COM644 Full-Stack Web and App Development

Practical C5: Front-end Authentication

Aims

- To identify authentication as a key component in online applications
- To introduce Authentication as a Service and Auth0 as an authentication provider
- To install Auth0 and integrate it with an Angular application
- To create an Authentication Service
- To implement authenticated login and logout
- To demonstrate the preservation of application state
- To implement restricted access to selected content

Contents

C5.1 Authentication as a Service

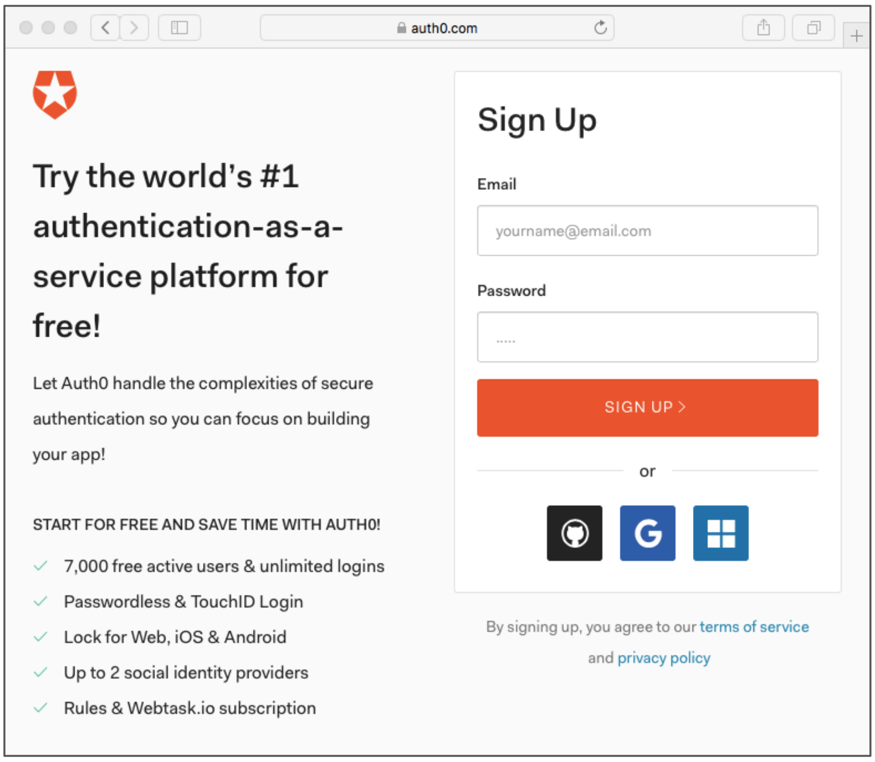
Authentication is emerging as one of the most critical aspects of web-based software development. Digital security has become front-page news and a lack of user confidence in the security of their data is one of the main factors that limits the uptake of new online systems.

Traditionally, software developers have handled authentication in-house, using self-written routines to handle login, authenticated transactions and protection from unauthorized users; but recently a number of 3rd party vendors have made available cloud-based systems which accept requests from user software and handle user authentication before passing control back to the host application.

One of the most popular providers of Authentication as a Service (AaaS) is **Auth0** (https://auth0.com), who provide a powerful free service which handles user authentication along with an extensive dashboard for user analysis.

C5.1.1 Signing up for Auth0

In order to use Auth0 in our application, we first need to sign up for a free account. Visit https://auth0.com and click on the "*Sign Up*" button in the top-right corner. You should then be presented with the screen shown in Figure C5.1 below.



auth0.com

Try the world's #1 authentication-as-a-service platform for free!

Let Auth0 handle the complexities of secure authentication so you can focus on building your app!

START FOR FREE AND SAVE TIME WITH AUTH0!

- ✓ 7,000 free active users & unlimited logins
- ✓ Passwordless & TouchID Login
- ✓ Lock for Web, iOS & Android
- ✓ Up to 2 social identity providers
- ✓ Rules & Webtask.io subscription

Sign Up

Email: yourname@email.com

Password: .....

