OAUTH 2.0 BASED TOKEN AUTHENTICATION LIBRARY FOR ANGULARJS IN TYPESCRIPT[*]

ANAND SINGH KUNWAR
13110
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
anandk@iitk.ac.in

Under the Guidance of
DR. T.V. PRABHAKAR
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
tvp@iitk.ac.in

OAuth 2.0 is one of the popular open standard authorization frameworks used. Javascript has become the language of choice for front-end in web development and with the upsurge of frameworks like AngularJS, the need for authentication libraries has risen. In this project, we develop an OAuth 2.0 based Token Authentication Module for AngularJS written in Typescript.

Keywords: AngularJS, OAuth2, Typescript

1. Introduction

OAuth 2.0 is an authorization protocol that has become the industry standard. It allows third-party applications to procure limited access an HTTP/s service in one of two ways. First, the resource owner can give permission on his own behalf or second, by arranging an interaction between the HTTP/s service and the resource owner. HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS attempts to do away with the shortcomings of HTML when trying for declaration of dynamic views in web applications. It tries to separate the Business Logic from UI Manipulation (DOM Manipulation) by the usage of data binding. With the advent of such front-end frameworks, there is a need authentication libraries for such libraries. Here we build ngOAuth2, an AngularJS OAuth 2.0 based Authentication Module which we write in Typescript.

[*]For the title, try not to use more than 3 lines. Typeset the title in 10 pt bold and uppercase.

2. OAuth 2.0

2.1. *Definitions*

The RFC on OAuth 2.0 gives the following definitions[1]:

- Resource Owner: Entity that has the user information. Can be
  - Either, the user itself
  - Or, an application like facebook that has user data, and use grants the third party app the permission to take data from this application
- Client: The application making the request for accessing the protected resource
- Authorization Grant: A means to authorization. Can be
  - Authorization Code
  - Implicit
  - Resource Owner Password Credentials
  - Client Credentials
- Authorization Server: Entity that verifies the received Authorization Grant and issues Access Tokens, Referesh Tokens accordigly
- Access Token: Token used for obtaining the requested protected resource
- Resource Server: Entity that actually holds the protected resource

2.2. *Authorization Flow*

Now we discuss the general flow while authorizing Applications (Client) using OAuth 2.0 Authorization Protocol[1]. The Application makes an Authorization Request to the User, also known as the Resource Owner. The resource owner returns an Authorization Grant which the application shall send to the Authorization Server. This server verifies the Authorization grant and sends an Access Token and Refresh Token Back to the Application. The Application now sends this Access Token in its subsequent requests to the Resource Server that verifies this and returns the Protected Resource to the Application. These subsequent requests usually have an Authorization header set with the Bearer scheme of authentication that is used to pass our access tokens[2].

3. Problem

Here are some existing libraries

- *satellizer*: A customizable popular oAuth 2.0 based authentication library
  - Does not have refresh token support.
  - Due to it's support for only JWT, backward support is missing for non Javascript Web Tokens.
- *angular-oauth2*: A recent AngularJS based authentication library

