Build your Angular 2 App: From Auth to calling an API

auth0.com/blog/2015/05/14/creating-your-first-real-world-angular-2-app-from-authentication-to-calling-an-api-and-everything-in-between/

TL;DR: Get the sample Angular 2 app from this Github repository. Also, check out this talk I did where I explain and live-code this same example.

This article was updated on May 18, 2016 to reflect Angular 2 rc.1

Last week, the Angular team announced that **Angular 2 was moving from Alpha to Developer Preview**. Therefore, we figured **it was time to give it a try**.

After looking around the internet, I learned that **all of the existing examples were only one single page** with just 1 or 2 components. Those examples, while nice, weren't really useful for creating a real world app. Therefore, in order to learn and help the community, we decided to **create a fully working, real life small application that would have multiple pages and would handle authentication as well as calling an API**. In order to do all this, we'd use most of the new Angular 2 features like the router, components, directives, pipes and DI, as well as Fetch for calling an API. In this article, we'll explain how we did it.

Before we start

Warning!

Before we start, I wanted to give you a warning. Angular 2 is changing constantly since it's still in a Developer Preview, which means we'll be working with the bleeding edge. Therefore, this example might become outdated. However, we'll work on updating the source code as often as we can to stay up to date.

Seed project

In order to start working with Angular 2, I strongly recommend checking Pawel's ng2-play. It makes it really easy to install and spin up a new project with Angular 2.

Read the comments!

Throughout this example, please **read the comments on the code**, which will give you insights about what each of the lines does.

Install angular2-jwt

We can use **angular2-jwt** to make authenticated HTTP requests easily.

```
npm install angular2-
jwt
```

Let's code!

Setting up the router

The first thing we should do is set up the router. For each URL, our job is to setup which component should be loaded and where.

First, we need to create our App component, which will set up the routes:

```
// src/app/app.ts
import {Component} from '@angular/core';
import {RouteConfig, RouterLink, Router} from '@angular/router-
deprecated';
import {LoggedInRouterOutlet} from './LoggedInOutlet';
import {Home} from '../home/home';
import {Login} from '../login/login';
import {Signup} from '../signup/signup';
@Component({
 // HTML selector for this component
 selector: 'auth-app'
 template: `
 <div class="container">
 <router-outlet></router-outlet>
 </div>
 `,
 directives: [LoggedInRouterOutlet]
})
@RouteConfig([
 { path: '/', redirectTo: ['/Home'] },
 { path: '/home', component: Home, as: 'Home' },
 { path: '/login', component: Login, as: 'Login' },
 { path: '/signup', component: Signup, as: 'Signup' }
])
export class App {
 constructor() {}
}
```

Now we can `bootstrap` the application to get it running.

```
// src/index.ts
import { bootstrap } from '@angular/platform-browser-
dynamic';
import { provide } from '@angular/core';
import { FORM_PROVIDERS } from '@angular/common';
import { ROUTER_PROVIDERS } from '@angular/router-
deprecated';
import { Http, HTTP_PROVIDERS } from '@angular/http';
import { AuthConfig, AuthHttp } from 'angular2-jwt';
import { App } from './app/app';
bootstrap(
 App,
 [
 FORM_PROVIDERS,
 ROUTER_PROVIDERS,
 HTTP_PROVIDERS,
 provide(AuthHttp, {
 useFactory: (http) => {
 return new AuthHttp(new AuthConfig({
 tokenName: 'jwt'
 }), http);
 },
 deps: [Http]
 })
 ]
);
```

To configure **angular2-jwt** we need to provide `AuthHttp` with `useFactory` pointing to a new instance.

Restricting access to pages

We don't want anonymous users to be able to access the `Home` route, so we should redirect them if they aren't authenticated. For that, we can create our own `RouterOutlet`, which will only let authenticated users access the home route.

For that, we need to do two things. First of all, we need to modify our `app.js` to use the new **LoggedInRouterOutlet** we'll create, rather than using the default one.

We need a `LoggedInRouterOutlet` directive which extends the `Router` to allow us to specify which routes are public and rediredcts if the route is private and the user isn't authenticated.

```
// src/app/LoggedInOutlet.ts
import {Directive, Attribute, ViewContainerRef, DynamicComponentLoader} from
'@angular/core';
import {Router, RouterOutlet, ComponentInstruction} from '@angular/router-
deprecated';
import {Login} from '../login/login';
@Directive({
 selector: 'router-outlet'
})
export class LoggedInRouterOutlet extends RouterOutlet {
 publicRoutes: any;
 private parentRouter: Router;
 constructor(_viewContainerRef: ViewContainerRef, _loader: DynamicComponentLoader,
 _parentRouter: Router, @Attribute('name') nameAttr: string) {
 super(_viewContainerRef, _loader, _parentRouter, nameAttr);
 this.parentRouter = _parentRouter;
 // The Boolean following each route below
 // denotes whether the route requires authentication to view
 this.publicRoutes = {
 'login': true,
 'signup': true
 };
 }
 activate(instruction: ComponentInstruction) {
 let url = instruction.urlPath;
 if (!this.publicRoutes[url] && !localStorage.getItem('jwt')) {
 this.parentRouter.navigateByUrl('/login');
 }
 return super.activate(instruction);
 }
}
```

