COM661 Full Stack Strategies and Development

FE16. 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 implement restricted access to selected content
- To add authentication functionality into the navigation bar

Table of Contents

16.1 Auth0 Authentication

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 systems.

When developing our back-end API, we implemented authentication ourselves using JSON Web Tokens that were passed in the header of requests to the API. Another option is to use a 3rd party provider, and we will investigate this in providing an authentication layer to our front-and application.

16.1.1 Authentication as a Service

Traditionally, software developers have handled authentication in-house, using self-written routines to handle login, authenticate transactions and protect 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.

16.1.2 Signing up for Auth0

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 15.1 below. You can sign up by providing a username and password or through a social media account.

Do it now!	Sign up for Auth0 and open a free account. You can provide a dedicated Auth0 account or signup using one of your existing social media accounts.

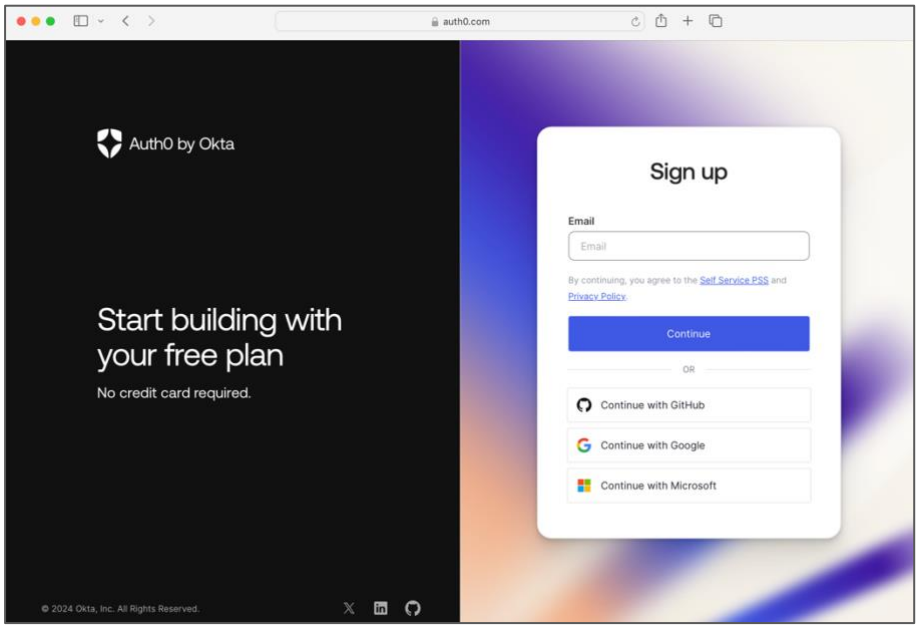

Figure 16.1 Auth0 Sign up

16.1.3 Configuring Auth0

When your Auth0 account is created and your profile has been established, you will be presented with the Auth0 Dashboard as shown in Figure 16.2, below.

