@auth0/angular-jwt

2.1.1 • Public • Published 16 hours ago

Readme

1 Dependencies

69 Dependents

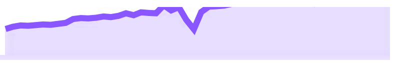
17 Versions

install

```
> npm i @auth0/angular-jwt
```

weekly downloads

69,675



version

2.1.1

license

MIT

open issues

130

pull requests

4

homepage

github.com

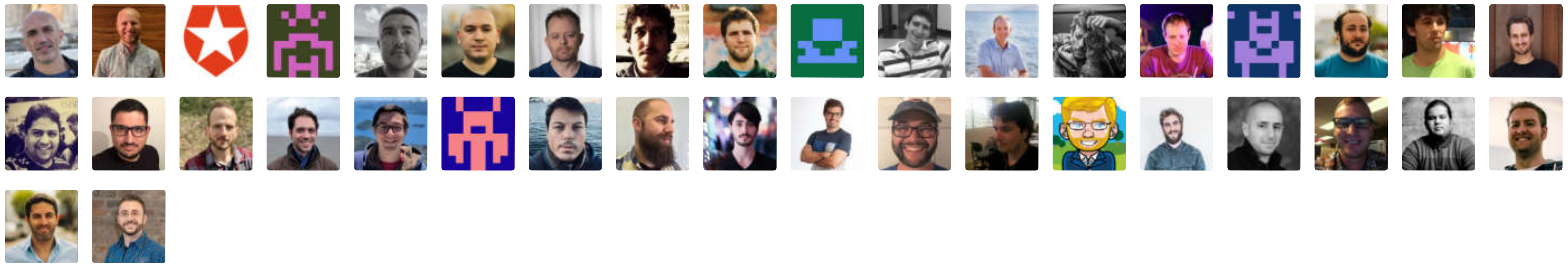
repository

github

last publish

16 hours ago

collaborators



Test with RunKit

Report a vulnerability

@auth0/angular-jwt

NOTE: This library is now at version 2 and is published on npm as `@auth0/angular-jwt`. If you're looking for the pre-v1.0 version of this library, it can be found in the `pre-v1.0` branch and on npm as `angular2-jwt`.

@auth0/angular-jwt v2 is to be used with Angular v6+ and RxJS v6+. For Angular v4.3 to v5+, use @auth0/angular-jwt v1

This library provides an `HttpInterceptor` which automatically attaches a **JSON Web Token** to `HttpClient` requests.

This library does not have any functionality for (or opinion about) implementing user authentication and retrieving JWTs to begin with. Those details will vary depending on your setup, but in most cases, you will use a regular HTTP request to authenticate your users and then save their JWTs in local storage or in a cookie if successful.

> **Note:** This library can only be used with Angular 4.3 and higher because it relies on an `HttpInterceptor` from Angular's `HttpClient`. This feature is not available on lower versions.

Installation

```
# installation with npm
npm install @auth0/angular-jwt

# installation with yarn
yarn add @auth0/angular-jwt
```

Usage: Standalone

If you are only interested in the JWT Decoder, and are not interested in extended injectable features, you can simply create an instance of the utility and use it directly:

```
import { JwtHelperService } from '@auth0/angular-jwt';

const helper = new JwtHelperService();

const decodedToken = helper.decodeToken(myRawToken);
const expirationDate = helper.getTokenExpirationDate(myRawToken);
const isExpired = helper.isTokenExpired(myRawToken);
```

Usage: Injection

Import the `JwtModule` module and add it to your imports list. Call the `forRoot` method and provide a `tokenGetter` function. You must also whitelist any domains that you want to make requests to by specifying a `whitelistedDomains` array.

Be sure to import the `HttpClientModule` as well.

```
import { JwtModule } from '@auth0/angular-jwt';
import { HttpClientModule } from '@angular/common/http';

export function tokenGetter() {
  return localStorage.getItem('access_token');
}
```

```
@NgModule({
  bootstrap: [AppComponent],
  imports: [
    // ...
    HttpClientModule,
    JwtModule.forRoot({
      config: {
        tokenGetter: tokenGetter,
        whitelistedDomains: ['example.com'],
        blacklistedRoutes: ['example.com/examplebadroute/']
      }
    })
  ]
})
export class AppModule {}
```

Any requests sent using Angular's `HttpClient` will automatically have a token attached as an `Authorization` header.