SIGN UP >

or

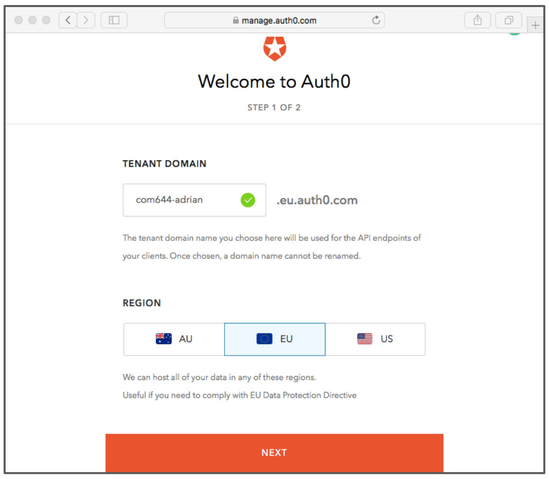
By signing up, you agree to our terms of service and privacy policy

Figure C5.1 Auth0 Sign-up

You can sign up either by providing a username and password or through one of your social media accounts.

C5.1.2 Configuring Auth0

Once signed up, you will be presented with two pages 2 pages of brief details. On the first (shown in Figure C5.2 below), choose your name and region and click '*Next*'. On the second screen (Figure C5.3), provide information about your intentions (I suggest selecting 'Personal', 'Developer' and 'Just playing around') and click the '*Create Account*' button.



manage.auth0.com

Welcome to Auth0

STEP 1 OF 2

TENANT DOMAIN

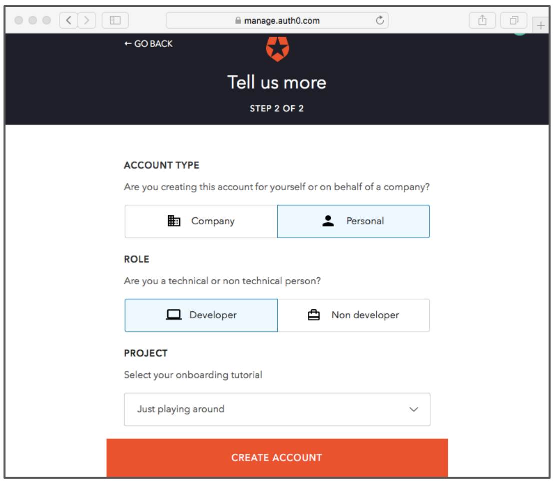
com644-adrian ✓ .eu.auth0.com

The tenant domain name you choose here will be used for the API endpoints of your clients. Once chosen, a domain name cannot be renamed.

REGION

AU [✓ EU] US

We can host all of your data in any of these regions.
Useful if you need to comply with EU Data Protection Directive



manage.auth0.com

← GO BACK

Tell us more

STEP 2 OF 2

ACCOUNT TYPE

Are you creating this account for yourself or on behalf of a company?

Company [✓ Personal]

ROLE

Are you a technical or non technical person?

