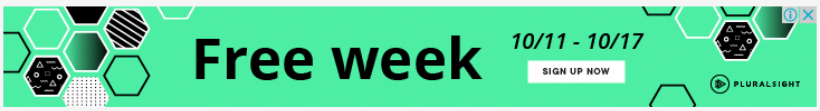
PUBLISHED: NOVEMBER 16 2018

Angular 7 - JWT Authentication Example & Tutorial



Tutorial built with **Angular 7.2.14** and **Webpack 4.30**

Other versions available:

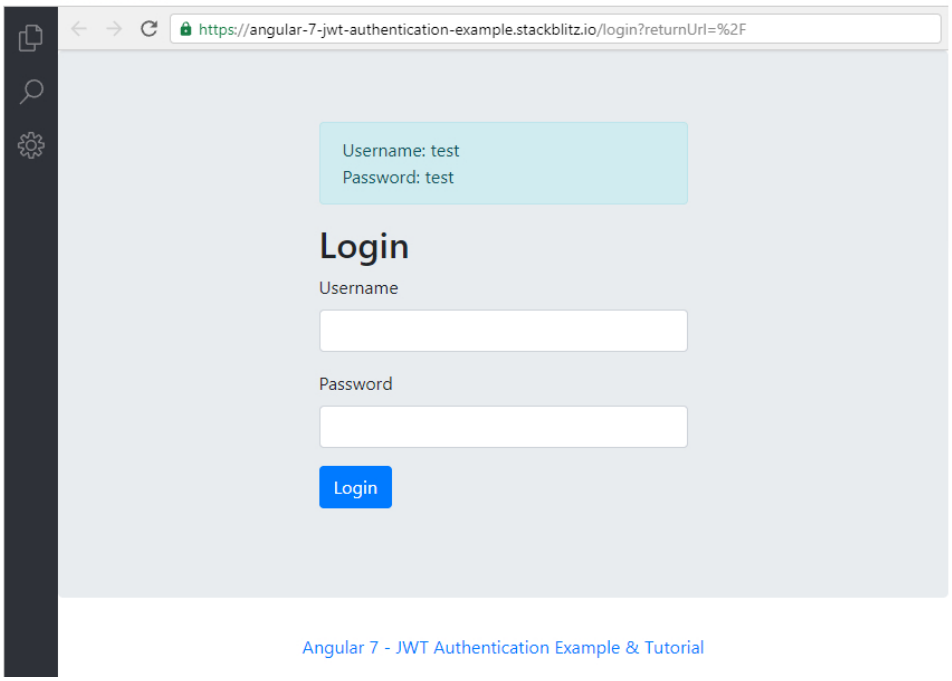
- **Angular:** Angular 10, 9, 8, 6, 2/5
- **React:** React + Recoil, React + Redux, React + RxJS
- **Vue:** Vue.js + Vuex
- **Next.js:** Next.js 11
- **AngularJS:** AngularJS

The following is a custom example and tutorial on how to setup a simple login page using Angular 7 and JWT authentication. For an extended example that includes role based access control check out Angular 7 - Role Based Authorization Tutorial with Example.

Webpack 4 is used to compile and bundle all the project files, and styling of the example is done with Bootstrap 4.

The tutorial code is available on GitHub at https://github.com/cornflourblue/angular-7-jwt-authentication-example

Here it is in action:



Angular 7 - JWT Authentication Example & Tutorial

(See on StackBlitz at https://stackblitz.com/edit/angular-7-jwt-authentication-example)

Update History:

- 02 Aug 2019 - Updated JWT Interceptor to only add auth token for requests to the configured api url
- 06 May 2019 - Updated tutorial to **Angular 7.2.14**, **Webpack 4.30** and **core-js 3.0.1**
- 16 Nov 2018 - Built tutorial with **Angular 7.0.4** and **Webpack 4.25**

Running the Angular 7 JWT Login Tutorial Example Locally

The tutorial example uses Webpack 4 to transpile the TypeScript code and bundle the Angular 7 modules together, and the webpack dev server is used as the local web server, to learn more about using webpack with TypeScript you can check out the webpack docs.

1. Install NodeJS and NPM from https://nodejs.org/en/download/.
2. Download or clone the tutorial project source code from https://github.com/cornflourblue/angular-7-jwt-authentication-example
3. Install all required npm packages by running `npm install` 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.

Running the Tutorial Example with a Real Backend API

The Angular 7 JWT example app uses a fake / mock backend by default so it can run in the browser without a real api, to switch to a real backend api you just have to remove or comment out the line below the comment `// provider used to create fake backend` located in the `/src/app/app.module.ts` file.

You can build your own backend api or start with one of the below options:

- To run the Angular 7 JWT auth example with a real backend API built with **NodeJS** follow the instructions at NodeJS - JWT Authentication Tutorial with Example API
- For a real backend API built with **ASP.NET Core 2.1** follow the instructions at ASP.NET Core 2.1 - JWT Authentication Tutorial with Example API

Angular 7 Tutorial Project Structure

The project and code structure of the tutorial mostly follows the best practice recommendations in the official Angular Style Guide, with a few of my own tweaks here and there.

