which is called from the logout link in the main nav bar above to log the user out and redirect them to the login page. The `isAdmin()` getter returns true if the logged in user is in the `Admin` role, or false for non-admin users.

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';

import { AuthenticationService } from './_services';
import { User, Role } from './_models';

@Component({ selector: 'app', templateUrl: 'app.component.html' })
export class AppComponent {
    user: User;

    constructor(private authenticationService: AuthenticationService) {
        this.authenticationService.user.subscribe(x => this.user = x);
    }

    get isAdmin() {
        return this.user && this.user.role === Role.Admin;
    }

    logout() {
        this.authenticationService.logout();
    }
}
```

Back to top

App Module

Path: /src/app/app.module.ts

The app module defines the root module of the application along with metadata about the module. For more info about angular 10 modules check out this page (https://angular.io/docs/ts/latest/guide/ngmodule.html) on the official docs site.

This is where the fake backend provider is added to the application, to switch to a real backend simply remove the providers located below the comment `// provider used to create fake backend.`

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';

// used to create fake backend
import { fakeBackendProvider } from './_helpers';

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

import { JwtInterceptor, ErrorInterceptor } from './_helpers';
import { HomeComponent } from './home';
import { AdminComponent } from './admin';
import { LoginComponent } from './login';

@NgModule({
    imports: [
        BrowserModule,
        ReactiveFormsModule,
        HttpClientModule,
        AppRoutingModule
    ],
    declarations: [
        AppComponent,
        HomeComponent,
        AdminComponent,
        LoginComponent
    ],
    providers: [
        { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
        { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },

        // provider used to create fake backend
        fakeBackendProvider
    ],
    bootstrap: [AppComponent]
})

export class AppModule { }
```

Back to top

Production Environment Config

Path: /src/environments/environment.prod.ts

The production environment config contains variables required to run the application in production. This enables you to build the application with a different configuration for each different environment (e.g. production & development) without updating the app code.

When you build the application for production with the command `ng build --prod`, the output `environment.ts` is replaced with `environment.prod.ts`.

```
export const environment = {
    production: true,
    apiUrl: 'http://localhost:4000'
};
```

Back to top

Development Environment Config

Path: /src/environments/environment.ts
The development environment config contains variables required to run the application in development.

Environment config is accessed by importing the environment object into any Angular service or component with the line `import { environment } from '@environments/environment'` and accessing properties on the `environment` object, see the user service for an example.

```
export const environment = {
    production: false,
    apiUrl: 'http://localhost:4000'
};
```

Back to top

Main Index Html File

Path: /src/index.html
The main index.html file is the initial page loaded by the browser that kicks everything off. The Angular CLI (with Webpack under the hood) bundles all of the compiled javascript files together and injects them into the body of the index.html page so the scripts can be loaded and executed by the browser.

```
<!DOCTYPE html>
<html>
<head>
    <base href="/" />
    <title>Angular 10 - Role Based Authorization Tutorial & Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- bootstrap css -->
    <link href="//netdna.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <app>Loading...</app>
</body>
</html>
```

Back to top

Main (Bootstrap) File

Path: /src/main.ts

The main file is the entry point used by angular to launch and bootstrap the application.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
    enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
    .catch(err => console.error(err));
```

Back to top

Polyfills

Path: /src/polyfills.ts

Some features used by Angular 10 are not yet supported natively by all major browsers, polyfills are used to add support for features where necessary so your Angular 10 application works across all major browsers.

This file is generated by the Angular CLI when creating a new project with the `ng new` command, I've excluded the comments in the file for brevity.

```
import 'zone.js/dist/zone';
```

Back to top

Global LESS/CSS Styles

Path: /src/styles.less
The global styles file contains LESS/CSS styles that are applied globally throughout the application.