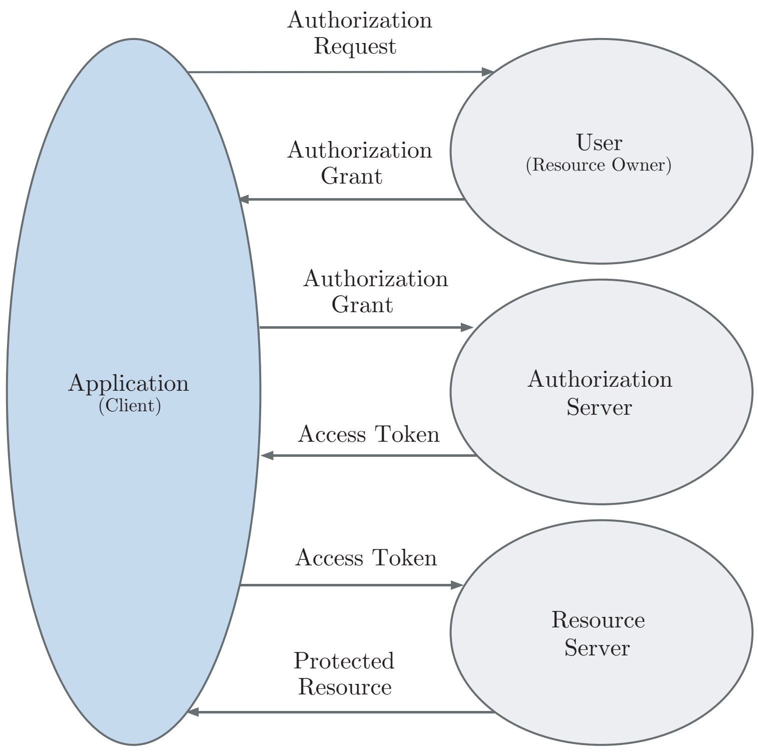

Fig. 1. OAuth 2.0 Authorization Flow

  - Does not support localStorage or sessionStorage
  - Does not have social authentication

It is clear that there is something or the other lacking in the current libraries and we attempt to tackle that.

4. Possible Solutions

A list of possible solutions for the above problem:

- Mix and match libraries and find which is compatible with which.
  - A tedious task, as these libraries aren't built to be compatible with each other
- Build a new library tackling the existing issues with the libraries or fork one of these to make changes

5. ngOAuth2

We built an AngularJS OAuth 2.0 based authentication module that we wrote in Typescript. Here is the link

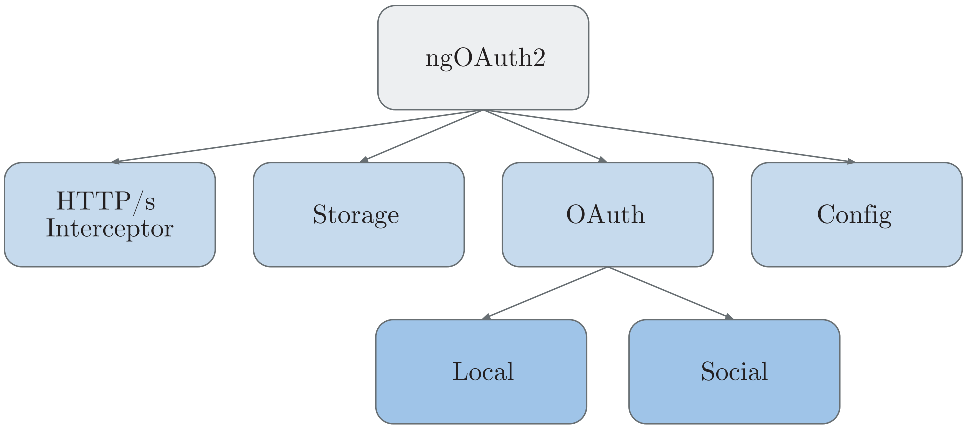

Fig. 2. Logical View for ngOAuth2 Library

5.1. *Logical View*

On the first level we have HTTP Interceptor that intercepts outgoing requests and adds Authorization Headers to them. Storage Module that is responsible storing of the authentication data as key-value pairs in HTML5's sessionStorage and localStorage. The OAuth Module that exposes the APIs of login/logout/refreshToken to the application using the library. The config module is responsible for the application's configuration decisions like whether to use sessionStorage or localStorage, the location of login URL, base URL etc. On the second level we have Local and Social which are responsible for password grant based and authorization code (returned from resource owners like Facebook and Google) based authorization.

5.2. *Process View*

Here we see the OAuth component is the central component that login, logout APIs go through. These in turn go through the HTTP interceptor component which finally returns the data fetching from the backend. The OAuth Component then sets access and refresh tokens in the Storage component. We also observe that for all outgoing HTTP/s requests, the requests go through HTTP Interceptor component which take the access token from Storage Component and sets Authorization Headers using that. The Config Component modifies the configurations variables of the library (state).