Each feature has it's own folder (home & login), other shared/common code such as services, models, guards 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 instead of the full module path and to enable importing multiple modules in a single import (e.g. `import { AuthenticationService, UserService } from '../_services'`).

A path alias '@' has been configured in the tsconfig.json and webpack.config.js that maps to the

'/src/app' directory. This allows imports to be relative to the '/src/app' folder by prefixing the import path with '@', removing the need to use long relative paths like `import MyComponent from '../../../MyComponent'`.

Here's the tutorial project structure:

- src
  - app
    - _guards
      - auth.guard.ts
      - index.ts
    - _helpers
      - error.interceptor.ts
      - fake-backend.ts
      - jwt.interceptor.ts
      - index.ts
    - _models
      - user.ts
      - index.ts
    - _services
      - authentication.service.ts
      - user.service.ts
      - index.ts
    - home
      - home.component.html
      - home.component.ts
      - index.ts
    - login
      - login.component.html
      - login.component.ts
      - index.ts
    - app.component.html
    - app.component.ts
    - app.module.ts
    - app.routing.ts
  - index.html
  - main.ts
  - polyfills.ts
  - typings.d.ts
- package.json
- tsconfig.json
- webpack.config.js

Below are brief descriptions and the code for the main files of the example authentication application, all the tutorial code is available in the github project linked at the top of the post.

Angular 7 Auth Guard

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

The auth guard is an angular route guard that's used to prevent unauthenticated 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 returns `true` from the `canActivate()` method, 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.ts

to protect the home page route.

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

import { AuthenticationService } from '@/_services';

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

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const currentUser = this.authenticationService.currentUserValue;
        if (currentUser) {
            // logged in 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

Angular 7 Http 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 response the user is automatically logged out of the application, all other errors are re-thrown to be caught by the calling service so an alert can be displayed to the user.

It's implemented using the HttpInterceptor class that was introduced in Angular 4.3 as part of the new HttpClientModule. By extending the HttpInterceptor class you can create a custom interceptor to catch all error responses from the server 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 '@/_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 (err.status === 401) {
                // auto logout if 401 response returned from api
                this.authenticationService.logout();
                location.reload(true);
            }

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

Back to top

Angular 7 Fake Backend Provider

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

The fake backend provider enables the example to run without a backend / backendless, I created it so I could focus the example and tutorial just on the angular code, and also so it would work on StackBlitz.

It's implemented using the HttpInterceptor class that was introduced in Angular 4.3 as part of the new HttpClientModule. By extending the HttpInterceptor class you can create a custom interceptor to modify http requests before they get sent to the server. In this case the FakeBackendInterceptor intercepts certain requests based on their URL and provides a fake response instead of going to the server.

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

import { User } from '@/_models';

@Injectable()
export class FakeBackendInterceptor implements HttpInterceptor {
    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const users: User[] = [
            { id: 1, username: 'test', password: 'test', firstName: 'Test', lastName: 'User' }
        ];

        const authHeader = request.headers.get('Authorization');
        const isLoggedIn = authHeader && authHeader.startsWith('Bearer fake-jwt-token');

        // wrap in delayed observable to simulate server api call
        return of(null).pipe(mergeMap(() => {

            // authenticate - public
            if (request.url.endsWith('/users/authenticate') && request.method === 'POST') {
                const user = users.find(x => x.username === request.body.username && x.password === request.
                if (!user) return error('Username or password is incorrect');
                return ok({
                    id: user.id,
                    username: user.username,
                    firstName: user.firstName,
                    lastName: user.lastName,
                    token: `fake-jwt-token`
                });
            }

            // get all users
            if (request.url.endsWith('/users') && request.method === 'GET') {
                if (!isLoggedIn) return unauthorised();
                return ok(users);
            }

            // pass through any requests not handled above
            return next.handle(request);
        }))
        // call materialize and dematerialize to ensure delay even if an error is thrown (https://github.com
        .pipe(materialize())
        .pipe(delay(500))
        .pipe(dematerialize());