Creating the login page

Now it's time to create our Login component. Its main function will be displaying the login form and calling the login API using `Http`. Once the server successfully authenticates the user, we'll save the JWT we get back in `localStorage` and then redirect the user to the home page.

```
// src/login/login.ts
import { Component } from '@angular/core';
import { Router, RouterLink } from '@angular/router-deprecated';
import { CORE_DIRECTIVES, FORM_DIRECTIVES } from '@angular/common';
import { Http, Headers } from '@angular/http';
import { contentHeaders } from '../common/headers';
let styles = require('./login.css');
let template = require('./login.html');
@Component({
 selector: 'login',
 directives: [RouterLink, CORE_DIRECTIVES, FORM_DIRECTIVES ],
 template: template,
 styles: [ styles ]
})
export class Login {
 constructor(public router: Router, public http: Http) {}
 login(event, username, password) {
 event.preventDefault();
 let body = JSON.stringify({ username, password });
 this.http.post('http://localhost:3001/sessions/create', body, { headers:
contentHeaders })
 .subscribe(
 response => {
 localStorage.setItem('jwt', response.json().id_token);
 this.router.parent.navigateByUrl('/home');
 },
 error => {
 alert(error.text());
 console.log(error.text());
 }
 );
 }
 signup(event) {
 event.preventDefault();
 this.router.parent.navigateByUrl('/signup');
 }
}
```

```
 <!-- src/login/login.html -->
 <div class="login jumbotron center-block">
 <h1>Login</h1>
 <form role="form" (submit)="login($event, username.value, password.value)">
 <div class="form-group">
 <label for="username">Username</label>
 <input type="text" #username class="form-control" id="username"
placeholder="Username">
 </div>
 <div class="form-group">
 <label for="password">Password</label>
 <input type="password" #password class="form-control" id="password"
placeholder="Password">
 </div>
 <button type="submit" class="btn btn-default">Submit</button>
 <a href="/signup">Click here to Signup</a>
 </form>
 </div>
```

Creating the Home component

The user is logged in. It's time to create the Home component, to which the user will arrive upon successful login. It will let the user call an authenticated API as well as display the JWT information.

```
// src/home/home.ts
import { Component } from '@angular/core';
import { CORE_DIRECTIVES } from '@angular/common';
import { Http, Headers } from '@angular/http';
import { Router } from '@angular/router-deprecated';
import { AuthHttp } from 'angular2-jwt';
let styles = require('./home.css');
let template = require('./home.html');
@Component({
 selector: 'home',
 directives: [CORE_DIRECTIVES],
 // Here we specify the template we'll use
 template: template,
 styles: [styles]
})
export class Home {
 // Here we define this component's instance variables
 // They're accessible from the template
 jwt: string;
 decodedJwt: string;
 response: string;
 api: string;
 constructor(public router: Router, public http: Http, public authHttp: AuthHttp) {
 // We get the JWT from localStorage
 this.jwt = localStorage.getItem('jwt');
 // We also store the decoded JSON from this JWT
 this.decodedJwt = this.jwt && window.jwt_decode(this.jwt);
 }
 logout() {
 // Method to be called when the user wants to logout
 // Logging out means just deleting the JWT from localStorage and redirecting the
```

```
 // Logging out means just deleting the JWT from localStorage and redirecting the
user to the Login page
 localStorage.removeItem('jwt');
 this.router.parent.navigateByUrl('/login');
 }
 callAnonymousApi() {
 this._callApi('Anonymous', 'http://localhost:3001/api/random-quote');
 }
 callSecuredApi() {
 // We call the secured API
 this._callApi('Secured', 'http://localhost:3001/api/protected/random-quote');
 }
 _callApi(type, url) {
 this.response = null;
 if (type === 'Anonymous') {
 // For non-protected routes, just use Http
 this.http.get(url)
 .subscribe(
 response => this.response = response.text(),
 error => this.response = error.text()
 );
 }
 if (type === 'Secured') {
 // For protected routes, use AuthHttp
 this.authHttp.get(url)
 .subscribe(
 response => this.response = response.text(),
 error => this.response = error.text()
 );
 }
 }
}


 <!-- src/home/home.html -->
 <div>
 <div class="home jumbotron centered">
 <h1>Welcome to the angular2 authentication sample!</h1>
 <h2 *ngIf="jwt">Your JWT is:</h2>
 <pre *ngIf="jwt" class="jwt"><code></code></pre>
 <pre *ngIf="jwt" class="jwt"><code></code></pre>
 <p>Click any of the buttons to call an API and get a response</p>
 <p><a class="btn btn-primary btn-lg" role="button" (click)="callAnonymousApi()">Call
Anonymous API</a></p>
 <p><a class="btn btn-primary btn-lg" role="button" (click)="callSecuredApi()">Call
Secure API</a></p>
 <p><a class="btn btn-primary btn-lg" role="button" (click)="logout()">Logout</a></p>
 <h2 *ngIf="response">The response of calling the <span class="red"></span> API is:
</h2>
 <h3 *ngIf="response"></h3>
 </div>
 </div>
```