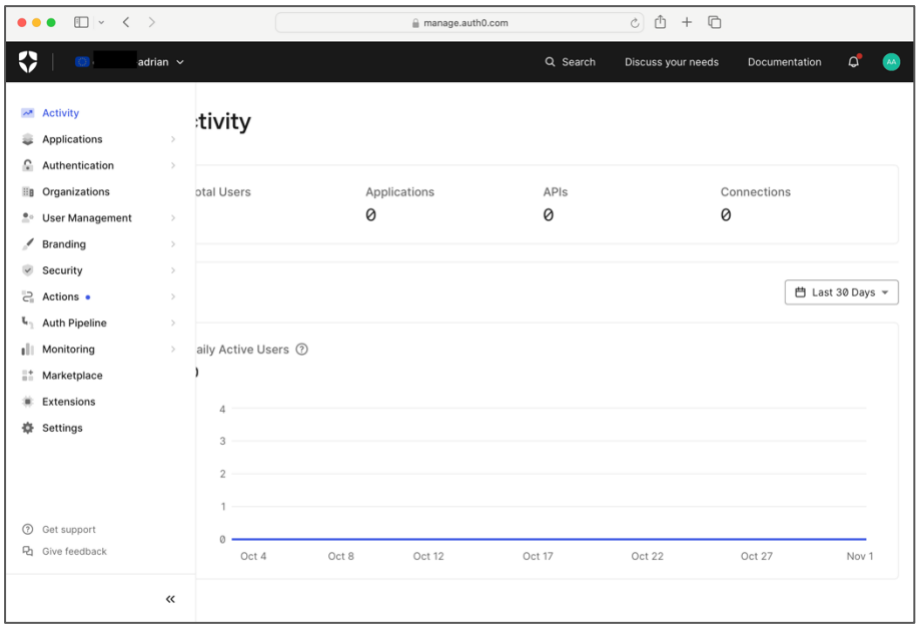

Figure 16.2 The Auth0 Dashboard

The dashboard is the control centre from where you can manage your applications and users. For our purposes the first activity is to create a new application by clicking on the “Applications” link on the menu on the left, then “Applications” from the sub-menu, and finally the “Create Application” button in the top-right of the browser. In response, you will

be presented with a screen where you provide the name for your application and a choice of application types, from which you should choose “Single Page Web Applications”, as illustrated in Figure 16.3, below.

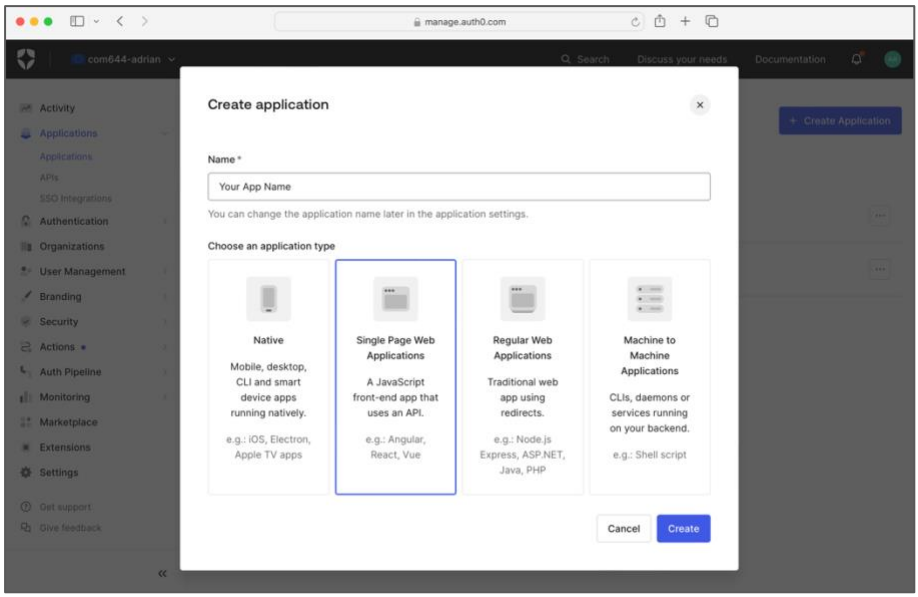

Figure 16.3 Create a New Application

Next, in response to the question ‘What technology are you using for your web app?’, click on the icon for ‘Angular’, as shown in Figure 16.4, below.