        // private helper functions

        function ok(body) {
            return of(new HttpResponse({ status: 200, body }));
        }

        function unauthorised() {
            return throwError({ status: 401, error: { message: 'Unauthorised' } });
        }

        function error(message) {
            return throwError({ status: 400, error: { message } });
        }
    }
}

export let fakeBackendProvider = {
```

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

Back to top

Angular 7 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.

It's implemented using the HttpInterceptor class that was introduced in Angular 4.3 as part of the new HttpClientModule. By extending the HttpInterceptor class 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 { AuthenticationService } from '@/_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 currentUser = this.authenticationService.currentUserValue;
        const isLoggedIn = currentUser && currentUser.token;
        const isApiUrl = request.url.startsWith(config.apiUrl);
        if (isLoggedIn && isApiUrl) {
            request = request.clone({
                setHeaders: {
                    Authorization: `Bearer ${currentUser.token}`
                }
            });
        }

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

Back to top

Angular 7 User Model

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

The user model is a small class that defines the properties of a user.

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

Back to top

Angular 7 JWT Authentication Service

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

The JWT authentication service is used to login and logout of the application, to login it posts the users credentials to the api and checks the response for a JWT token, if there is one it means authentication was successful so the user details are added to local storage with the token. The token is used by the JWT interceptor above to set the authorization header of http requests made to secure api endpoints.

The logged in user details are stored in local storage so the user will stay logged in if they refresh the browser and also between browser sessions until they logout. If you don't want the user to stay logged in between refreshes or sessions the behaviour could easily be changed by storing user details somewhere less persistent such as session storage which would persist between refreshes but not browser sessions, or in a private variable in the authentication service which would be cleared when the browser is refreshed.

There are two properties exposed by the authentication service for accessing the currently logged in user. The `currentUser` observable can be used when you want a component to reactively update when a user logs in or out, for example in the app.component.ts so it can show/hide the main nav bar when the user logs in/out. The `currentUserValue` property can be used when you just want to get the current value of the logged in user but don't need to reactively update when it changes, for example in the auth.guard.ts which restricts access to routes by checking if the user is currently logged in.

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

import { User } from '@/_models';

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

    constructor(private http: HttpClient) {
        this.currentUserSubject = new BehaviorSubject<User>(JSON.parse(localStorage.getItem('currentUser')))
        this.currentUser = this.currentUserSubject.asObservable();
    }

    public get currentUserValue(): User {
        return this.currentUserSubject.value;
    }

    login(username: string, password: string) {
        return this.http.post<any>(`${config.apiUrl}/users/authenticate`, { username, password })
            .pipe(map(user => {
                // login successful if there's a jwt token in the response
                if (user && user.token) {
                    // store user details and jwt token in local storage to keep user logged in between page
                    localStorage.setItem('currentUser', JSON.stringify(user));
                    this.currentUserSubject.next(user);
                }

                return user;
            }));
    }

    logout() {
        // remove user from local storage to log user out
        localStorage.removeItem('currentUser');
        this.currentUserSubject.next(null);
    }
}
```

Back to top

Angular 7 User Service

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

The user service contains a method for getting all users from the api, I included it to demonstrate accessing a secure api endpoint with the http authorization header set after logging in to the application, the auth header is set with a JWT token with the JWT Interceptor above. The secure endpoint in the example is a fake one implemented in the fake backend provider above.

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

import { User } from '@/_models';

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

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

Back to top

Angular 7 Home Component Template

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

The home component template contains html and angular 7 template syntax for displaying a simple welcome message and a list of users from a secure api endpoint.

```
<h1>Home</h1>
<p>You're logged in with Angular 7 & JWT!!</p>
<div>
    Users from secure api end point:
    <ul>
        <li *ngFor="let user of users">{{user.firstName}} {{user.lastName}}</li>
    </ul>
</div>
```

Back to top

Angular 7 Home Component

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

The home component defines an angular 7 component that gets all users from the user service and makes them available to the template via a `users` array property.

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

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

@Component({ templateUrl: 'home.component.html' })
export class HomeComponent {
    users: User[] = [];

    constructor(private userService: UserService) { }

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

```
        });
    }
}
```

Back to top

Angular 7 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 check out Angular 7 - Reactive Forms Validation Example.

```
<div class="alert alert-info">
    Username: test<br />
    Password: test
</div>
<h2>Login</h2>
<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': submit
        <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': su
        <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
            <div *ngIf="f.password.errors.required">Password is required</div>
        </div>
    </div>
    <div class="form-group">
        <button [disabled]="loading" class="btn btn-primary">Login</button>
        <img *ngIf="loading" class="pl-2" src="
    </div>
    <div *ngIf="error" class="alert alert-danger">{{error}}</div>
</form>
```

Back to top

Angular 7 Login Component

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

The login component uses the authentication service to login and logout of the application. It automatically logs the user out when it initializes (ngOnInit) so the login page can also be used to logout.

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]="loginForm"` directive.

```
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 '@/_services';

@Component({ templateUrl: 'login.component.html' })
export class LoginComponent implements OnInit {
```

```
    loginForm: FormGroup;
    loading = false;
    submitted = false;
    returnUrl: string;
    error = '';

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private authenticationService: AuthenticationService
    ) { }

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

        // reset login status
        this.authenticationService.logout();

        // get return url from route parameters or default to '/'
        this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
    }

    // 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(
                data => {
                    this.router.navigate([this.returnUrl]);
                },
                error => {
                    this.error = error;
                    this.loading = false;
                });
    }
}
```

Back to top

Angular 7 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.