Aside: Using Angular 2 with Auth0

Auth0 issues **JSON Web Tokens** on every login for your users. That means that you can have a solid identity

infrastructure, including single sign-on, user management, support for social (Facebook, Github, Twitter, etc.), enterprise (Active Directory, LDAP, SAML, etc.) and your own database of users with just a few lines of code.

We can add Auth0 to the app we just created really easily. There are just a few simple steps:

Step 0: Sign Up for Auth0

If you don't already have any Auth0 account, sign up for one now to follow along with the other steps.

Step 1: Add Auth0Lock to Your App

Lock is the beautiful (and totally customizable) login box widget that comes with Auth0. The script for it can be brought in from a CDN link or with npm.

> *Note: If you use npm to get Auth0Lock, you will need to include it in your build step.*

```
<!-- src/client/index.html -->
...
<!-- Auth0 Lock script -->
<script src="https://cdn.auth0.com/js/lock-9.1.min.js"></script>
<!-- Setting the right viewport -->
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
...
```

Step 2: Add an Authentication Service

It's best to set up an injectable service for authentication that can be used across the application.

With Auth0, we get access to the user's profile and JWT in the `lock.show` callback and these items can be saved in local storage for use later.

```
// src/client/shared/auth.service.ts
import {Injectable, NgZone} from 'angular2/core';
import {Router} from 'angular2/router';
import {AuthHttp, tokenNotExpired} from 'angular2-jwt';
// Avoid name not found warnings
declare var Auth0Lock: any;
@Injectable()
export class Auth {
 lock = new Auth0Lock('YOUR_AUTH0_CLIENT_ID', 'YOUR_AUTH0_DOMAIN');
 refreshSubscription: any;
 user: Object;
 zoneImpl: NgZone;
 constructor(private authHttp: AuthHttp, zone: NgZone, private router: Router)
{
 this.zoneImpl = zone;
 this.user = JSON.parse(localStorage.getItem('profile'));
 }
 public authenticated() {
 // Check if there's an unexpired JWT
 return tokenNotExpired();
 }
 public login() {
 // Show the Auth0 Lock widget
 this.lock.show({}, (err, profile, token) => {
 if (err) {
 alert(err);
 return;
 }
 // If authentication is successful, save the items
 // in local storage
 localStorage.setItem('profile', JSON.stringify(profile));
 localStorage.setItem('id_token', token);
 this.zoneImpl.run(() => this.user = profile);
 });
 }
 public logout() {
 localStorage.removeItem('profile');
 localStorage.removeItem('id_token');
 this.zoneImpl.run(() => this.user = null);
 this.router.navigate(['Home']);
 }
}
```

Step 3: Add a Click Handler to Login

We can use the methods from our authentication service in any of our components which means we can easily add a click handler to a "Login" and "Logout" button.

```
 <!-- src/client/app.component.html -->
 ...
 <button (click)="auth.login()" *ngIf="!auth.authenticated()">Log In</button>
 <button (click)="auth.logout()" *ngIf="auth.authenticated()">Log
Out</button>
 ...
```

Step 4: Make Authenticated HTTP Requests

We can again use `AuthHttp` from **anuglar2-jwt** to automatically have our JWTs sent in HTTP requests.

```
// src/home/home.ts
...
 _callApi(type, url) {
 this.response = null;
 if (type === 'Anonymous') {
 // For non-protected routes, just use Http
 this.http.get(url)
 .subscribe(
 response => this.response =
response.text(),
 error => this.response = error.text()
 );
 }
 if (type === 'Secured') {
 // For protected routes, use AuthHttp
 this.authHttp.get(url)
 .subscribe(
 response => this.response =
response.text(),
 error => this.response = error.text()
 );
 }
 }
...
```

Step 5: Done!

That's all there is to it to add authentication to your Angular 2 app with Auth0!

Conclusions

In this article, we've learned how to create a multiple page Angular 2 app that uses the router, templates, directives and components to implement both authentication and calling an API. You can see the complete example on Github, as well as a talk that I did, where this example is live-coded.

Before ending, I want to thank David East for his support with some questions, PatrickJS for his help on coding parts of the example and to Jesus Rodriguez for cleaning up some of the unused code.

Happy hacking :).