[✓ Developer] Non developer

PROJECT

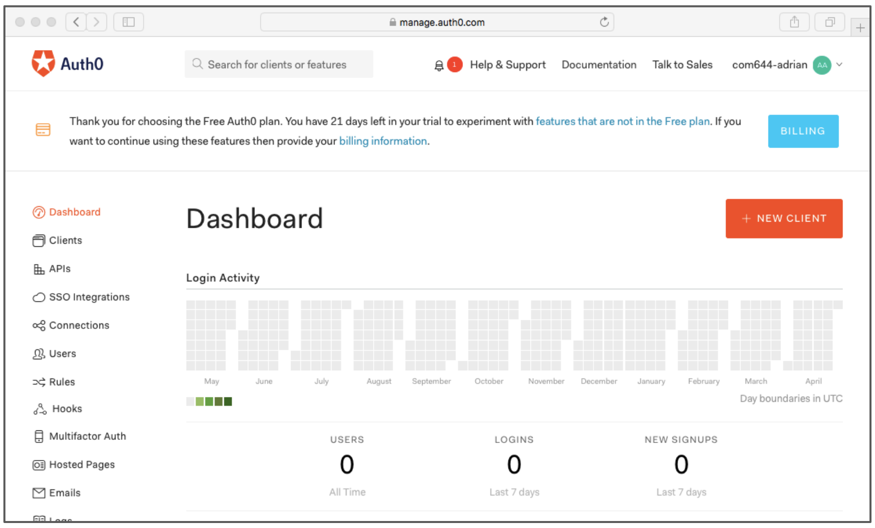
Select your onboarding tutorial

Just playing around

Figure C5.2 Choose Name and Region

Figure C5.3 Further Details

When the account is created, you will see your main Dashboard (Figure C5.4) which will provide analysis and report on users and logins for all of your applications.



manage.auth0.com

Thank you for choosing the Free Auth0 plan. You have 21 days left in your trial to experiment with features that are not in the Free plan. If you want to continue using these features then provide your billing information.

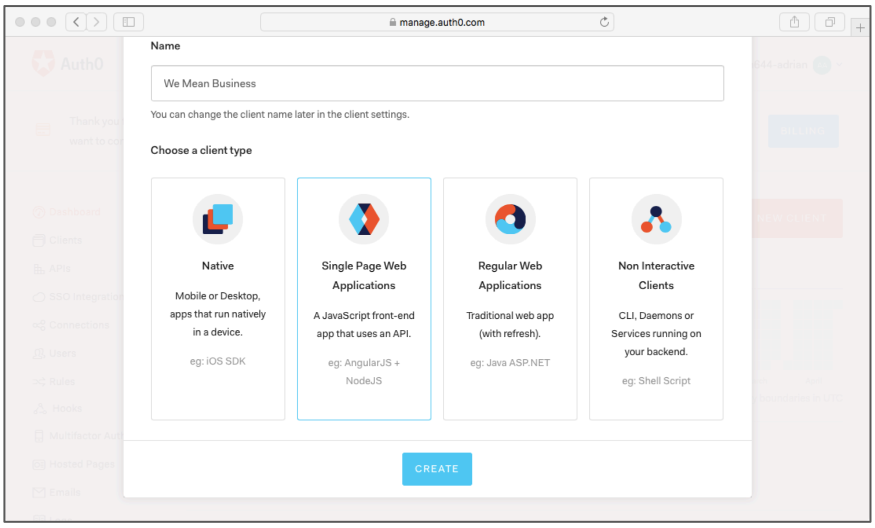
Dashboard

Login Activity

USERS	LOGINS	NEW SIGNUPS
0	0	0
All Time	Last 7 days	Last 7 days

Figure C5.4 New Client

Each application that you want to manage with Auth0 requires you to create a new Auth0 Client, so click on the "*New Client*" button. First, you will be asked to provide a name and select a client type as shown in Figure C5.5 below. Any suitable name is OK, but make sure you select "Single Page Web Application" as the client type.



manage.auth0.com

Name

We Mean Business

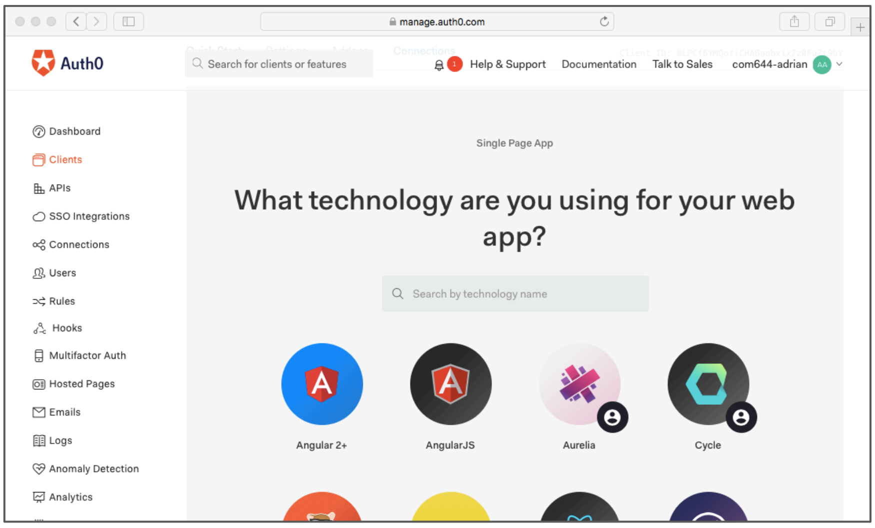
You can change the client name later in the client settings.