5.3. *Features*

- Event broadcasting in case of login(success/error), token error (invalid/missing), logout(error/success).
- Social login support for Facebook and Google.
- LocalStorage and SessionStorage storage options supported

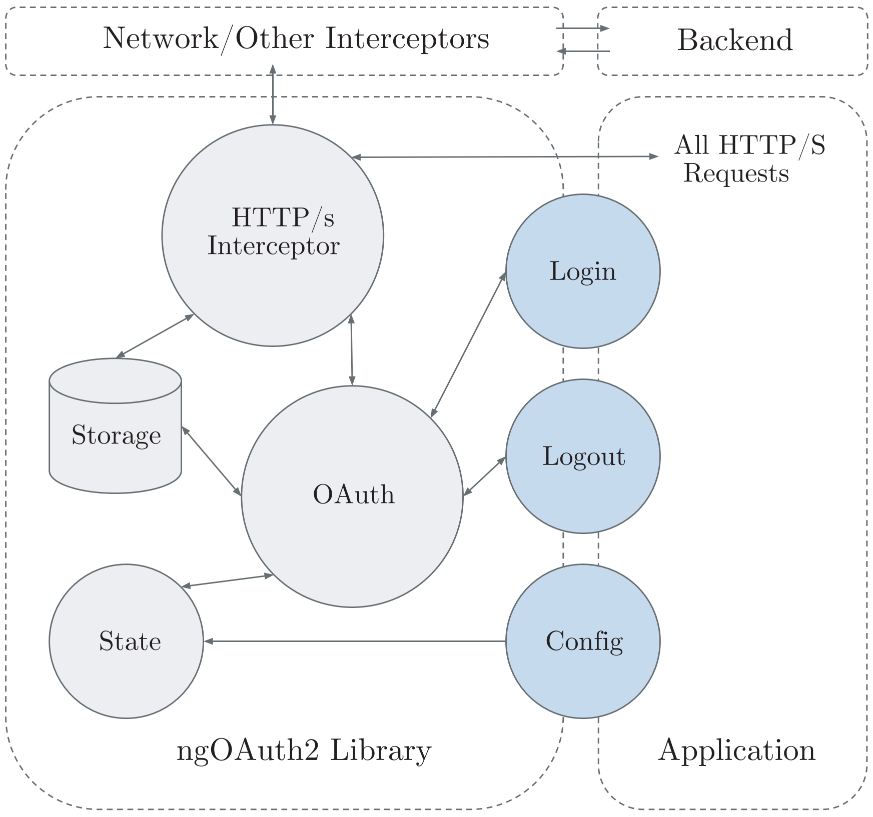

Fig. 3. Process View for ngOAuth2 Library

- Highly configurable parameters like event names, methods to login/logout requests, choice of storage, name of authorization header etc.

5.4. *Implementation*

As mentioned, we implemented the ngOAuth2 library in Typescript which gets translated to Javascript ES5 which then gets bundled up.

5.4.1. *Directory Structure*

- src/
  - ngOAuth2/
    - constants/
    - interfaces/
    - providers/
    - services/

  - main.ts

- dist/
- build/
- examples/
- gulpfile.js
- package.json
- tslint.json
- ...

The src directory has all the Typescript source code that shall be converted into Javascript ES5 code using gulp scripts written in gulpfile.js. This is ES5 code goes into the build/ directory where browserify's gulp script converts it into a bundled browser-ready Javascript in the dist/ directory.

5.4.2. *Design Choices*

- Typescript, Javascript(ES6 (2015), ES5(2012)), CoffeeScript
  - Typescript due to mainly the typed nature, easy for spotting bugs in an IDE.
- NPM scripts vs Gulp scripts
  - Gulp scripts to avoid the lengthy and complex npm scripts
- Regular Tokens, Javascript Web Tokens
  - Regular Tokens for now, JWT a future support
- Cookies, localStorage, sessionStorage
  - localStorage, sessionStorage for now, Cookies a future support

5.4.3. *Other Technologies Used*

- UglifyJS: To Uglify and Minify Javascript
- Browserify: To convert require('modules') into browser compatible bundled dependency
- Yeoman: To generate initial library template
- TSLint: For checking our Typescript code for functionality errors, readability and maintainability

6. Future Work

Future Work includes storage support for Cookies, support for JSON Web Tokens and platforms other than Facebook and Google. Since the work is done in Typescript, it will be easier to upgrade it for newer versions Angular.

7. References

References

[1] D. Hardt, “The oauth 2.0 authorization framework,” 2012.

[2] M. Jones and D. Hardt, “The oauth 2.0 authorization framework: Bearer token usage,” tech. rep., 2012.