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

<!-- main app container -->
<div class="jumbotron">
    <div class="container">
        <div class="row">
            <div class="col-md-6 offset-md-3">
                <router-outlet></router-outlet>
            </div>
        </div>
    </div>
</div>
```

Back to top

Angular 7 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 `currentUser` 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.

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

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

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

    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) {
        this.authenticationService.currentUser.subscribe(x => this.currentUser = x);
    }

    logout() {
        this.authenticationService.logout();
        this.router.navigate(['/login']);
    }
}
```

Back to top

Angular 7 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 7 modules check out this page 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 { routing } from './app.routing';

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

@NgModule({
```

```
    imports: [
        BrowserModule,
        ReactiveFormsModule,
        HttpClientModule,
        routing
    ],
    declarations: [
        AppComponent,
        HomeComponent,
        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

Angular 7 App Routing

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

The app routing file defines the routes of the application, each route contains a path and associated component. The home route is secured by passing the AuthGuard to the `canActivate` property of the route.

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

import { HomeComponent } from './home';
import { LoginComponent } from './login';
import { AuthGuard } from './_guards';

const appRoutes: Routes = [
    {
        path: '',
        component: HomeComponent,
        canActivate: [AuthGuard]
    },
    {
        path: 'login',
        component: LoginComponent
    },

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

export const routing = RouterModule.forRoot(appRoutes);
```

Back to top

Angular 7 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. Webpack bundles all of the javascript files together and injects them into the body of the index.html page so the scripts get loaded and executed by the browser.

```
<!DOCTYPE html>
<html>
```

```
<head>
    <base href="/" />
    <title>Angular 7 - JWT Authentication Tutorial & Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- bootstrap css -->
    <link href="//netdna.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" rel="stylesheet" />

    <style>
        a { cursor: pointer }
    </style>
</head>
<body>
    <app>Loading...</app>
</body>
</html>
```

Back to top

Angular 7 Main (Bootstrap) File

Path: /src/main.ts

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

```
import './polyfills';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

Back to top

Angular 7 Polyfills

Path: /src/polyfills.ts

Some features used by Angular 7 are not yet supported natively by all major browsers, polyfills are used to add support for features where necessary so your Angular 7 application works across all major browsers.

```
import 'core-js/features/reflect';
import 'zone.js/dist/zone';
```

Back to top

Angular 7 Custom Typings File

Path: /src/typings.d.ts

A custom typings file is used to declare types that are created outside of your angular application, so the TypeScript compiler is aware of them and doesn't give you errors about unknown types. This typings file contains a declaration for the global `config` object that is created by webpack (see webpack.config.js below).

```
// so the typescript compiler doesn't complain about the global config object
declare var config: any;
```

Back to top

npm package.json

Path: /package.json

The package.json file contains project configuration information including package dependencies which

get installed when you run `npm install`. Full documentation is available on the npm docs website.

```
{
    "name": "angular-7-jwt-authentication-example",
    "version": "1.0.0",
    "repository": {
        "type": "git",
        "url": "https://github.com/cornflourblue/angular-7-jwt-authentication-example.git"
    },
    "scripts": {
        "build": "webpack --mode production",
        "start": "webpack-dev-server --mode development --open"
    },
    "license": "MIT",
    "dependencies": {
        "@angular/common": "^7.0.1",
        "@angular/compiler": "^7.0.1",
        "@angular/core": "^7.0.1",
        "@angular/forms": "^7.0.1",
        "@angular/platform-browser": "^7.0.1",
        "@angular/platform-browser-dynamic": "^7.0.1",
        "@angular/router": "^7.0.1",
        "core-js": "^3.0.1",
        "rxjs": "^6.3.3",
        "zone.js": "^0.9.1"
    },
    "devDependencies": {
        "@types/node": "^12.0.0",
        "angular2-template-loader": "^0.6.2",
        "html-webpack-plugin": "^3.2.0",
        "raw-loader": "^1.0.0",
        "ts-loader": "^5.2.2",
        "typescript": "^3.1.3",
        "webpack": "^4.24.0",
        "webpack-cli": "^3.1.2",
        "webpack-dev-server": "^3.1.10"
    }
}
```

Back to top

TypeScript tsconfig.json

Path: /tsconfig.json

The tsconfig.json file configures how the TypeScript compiler will convert TypeScript into JavaScript that is understood by the browser. More information is available on the TypeScript docs.

```
{
    "compilerOptions": {
        "emitDecoratorMetadata": true,
        "experimentalDecorators": true,
        "lib": [
            "es2015",
            "dom"
        ],
        "module": "commonjs",
        "moduleResolution": "node",
        "noImplicitAny": false,
        "sourceMap": true,
        "suppressImplicitAnyIndexErrors": true,
        "target": "es5",
        "baseUrl": "src",
        "paths": {
            "@/*": [
                "app/*"
            ]
        }
    }
}
```

Back to top

Webpack 4 Config

Path: /webpack.config.js

Webpack 4 is used to compile and bundle all the project files so they're ready to be loaded into a browser, it does this with the help of loaders and plugins that are configured in the webpack.config.js file. For more info about webpack check out the webpack docs.