```
/* You can add global styles to this file, and also import other style files */
a { cursor: pointer }
```

Back to top

Package.json

Path: /package.json
The package.json file contains project configuration information including package dependencies that get installed when you run `npm install` and scripts that are executed when you run `npm start` or `npm run build` etc. Full documentation is available at https://docs.npmjs.com/files/package.json (https://docs.npmjs.com/files/package.json).

```
{
    "name": "angular-role-based-authorization-example",
    "version": "0.0.0",
    "scripts": {
        "ng": "ng",
        "start": "ng serve --open",
        "build": "ng build",
        "test": "ng test",
        "lint": "ng lint",
        "e2e": "ng e2e"
    },
    "private": true,
    "dependencies": {
        "@angular/animations": "^10.1.0",
        "@angular/common": "^10.1.0",
        "@angular/compiler": "^10.1.0",
        "@angular/core": "^10.1.0",
        "@angular/forms": "^10.1.0",
        "@angular/platform-browser": "^10.1.0",
        "@angular/platform-browser-dynamic": "^10.1.0",
        "@angular/router": "^10.1.0",
        "rxjs": "^6.6.0",
        "tslib": "^2.0.0",
        "zone.js": "^0.10.2"
    },
    "devDependencies": {
        "@angular-devkit/build-angular": "^0.1001.0",
        "@angular/cli": "^10.1.0",
        "@angular/compiler-cli": "^10.1.0",
        "@types/node": "^12.11.1",
        "@types/jasmine": "^3.5.0",
        "@types/jasminewd2": "^2.0.3",
        "codelyzer": "^6.0.0",
        "jasmine-core": "^3.6.0",
        "jasmine-spec-reporter": "^5.0.0",
        "karma": "^5.0.0",
        "karma-chrome-launcher": "^3.1.0",
        "karma-coverage-istanbul-reporter": "^3.0.2",
        "karma-jasmine": "^4.0.0",
        "karma-jasmine-html-reporter": "^1.5.0",
        "protractor": "^7.0.0",
        "ts-node": "^8.3.0",
        "tslint": "^6.1.0",
        "typescript": "^4.0.2"
    }
}
```

Back to top

TypeScript tsconfig.base.json

Path: /tsconfig.base.json

The tsconfig.base.json file contains the base TypeScript compiler configuration for all projects in the Angular workspace, it configures how the TypeScript code will be compiled / transpiled into JavaScript that is understood by the browser. For more info see https://angular.io/config/tsconfig (https://angular.io/config/tsconfig).

Most of the file is unchanged from when it was generated by the Angular CLI, only the `paths` property has been added to map the `@app` and `@environments` aliases to the `/src/app` and `/src/environments` directories. This allows imports to be relative to the app and environments folders by prefixing import paths with aliases instead of having to use long relative paths (e.g. `import MyComponent from '@app/MyComponent'` instead of `import MyComponent from '../../../MyComponent'`).

```
{
    "compileOnSave": false,
    "compilerOptions": {
        "baseUrl": "./",
        "outDir": "./dist/out-tsc",
        "sourceMap": true,
        "declaration": false,
        "downlevelIteration": true,
        "experimentalDecorators": true,
        "moduleResolution": "node",
        "importHelpers": true,
        "target": "es2015",
        "module": "es2020",
        "lib": [
            "es2018",
            "dom"
        ],
        "paths": {
            "@app/*": ["src/app/*"],
            "@environments/*": ["src/environments/*"]
        }
    }
}
```

Back to top

Subscribe or Follow Me For Updates

Subscribe to my YouTube channel or follow me on Twitter or GitHub to be notified when I post new content.

- Subscribe on YouTube at https://www.youtube.com/c/JasonWatmore (https://www.youtube.com/c/JasonWatmore?sub_confirmation=1)
- Follow me on Twitter at https://twitter.com/jason_watmore (https://twitter.com/jason_watmore)
- Follow me on GitHub at https://github.com/cornflourblue (https://github.com/cornflourblue)
- Feeds formats available: RSS (/rss), Atom (/atom), JSON (/json)