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

export class AppComponent {
  constructor(public http: HttpClient) {}

  ping() {
    this.http
      .get('http://example.com/api/things')
```

```
        .subscribe(data => console.log(data), err => console.log(err));
    }
  }
```

Configuration Options

`tokenGetter: function`

The `tokenGetter` is a function which returns the user's token. This function simply needs to make a retrieval call to wherever the token is stored. In many cases, the token will be stored in local storage or session storage.

```
// ...
JwtModule.forRoot({
  config: {
    // ...
    tokenGetter: () => {
      return localStorage.getItem('access_token');
    }
  }
});
```

`whitelistedDomains: array`

Authenticated requests should only be sent to domains you know and trust. Many applications make requests to APIs from multiple domains, some of which are not controlled by the developer. Since there is no way to know what the API being called will do with the information contained in the request, it is best to not send the user's token any and all APIs in a blind fashion.

List any domains you wish to allow authenticated requests to be sent to by specifying them in the the `whitelistedDomains` array. **Note that standard http port 80 and https port 443 requests don't require a port to be specified. A port is only required in the whitelisted host name if you are authenticating against a non-standard port e.g. localhost:3001**

```
// ...
JwtModule.forRoot({
  config: {
    // ...
    whitelistedDomains: ['localhost:3001', 'foo.com', 'bar.com']
  }
});
```

`blacklistedRoutes: array`

If you do not want to replace the authorization headers for specific routes, list them here. This can be useful if your initial auth route(s) are on a whitelisted domain and take basic auth headers.

```
// ...
JwtModule.forRoot({
  config: {
    // ...
    blacklistedRoutes: [
      'localhost:3001/auth/',
      'foo.com/bar/',
      /localhost:3001\/foo\/far.*/
    ] // strings and regular expressions
  }
});
```

Note: If requests are sent to the same domain that is serving your Angular application, you do not need to add that domain to the `whitelistedDomains` array. However, this is only the case if you don't specify the domain in the `Http` request.

For example, the following request assumes that the domain is the same as the one serving your app. It doesn't need to be whitelisted in this case.

```
this.http.get('/api/things')
  .subscribe(...)
```

However, if you are serving your API at the same domain as that which is serving your Angular app **and** you are specifying that domain in `Http` requests, then it **does** need to be whitelisted.

```
// Both the Angular app and the API are served at
// localhost:4200 but because that domain is specified
// in the request, it must be whitelisted
this.http.get('http://localhost:4200/api/things')
  .subscribe(...)
```

`headerName: string`

The default header name is `Authorization`. This can be changed by specifying a custom `headerName` which is to be a string value.

```
// ...
JwtModule.forRoot({
  config: {
    // ...
    headerName: 'Your Header Name'
  }
});
```

`authScheme: string`

The default authorization scheme is `Bearer` followed by a single space. This can be changed by specifying a custom `authScheme` which is to be a string.

```
// ...
JwtModule.forRoot({
  config: {
    // ...
    authScheme: 'Your Auth Scheme'
  }
});
```

`throwNoTokenError: boolean`

Setting `throwNoTokenError` to `true` will result in an error being thrown if a token cannot be retrieved with the `tokenGetter` function. Defaults to `false` .

```
// ...
```

```
JwtModule.forRoot({
  config: {
    // ...
    throwNoTokenError: true
  }
});
```

`skipWhenExpired: boolean`

By default, the user's JWT will be sent in `HttpClient` requests even if it is expired. You may choose to not allow the token to be sent if it is expired by setting `skipWhenExpired` to true.

```
// ...
JwtModule.forRoot({
  config: {
    // ...
    skipWhenExpired: true
  }
});
```

Using a Custom Options Factory Function

In some cases, you may need to provide a custom factory function to properly handle your configuration options. This is the case if your `tokenGetter` function relies on a service or if you are using an asynchronous storage mechanism (like Ionic's `Storage`).

Import the `JWT_OPTIONS` `InjectionToken` so that you can instruct it to use your custom factory function.

Create a factory function and specify the options as you normally would if you were using `JwtModule.forRoot` directly. If you need to use a service in the function, list it as a parameter in the function and pass it in the `deps` array when you provide the function.