This is a minimal webpack.config.js for bundling an Angular 7 application, it compiles TypeScript files

using `ts-loader`, loads angular templates with `raw-loader`, and injects the bundled scripts into the body of the index.html page using the `HtmlWebpackPlugin`. It also defines a global config object with the plugin `webpack.DefinePlugin`.

A path alias '@' is configured in the webpack.config.js and the tsconfig.json that maps to the '/src/app' directory. This allows imports to be relative to the '/src/app' folder by prefixing the import path with '@', removing the need to use long relative paths like `import MyComponent from '../../../MyComponent'`.

```
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');

module.exports = {
    entry: './src/main.ts',
    module: {
        rules: [
            {
                test: /\.ts$/,
                use: ['ts-loader', 'angular2-template-loader'],
                exclude: /node_modules/
            },
            {
                test: /\.(html|css)$/,
                loader: 'raw-loader'
            },
        ]
    },
    resolve: {
        extensions: ['.ts', '.js'],
        alias: {
            '@': path.resolve(__dirname, 'src/app/'),
        }
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: './src/index.html',
            filename: 'index.html',
            inject: 'body'
        }),
        new webpack.DefinePlugin({
            // global app config object
            config: JSON.stringify({
                apiUrl: 'http://localhost:4000'
            })
        })
    ],
    optimization: {
        splitChunks: {
            chunks: 'all',
        },
        runtimeChunk: true
    },
    devServer: {
        historyApiFallback: true
    }
};
```

Back to top

Subscribe or Follow Me For Updates

Subscribe to my YouTube channel or follow me on Twitter, Facebook or GitHub to be notified when I

post new content.

- Subscribe on YouTube at https://www.youtube.com/JasonWatmore
- Follow me on Twitter at https://twitter.com/jason_watmore
- Follow me on Facebook at https://www.facebook.com/Jason-Watmores-Blog
- Follow me on GitHub at https://github.com/cornflourblue
- Feed formats available: RSS, Atom, JSON

Other than coding...

I'm currently attempting to travel around Australia by motorcycle with my wife Tina on a pair of Royal Enfield Himalayans. You can follow our adventures on YouTube, Instagram and Facebook.

- Subscribe on YouTube at https://www.youtube.com/TinaAndJason
- Follow us on Instagram at https://www.instagram.com/tinaandjason
- Follow us on Facebook at https://www.facebook.com/TinaAndJasonVlog

Need Some Angular 7 Help?

Search fiverr to find help quickly from experienced Angular 7 developers.

Tags: Angular 7, Angular 2, TypeScript, Authentication and Authorization, Security, JWT

Share:

More Angular 7 Posts

- Angular - HTTP POST Request Examples
- Angular - HTTP GET Request Examples
- Angular + Webpack - How to add global CSS styles to Angular with webpack
- Angular 7 Tutorial Part 7 - Migrating to an Angular CLI Project
- Angular 7 Tutorial Part 6 - Home Page & Alert Component
- Angular 7 Tutorial Part 5 - Registration Form & User Service
- Angular 7 Tutorial Part 4 - Login Form, Authentication Service & Route Guard
- Angular 7 - Mock Backend Example for Backendless Development
- Angular 7 Tutorial Part 3 - Add Routing & Multiple Pages
- Angular 7 Tutorial Part 2 - Create Base Project Structure & Webpack Config

Show more...

What do you think?
78 Responses

Upvote Funny Love Surprised Angry Sad

54 Comments Jason Watmore's Blog Disqus' Privacy Policy Login

Recommend 5 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

Bilal Elmursi • 2 years ago • edited
Wow very nice tutorial
4 | Reply • Share ›

Waleed Tayseer El-Tahtawy • 3 years ago
Thank you so much Jason, I've implemented the whole authentication thing!! thanks to you with you I wouldn't have done it :)
Its simple, clean and most importantly Works!! :)
3 | Reply • Share ›

Sayidazim Mahmudov • 2 years ago
Great tutorial! Thanks, it helped me a lot
1 | Reply • Share ›

felipeboss • 2 years ago
Helped me a lot. Thanks!
1 | Reply • Share ›

ZhouSir • 3 years ago
Hi Jason, I just wanna ask what if I insert an item called 'currentUser' into the localStorage manually. In this way the home page is not guarded and can be accessed. So what's the point of guard. Thanks!
1 | Reply • Share ›

Jason Watmore Mod ➔ ZhouSir • 2 years ago
Hi ZhouSir, manually editing localStorage will give you access to the client-side home page template, but you already have access to that because it's downloaded to the browser as part of the SPA, client side "security" is really more about UX than real security.

Real security is implemented on the server side and any sensitive data should be protected there, manually editing localStorage won't give you access to server side data because you need a valid JWT that is generated and signed on the server.