Choose a client type

Native
Mobile or Desktop, apps that run natively in a device.
eg: iOS SDK
Single Page Web Applications
A JavaScript front-end app that uses an API.
eg: AngularJS + NodeJS
Regular Web Applications
Traditional web app (with refresh).
eg: Java ASP.NET
Non Interactive Clients
CLI, Daemons or Services running on your backend.
eg: Shell Script
CREATE

Figure C5.5 Select Client Type

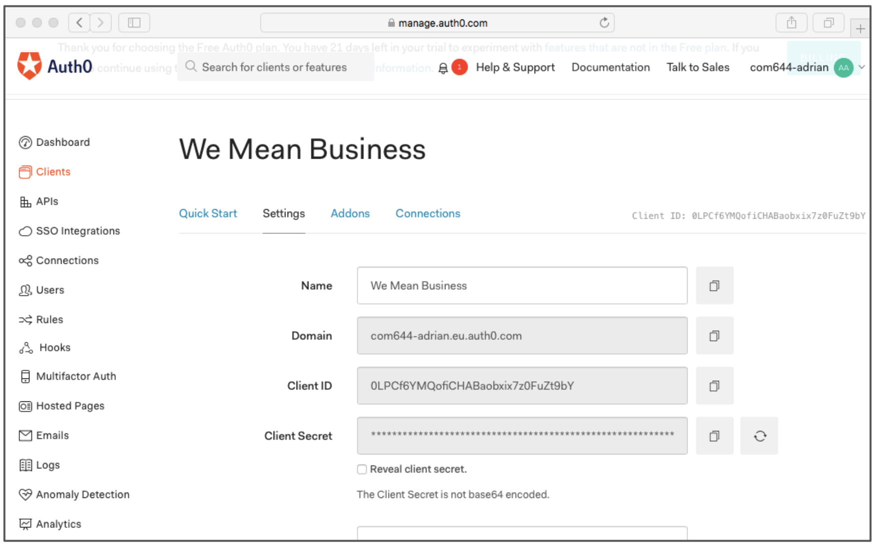
Next, in response to the question 'What technology are you using for your web app?', click on the icon for '*Angular 2+*' (Figure C5.6).



manage.auth0.com

C5.6 Choose Technology Stack

Finally, from the menu in the next page, click on the 'Settings' link to display the list of Client settings as shown in Figure C5.7.



We Mean Business

Client ID: 0LPCf6YMQofiCHABaobxix7z0FuZt9bY

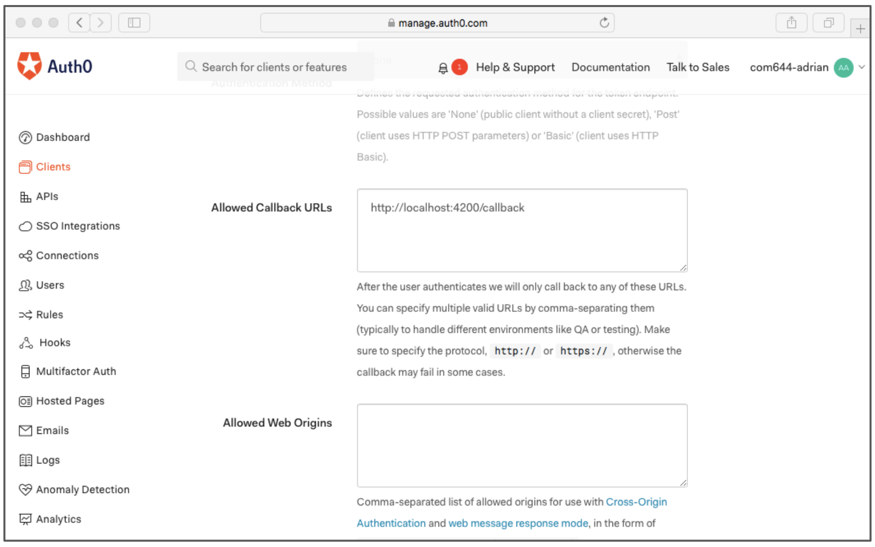
Name	We Mean Business
Domain	com644-adrian.eu.auth0.com
Client ID	0LPCf6YMQofiCHABaobxix7z0FuZt9bY
Client Secret	**

☐ Reveal client secret.