```
import { JwtModule, JWT_OPTIONS } from '@auth0/angular-jwt';
import { TokenService } from './app.tokenservice';

// ...

export function jwtOptionsFactory(tokenService) {
  return {
    tokenGetter: () => {
      return tokenService.getAsyncToken();
    }
  }
}

// ...

@NgModule({
  // ...
  imports: [
    JwtModule.forRoot({
      jwtOptionsProvider: {
        provide: JWT_OPTIONS,
        useFactory: jwtOptionsFactory,
        deps: [TokenService]
      }
    })
  ],
  providers: [TokenService]
})
```

NOTE: If a `jwtOptionsFactory` is defined, then `config` is ignored. *Both configuration alternatives can't be defined at the same time.*

Configuration for Ionic 2+

The custom factory function approach described above can be used to get a token asynchronously with Ionic's `Storage` .

```
import { JwtModule, JWT_OPTIONS } from '@auth0/angular-jwt';
import { Storage } from '@ionic/storage';

export function jwtOptionsFactory(storage) {
  return {
    tokenGetter: () => {
      return storage.get('access_token');
    }
  }
}
```

```
// ...

@NgModule({
  // ...
  imports: [
    JwtModule.forRoot({
      jwtOptionsProvider: {
        provide: JWT_OPTIONS,
        useFactory: jwtOptionsFactory,
        deps: [Storage]
      }
    })
  ]
})
```

Configuration Options

`JwtHelperService: service`

This service contains helper functions:

isTokenExpired (old tokenNotExpired function)

```
import { JwtHelperService } from '@auth0/angular-jwt';
// ...
```

```
constructor(public jwtHelper: JwtHelperService) {}

ngOnInit() {
console.log(this.jwtHelper.isTokenExpired()); // true or false
}
```

getTokenExpirationDate

```
import { JwtHelperService } from '@auth0/angular-jwt';
// ...
constructor(public jwtHelper: JwtHelperService) {}

ngOnInit() {
console.log(this.jwtHelper.getTokenExpirationDate()); // date
}
```

decodeToken

```
import { JwtHelperService } from '@auth0/angular-jwt';
// ...
constructor(public jwtHelper: JwtHelperService) {}

ngOnInit() {
  console.log(this.jwtHelper.decodeToken(token)); // token
  }
```

What is Auth0?

Auth0 helps you to:

- Add authentication with multiple authentication sources, either social like **Google, Facebook, Microsoft Account, LinkedIn, GitHub, Twitter, Box, Salesforce, amont others**, or enterprise identity systems like **Windows Azure AD, Google Apps, Active Directory, ADFS or any SAML Identity Provider**.
- Add authentication through more traditional username/password databases.
- Add support for linking different user accounts with the same user.
- Support for generating signed Json Web Tokens to call your APIs and **flow the user identity** securely.
- Analytics of how, when and where users are logging in.
- Pull data from other sources and add it to the user profile, through JavaScript rules.

Create a free Auth0 account

1. Go to **Auth0** and click Sign Up.
2. Use Google, GitHub or Microsoft Account to login.

Issue Reporting

If you have found a bug or if you have a feature request, please report them at this repository issues section. Please do not report security vulnerabilities on the public GitHub issue tracker. The **Responsible Disclosure Program** details the procedure for disclosing security issues.

Author

Auth0

License

This project is licensed under the MIT license. See the **LICENSE** file for more info.

Keywords

angular angular 2 authentication jwt

HELP

Documentation

Resources

Support / Contact Us

Registry Status

Report Issues

npm Community Site

Security

ABOUT NPM

About npm, Inc

JavaScriptSurvey.com

Events

Jobs

Press

npm Weekly

Blog

Twitter

GitHub

TERMS & POLICIES

Terms of Use

Code of Conduct

Package Name Disputes

Privacy Policy

Reporting Abuse

Other policies