Cheers,
Jason
| Reply • Share ›

Wesley Christelis ➔ ZhouSir • 3 years ago
I am sure this is just for brevity. Normally you would validate the token on the guard.
| Reply • Share ›

Bernardo Mondragon Brozon ➔ Wesley Christelis • 2 years ago
Have you managed to come up with a solution to this? What would be the best way to protect the home page template (for better UX porpuses)? I also think that the auth guard is not doing a good job returning ture only if there is a value for currentUser in the localStorage.
| Reply • Share ›

Mhar Bercasio Daniel • 2 years ago
hi sir how to resolve **@/_services** on **visual studio code**? the code is working but the vs code editor **ca'nt find the module**. thanks :D
| Reply • Share ›

Merve Yemenli ➔ Mhar Bercasio Daniel • 2 years ago
///import { AuthService } from '../../services/auth.service';
you can show the reference like this. (ClientApp->src->app ->services->auth.service)
| Reply • Share ›

Yasser Kantour • 2 years ago
Nice post, however there can be a huge security breach there. You MUST not set the auth header for every intercepted http request. There should be a control to which URL's the token is sent to
| Reply • Share ›

Jason Watmore Mod ➔ Yasser Kantour • 2 years ago
Hi Yasser, that's a good point, I've just updated the JWT interceptor to only add the auth header if the user is logged in and the request is to the configured api url.

Cheers,
Jason
| Reply • Share ›

ivaylo • 2 years ago
Hey, I just want to ask why are you using observable when you are already injecting the service? Why not take the user directly in the nav like *ngIf = "authenticationService.currentUserValue" - or even just a boolean is auth *ngIf = "authenticationService.isAuth"? and when you logout your nav *ngIf is anyway binded to the service's currentUser or boolean and when at logout you make it null or false it will update the nav? Do you gain something with the observable way?
| Reply • Share ›

Jason Watmore Mod ➔ ivaylo • 2 years ago
Hi Ivaylo, I prefer to encapsulate service functionality inside the component and only expose the bits that the template needs. But you're right you could also make the authentication service public and use it directly in the template.

Cheers,
Jason

Jason

Reply • Share ›

Hectorín Sotowsky • 2 years ago

Nice tutorial, but ¿how Can I redirect after authenticated? and ¿block/hide elements if authenticated?

Reply • Share ›

Jason Watmore Mod ➚ Hectorín Sotowsky • 2 years ago • edited

Hi Hectorin, both of these things are already done in the example, the main nav bar is hidden until the user is authenticated, and the auth guard includes the returnUrl in the queryParams when redirecting to the login page with this code:

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

Cheers,
Jason

Reply • Share ›

Prashantha CG • 2 years ago

I got error like ./src/app/_guards/auth.guard.ts
Module not found: Error: Can't resolve '@/services' in 'C:\Users\Prashantha\Desktop\my-new-project\src\app_guards'.
where am using this line in pages like "import { AuthenticationService } from '@/services';" shows like above mentioned error.
thanks.

Reply • Share ›

bzglx • 2 years ago

Hello!

How to get currentUserValue from server using ajax request?

Need redirect to login page if customer not logged on server.

Thanks!

Reply • Share ›

Prince Painadath • 2 years ago • edited

Hi Jason,
I commented the fakeBackendProvider line from app.module.ts. But when i call the login function in authentication.service.ts its not showing any error message if the user value return's null. I have created proper backend with asp.net web api that create token. Is I miss anything?

Reply • Share ›

Jason Watmore Mod ➚ Prince Painadath • 2 years ago

Hi Prince, make sure your web api is returning a 400 response with an error message in the body for failed authentication, like this line from the compatible asp.net core api that I linked to near the top of the tutorial - https://github.com/cornflourblue/aspnet-core-jwt-authentication-api/blob/master/Controllers/UsersController.cs#L27.

Cheers,
Jason

Reply • Share ›

Francisco Ramon • 2 years ago

hey man, why do not you use ng-cli for the project, any reason for that?

Reply • Share ›

Jason Watmore Mod ➚ Francisco Ramon • 2 years ago • edited

Hi Francisco, I like using webpack directly to keep the project lean so it only contains the code it needs.

If you prefer using the Angular CLI it's not hard to migrate the code. I've created a few Angular CLI versions of example projects and pretty much all I had to do was copy the source code into a new Angular CLI project and it worked (e.g. https://jasonwatmore.com/post/2018/10/29/angular-7-user-registration-and-login-example-tutorial)

Cheers,
Jason

Reply • Share ›

Mauro • 2 years ago

Hi, thanks for this nice tutorial, have you tried upgrade to core-js 3.0.1 ?

Reply • Share ›

Jason Watmore Mod ➚ Mauro • 2 years ago

Hi Mauro, I haven't with this JWT example but I did use core-js 3.0.1 recently in another post - Angular 7 Tutorial Part 2 - Create Base Project Structure & Webpack Config

.