The Client Secret is not base64 encoded.

Figure C5.7 Settings

In the settings list, scroll down to '*Allowed callback URLs*' and add the URL http://localhost:4200/callback. This is the URL to which the browser will be directed once an authentication operation has been processed – we will add this route to our application later.



Possible values are 'None' (public client without a client secret), 'Post' (client uses HTTP POST parameters) or 'Basic' (client uses HTTP Basic).

Allowed Callback URLs: http://localhost:4200/callback

After the user authenticates we will only call back to any of these URLs. You can specify multiple valid URLs by comma-separating them (typically to handle different environments like QA or testing). Make sure to specify the protocol, `http://` or `https://`, otherwise the callback may fail in some cases.

Allowed Web Origins:

Comma-separated list of allowed origins for use with Cross-Origin Authentication and web message response mode, in the form of

Figure C5.8 Specify Callback URL

C5.2 Adding Authentication to the Application

C5.2.1 Install Auth0 Package

Now that the Auth0 Client is prepared, we can add authentication to our application.

First, we use npm to install the auth0-js package to our Angular application by the command

```
U:/C5> npm install --save auth0-js
```

This adds a JavaScript library to our **node_modules** folder and we need to include it in our application by specifying its path in the scripts entry within the file *.angular-cli.json*.

File*: C5/.angular-cli.json*

```
...
"scripts": ["../node_modules/auth0-js/build/auth0.js"];
...
```

C5.2.2 Create an Authentication Service

The next stage is to provide the new Angular service that will handle invoke Auth0 to handle authentication within the application.

The code for the Authentication Service can be found by clicking the “Quickstart” link in the menu shown in Figure C5.7 above and selecting Angular 2+ from the next menu presented. Scrolling down to “Create an Authentication Service” will reveal the code with your own ClientID, Domain and Audience already filled in.

(Note: If you cannot find the menu, click on “Clients” in the menu on the left-hand side of the browser, then click on your Client name.)

File*: C5/src/app/auth.service.ts*

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { filter } from 'rxjs/operators';
import * as auth0 from 'auth0-js';

@Injectable()
export class AuthService {

  auth0 = new auth0.WebAuth({
    clientID: '   --- YOUR CLIENT ID ---    ',
    domain: '   --- YOUR DOMAIN ---    ',
    responseType: 'token id_token',
    audience: '  --- YOUR AUDIENCE ---    ',
    redirectUri: 'http://localhost:4200/callback',
    scope: 'openid'
  });

  constructor(public router: Router) {}

  public login(): void {
    this.auth0.authorize();
  }

}
```

Note: You will need to change the **`redirectUri`** value in the code above to http://localhost:**4200**/callback. The code in the Quickstart example defaults to port 3000.

C5.2.3 Test the Auth0 Integration

To test the installation of the Auth0 package and its integration into our application, we can add a "*Login*" button to our home page that invokes the **`AuthService login()`** function.

First, we need to register the **`AuthService`** with the application by adding it to *app.module.ts*. As usual, we **`import`** the service and add it to the list of **`providers`** in the **`@ngModule`** definition.

File*: C5/src/app/app.module.ts*

```
...

import { AuthService } from './auth.service';

...


providers: [WebService, AuthService],

...