The main change I needed to make was to the polyfills file:

```
import 'core-js/features/reflect';
import 'zone.js/dist/zone';
```

Cheers,
Jason

Reply • Share ›

Mauro ➚ Jason Watmore • 2 years ago

Hi Jason, thanks for reply, can you update your github or post updated polyfills.ts & tsconfig.json ?
I've tryed but still does'nt works for me..

Reply • Share ›

Jason Watmore Mod → Mauro • 2 years ago

No problem, just updated to core-js 3.01 and updated a couple of other packages as well, the only update I needed to make was to the polyfills file, you can see the commit at https://github.com/cornflou...

Cheers,
Jason

Reply • Share ›

aymen023 • 2 years ago

Hello i get this error in app.module.ts :
Uncaught Error: Invalid provider for InjectionToken HTTP_INTERCEPTORS. useClass cannot be undefined.
Usually it happens when:
1. There's a circular dependency (might be caused by using index.ts (barrel) files).
2. Class was used before it was declared. Use forwardRef in this case.

Reply • Share ›

Rob • 2 years ago • edited

Very nice tutorial. Thanks! Just had a little issue when used in combination of resolver which as well redirect to an error page for example, there is a simultanous redirection to login page and error page.. result is random landing on error or login page depending of browser.
I figured out how to resolve this problem. I modified the error interceptor in this way :

```
if (err.status === 401) {
// auto logout if 401 response returned from api
this.authenticationService.logout();
location.reload(true);
return EMPTY;
} else {
const error = err.error.message || err.statusText;
return throwError(error);
}
```

Actually if 401 error is handled by the interceptor which redirect to login page, it shouldn't throw an error anymore but I agree it depends of actions made on error.. in case of a simple log it can be valid.

Regards

Reply • Share ›

sidelinedigital • 3 years ago • edited

Hi, great tutorial. We are using a jwt access token, I'm a little stuck as to how/where we can decode the token and then store it for currentUser. We aren't keen on using a library for this.

Reply • Share ›

Elsa Gong • 3 years ago

Hi, thanks so much for your great tutorial, finally I can log in my project with a real API server, but I don't know how to use JWT authentication in other pages(still 401 Unauthorized), I noticed in /src/app/home/home.component.ts, you imported AuthenticationService, but you didn't use it only use UserService. Could you please help me figure out it? Thank you so much for any advice!🌹👼

Reply • Share ›

Jason Watmore Mod → Elsa Gong • 3 years ago

Hi Elsa, the JWT auth token is added to http request headers in the JWT Interceptor with the following code:

```
// add authorization header with jwt token if available
let currentUser = this.authenticationService.currentUserValue;
if (currentUser && currentUser.token) {
    request = request.clone({
        setHeaders: {
            Authorization: `Bearer ${currentUser.token}`
        }
    });
}
```

Cheers,
Jason

2 Reply • Share ›

Zafer Ka • 3 years ago

I can't use @ sign at imports like you.

import { User } from '@/models';

I haven't seen it before? Any trick?

Reply • Share ›

Jason Watmore Mod → Zafer Ka • 3 years ago

Hi Zafer,

The path alias '@' is configured in the webpack.config.js and the tsconfig.json that maps to the '/src/app' directory.

Cheers,
Jason

Reply • Share ›

mvslc • 3 years ago

I am getting the following error when I run "ng serve". My angular CLI version is 7.3.1. I tried upgrading it but it didn't take.

"The serve command requires to be run in an Angular project, but a project definition could not be found".

Reply • Share ›

Jason Watmore Mod → mvslc • 3 years ago

Hi **@mvslc**,

The example project uses webpack directly not angular cli, to start it run `npm start`.

Cheers,
Jason

Reply • Share ›

pavan raja • 3 years ago • edited

Hi,

As per my understanding,We have no need of fakeBackendProvider(fack-backend.ts) if I use any api or nodejs. pls let me know that is this understanding correct ?

Reply • Share ›

Jason Watmore Mod ➚ pavan raja • 3 years ago

Hi pavan, that's right if you have a real backend you can remove the fake one.

Cheers,
Jason

Reply • Share ›

suckerp • 3 years ago

What's the reason behind converting the BehaviorSubject into an Observable instead of making the BehaviorSubject public and use only the BehaviorSubject?

Reply • Share ›

Jason Watmore Mod ➚ suckerp • 3 years ago

Hi suckerp, the `BehaviorSubject` is private so only the authentication service has access to set the logged in status of the user, if it was public then any component could log the user in/out. The `Observable` is public so other components can subscribe to the currently logged in user.

Cheers,
Jason

3 Reply • Share ›

Rajesh • 3 years ago

I am getting the below error while call web api

code: 15
message: "Failed to set the 'responseType' property on 'XMLHttpRequest': The response type cannot be changed for synchronous requests made from a
name: "InvalidAccessError"

Reply • Share ›

Arka Ghosh • 3 years ago

Hi Jason,

Very nice tutorial. I am just facing one problem. I am not able to find the webpack.config.js file. I searched around in stackoverflow and found that modifying webpack is not supported as of Angular 6. If that is the case, then where should I define the apiUrl? I am completely new to Angular, so I am not sure if this is a stupid question.