}
```

Now, we create the Callback Component to support the callback URL that we provided when specifying the Auth0 Client settings.

Note how we choose to define a `template` rather than a `templateURL`. This is commonly done if the template is very brief. Take care that the template code is specified between *backtick* characters ` `. This character is available on a Mac keyboard (usually beside the left *Shift* key), but if not available on a PC keyboard you can obtain it by holding down the ALT key and then tapping '9' and '6' on the numeric keypad.

File*: C5/src/app/callback.component.ts*

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

@Component({
  selector: 'app-callback',
  template: `<p>Loading...</p>`,
  styleUrls: []
})

export class CallbackComponent {

    constructor(private router: Router) {}

    ngOnInit() {
        this.router.navigate(['']);
    }
}
```

Now that the **`CallbackComponent`** is created, we can add the */callback* route to the list supported and register the Component with the application.

File*: C5/src/app/app.module.ts*

```
import { CallbackComponent } from './callback.component';

...

var routes = [

  ...

  {
    path: 'callback',
    component: CallbackComponent
  }
]

...

@NgModule({
  declarations: [
    AppComponent, Businesses, HomeComponent,
    BusinessComponent, CallbackComponent
  ],

...
```

Next, we import the **`AuthService`** into the **`HomeComponent`** and inject it into the constructor so that we can refer to it in the template.

File*: C5/src/app/home.component.ts*

```
import { Component } from '@angular/core';
import { AuthService } from './auth.service';
@Component({
  selector: 'home',
  templateUrl: './home.component.html',
  styleUrls: []
})

export class HomeComponent {
  constructor(private authService: AuthService) {}
}
```

Now, we can add a button to the **`HomeComponent`** template and bind the **`click`** action to the **`login()`** function within the **`AuthService`**.

File: ***C5/src/app/home.component.html***

```
<div class="jumbotron">
    <h1>We MEAN Business</h1>
    <button (click)="authService.login()">Login</button>
</div>
```

We can now test the Auth0 integration by loading the application and navigating to http://localhost:4200/ to confirm that the button has been added to the template (Figure C5.9). Clicking the button should result in the Auth0 login page being displayed (Figure C5.10). Note that the login functionality is not yet complete and so the login action will not fully work.

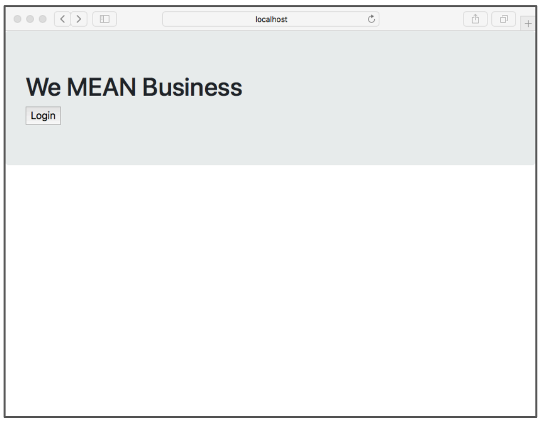


Figure C5.9 Login option

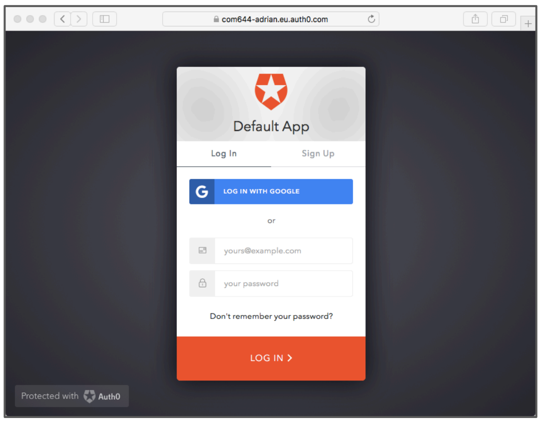


Figure C5.10 Auth0 Login

C5.2.4 Complete the Authentication Service

Now that we can see that everything is properly integrated, we can complete the **`AuthService`** by taking the code from the Auth0 Quickstart example. (Scroll down to the "*Finish the service*" section.

Note that we have changed the two lines highlighted in the code box below – changing a navigation path from ***/home*** to ***/***.

File*: C5/src/app/auth.service.ts*

```
...

@Injectable()

export class AuthService {

...

  public handleAuthentication(): void {
    this.auth0.parseHash((err, authResult) => {
      if (authResult && authResult.accessToken &&
                        authResult.idToken) {
          window.location.hash = '';
          this.setSession(authResult);
          this.router.navigate(['/']);
      } else if (err) {
          this.router.navigate(['/']);
          console.log(err);
      }
    });
  }

  private setSession(authResult): void {
    const expiresAt = JSON.stringify(
       (authResult.expiresIn * 1000) + new Date().getTime());
    localStorage.setItem('access_token',
                          authResult.accessToken);
    localStorage.setItem('id_token', authResult.idToken);
    localStorage.setItem('expires_at', expiresAt);
  }

  public logout(): void {
    localStorage.removeItem('access_token');
    localStorage.removeItem('id_token');
    localStorage.removeItem('expires_at');
    this.router.navigate(['/']);
  }

  public isAuthenticated(): boolean {
    const expiresAt =
          JSON.parse(localStorage.getItem('expires_at'));
    return new Date().getTime() < expiresAt;
  }

}
```

To complete the authentication setup, we now include a call to the **`AuthService handleAuthentication()`** method in the application's root component (*app.component.ts*). The method processes the authentication hash while the application loads.

File*: C5/src/app/app.component.ts*

```
import { Component } from '@angular/core';
import { AuthService } from './auth.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {

  constructor (private authService: AuthService) {
    authService.handleAuthentication();
  }
}
```

C5.3 Authentication-dependent content

C5.3.1 Add a logout button

One of the main uses of user authentication is to restrict access to certain content or actions to users who have been successfully authenticated. We demonstrate this by providing a second button to call the **`logout()`** function and then using the **`isAuthenticated()`** function to control which button is displayed.

By combining the **`isAuthenticated()`** function with the Angular **`ngIf`** directive, we specify that the *Login* button should only be displayed when there is no current authenticated user, while the *Logout* button is displayed only when are user has successfully logged in.

File*: C5/src/app/home.component.html*

```
<div class="jumbotron">
    <h1>We MEAN Business</h1>
    <button *ngIf="!authService.isAuthenticated()"
            (click)="authService.login()">Login</button>
    <button *ngIf="authService.isAuthenticated()"
            (click)="authService.logout()">Logout</button>
</div>
```

Running the application and checking its operation in the browser should confirm that the Login and Logout buttons are displayed as intended.