Reply • Share ›

Jason Watmore Mod ➚ Arka Ghosh • 3 years ago

Hi Arka, it sounds like you might be using the angular-cli to build your project instead of webpack directly like in this example. If that's the case you should use the `environment.ts` file for the apiUrl. I posted a similar example that includes user registration here that also includes a version built with angular-cli. The specific section of the angular-cli version that contains the apiUrl is here.

Hope this helps

Cheers,
Jason

1 Reply • Share ›

Arka Ghosh ➚ Jason Watmore • 3 years ago

Thanks Jason. I followed the other tutorial and got it to work.

Best,
Arka

Reply • Share ›

J • 3 years ago

I believe the intercept signature for the ErrorInterceptor should be

intercept(request: HttpRequest<any>, next: HttpHandler): Observable<httpevent<any>>

Reply • Share ›

Jason Watmore Mod ➚ J • 3 years ago

Thanks for spotting that J, looks like I forgot to format that code in the post, just fixed it.

Cheers,
Jason

Reply • Share ›

Alejandro Lafourcade Despaigne • 3 years ago

Hello, thanks for the example. I have a question, if I want to implement the refresh token, how can I do it?

Reply • Share ›

Amol Chaware • 3 years ago • edited

can you please tell me, how to connect this application to the AWS user-pool. I want to store the username & other detail in my AWS user pool account & how we

will add mobile and email verification in a given application(at the time of registration).

Reply • Share ›

Josh • 3 years ago

When I put my own server in and change the api url, how do I return login errors? I got it to work with when it returns a token and logs in OK but when there's no token (eg. invalid credentials) how do I display the error w/out the fake backend?

My json returns an error with & message but I don't know how to "throw the error" if no token comes back.

2 • Reply • Share ›

Stefan Jacomeit • 3 years ago • edited

Hi Jason,

thanks for this tutorial. I am beginner of Angular and have create a new service for get some data. I do not understand, how the JwtInterceptor now works on my requests. If I am trying to use my new service the system redirect me back to the login. Both interceptors I have add in my app.module.ts as providers.

My Service (customer.service.ts):

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpErrorResponse } from '@angular/common/http';
import { Observable, of } from 'rxjs';
import { environment } from '../../environments/environment';
import {Customer} from '../_models/customer';

@Injectable({providedIn: 'root'})
export class CustomerService {

constructor(private http: HttpClient) { }

getCustomer(id): Observable<customer> {
```

see more

2 • Reply • Share ›

Chandan ➚ Stefan Jacomeit • 3 years ago

How good it is to store **UserToken** in **localstroage**?

Reply • Share ›

Load more comments

ABOUT

I'm a web developer in Sydney Australia and co-founder of Point Blank Development, I've been building websites and web applications in Sydney since 1998.

Other than coding, I'm currently attempting to travel around Australia by motorcycle with my wife Tina, you can follow our adventure on YouTube and Instagram.

Support me on: Patreon

Find me on:

Subscribe to Feed: RSS, Atom, JSON

Code Snippets

MONTHS

2021
- October (8)
- September (30)
- August (12)
- July (4)
- June (7)
- May (8)
- April (6)

2020
2019
2018
2017
2016
2015
2014
2013
2012
2011

TAGS

Alerts, Angular, Angular 10, Angular 11, Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular 8, Angular 9, Angular Directive, Angular UI Router, AngularJS, Animation, ASP.NET, ASP.NET Core, ASP.NET Web API, Authentication and Authorization, AWS, Axios, Azure, Basic Authentication, Blazor, Bootstrap, C#, Chai, CKEditor, CSS3, DDD, Deployment, Design Patterns, Dynamic LINQ, EF Core, ELMAH, ES6, Exchange, Facebook, Fetch, Fluent NHibernate, Formik, Google Analytics, Google API, Google Maps API, Google Plus, Heroku, HTML5, HTTP, IIS, Insecure Content, Instagram API, Ionic Framework, iOS, iPhone, JavaScript, jQuery, JWT, LinkedIn, LINQ, Login, MEAN Stack, MERN Stack, MEVN Stack, Mocha, Modal, MongoDB, Moq, MVC, MVC5, MySQL, .NET, NextJS, NGINX, ngMock, NHibernate, Ninject, NodeJS, npm, Pagination, Pinterest, Razor Pages, React, React Hook Form, React Hooks, Recoil, Redmine, Redux, Registration, Repository, RxJS, Security, Sequelize, Shell Scripting, Sinon, SinonJS, SSH, TDD, Terraform, Twitter, TypeScript, Ubuntu, Umbraco, Unit of Work, Unit Testing, URL Rewrite, Validation, Vanilla JS, VS Code, Vue, Vue 3, Vuex, Webpack, Windows Server 2008