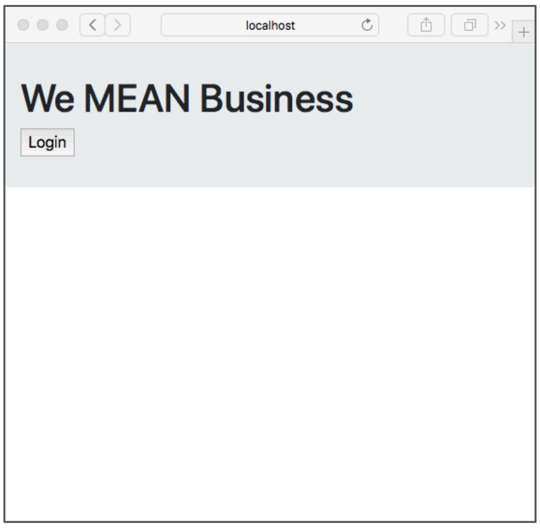


Figure C5.11 Before login

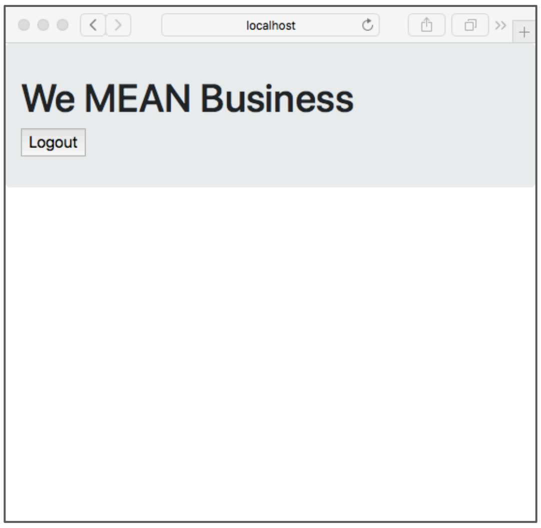


Figure C5.12 After login

We can see how the authentication operates by opening the Browser console and selecting the "*Storage*" (or "*Application*" in Chrome) tab and selecting the *LocalStorage* element. You should be able to see how the **`login()`** function results in an access token, ID token and expiry value being set, while the logout() function deletes these values. Examining the **`login()`** and **`logout()`** source in the **`AuthService`** should also demonstrate how the manipulation of values in *LocalStorage* is used to manage authentication.

In our application, we want to be able to login and logout from every page, so add the buttons to the templates for the **`BusinessComponent`** and **`BusinessesComponent`**.

**Files*: C5/src/app/businesses.component.html*
and *C5/src/app/business.component.html***

```
<div class="jumbotron">
    <h1>We MEAN Business</h1>
    <button *ngIf="!authService.isAuthenticated()"
            (click)="authService.login()">Login</button>
    <button *ngIf="authService.isAuthenticated()"
            (click)="authService.logout()">Logout</button>
</div>

...
```

To refer to the **`AuthService`** in the template, we also import it into the **`BusinessesComponent`** and **`BusinessComponent`** TypeScript fles.

File*: C5/src/app/businesses.component.ts*

```
...

import { AuthService } from './auth.service';

...

    constructor(private webService: WebService,
                private authService: AuthService) {}

...
```

File*: C5/src/app/business.component.ts*

```
...

import { AuthService } from './auth.service';

...

    constructor(private webService: WebService,
                private route: ActivatedRoute,
                private formBuilder: FormBuilder,
                private authService: AuthService) {

...
```

Testing this in the browser shows that our login and logout options are indeed available on each page, but we discover that no matter which page the user selects the login option on, the user is always left on the application home page – due to the navigation in the **`CallbackComponent`**.

C5.3.2 Preserving application state

The solution to the login navigation problem is to store the current location in the browser's *SessionStorage* before login and then have the **`CallbackComponent`** re-direct to that location after login.

Note: Values held in browser *LocalStorage* are permanent and will persist across browser sessions until explicitly removed. Values in *SessionStorage* are only valid for the current browsing session and are automatically destroyed when the browser tab is closed.

File*: C5/src/app/auth.service.ts*

```
...

  public login(): void {
    sessionStorage.url = window.location.href;
    this.auth0.authorize();
  }

...
```

File*: C5/src/app/callback.component.ts*

```
...

   ngOnInit() {
     window.location.href = sessionStorage.url;
   }
...
```

Finally, we remove the default navigation that re-directs the user to the application home page after logout. Note that in applications where entire pages are restricted to authenticated users, it may be necessary to re-direct the user after logout, depending on the page on which the logout operation is selected.

File*: C5/src/app/auth.service.ts*

```
...

  public logout(): void {
    localStorage.removeItem('access_token');
    localStorage.removeItem('id_token');
    localStorage.removeItem('expires_at');
    // this.router.navigate(['/']);
  }

...
```

Testing the application in the browser should demonstrate that users are now able to log in and log out on any page, while maintaining their position within the application.

C5.3.3 Restricting access to selected content

Finally, we want to restrict the ability to leave a review only to users who have been successfully authenticated. We achieve this in the same way as we have already seen with the Login and Logout buttons, using the **`isAuthenticated()`** function and the **`ngIf`** directive to only display the review form if an authenticated user is present. Where there is no logged in user, we provide a message instructing users that they should log in to leave a review.

File***: C5/src/app/business.component.html***

```
...

  <span *ngIf = "authService.isAuthenticated()">
    <h2>Please review this business</h2>
    <form [formGroup]="reviewForm" (ngSubmit)="onSubmit()">

    ...

    </form>
  </span>

  <span *ngIf = "!authService.isAuthenticated()">
    <h5 class="text-primary">
       Please log in to leave a review for this business
    </h5>
  </span>

</div>
```