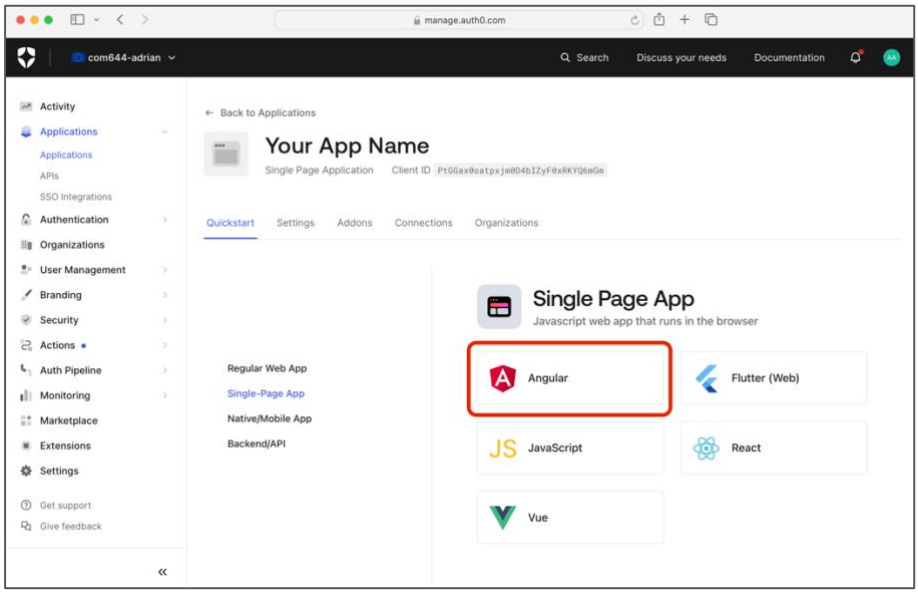

Figure 16.4 Choose Platform

Then, from the application configuration page, we need to click the “Settings” tab from the menu and set the “Allowed Callback URLs”, “Allowed Logout URLs” and “Allowed Web Origins” each to http://localhost:4200, as shown in Figure 16.5, below.

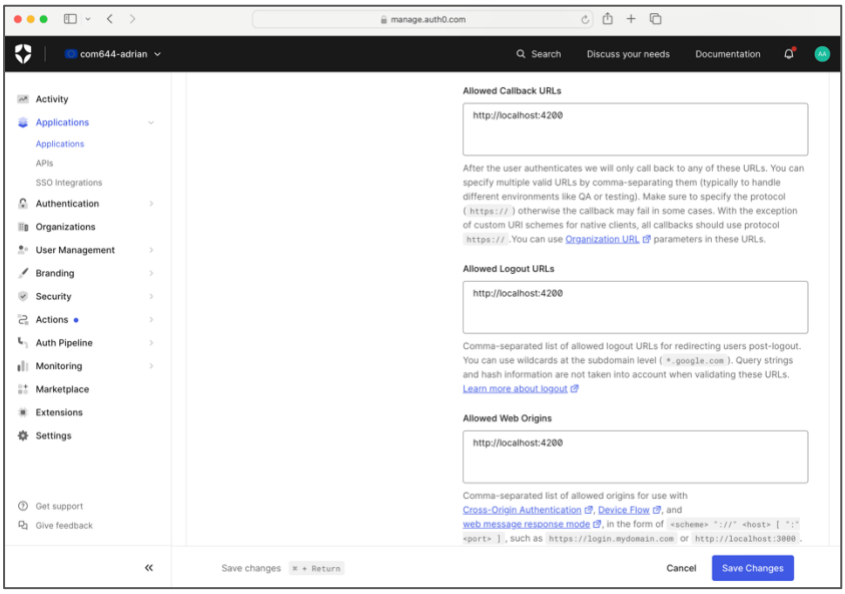

Figure 16.5 Application Settings

The Callback URL and Logout URLs are the locations to which the user will be redirected after a login and logout event, respectively; while the Allowed Web Origin permits Auth0 to silently refresh the authentication tokens when the user visits or refreshes a page, so preventing repeated login.

Do it now!	Create a new Auth0 application as directed above and provide the settings for the application configuration. Remember to click “Save Changes” at the bottom of the configuration page

16.1.4 Install the Auth0 package

The final preparation step is to add the Auth0 package to our Angular application. We do this by using use **npm** to install the **auth0-angular** package as shown in the following command.

```
C:\bizFE> npm install @auth0/auth0-angular --save
```

Do it now!	Install the auth0-angular package to the bizFE application as shown above.

16.2 Integrating Auth0 with the Application

When we install the **auth0-angular** package, we make available a Software Development Kit (SDK) that exposes several elements that enable us to integrate Auth0 with our application.

In this section, we perform the necessary setup and initialization, and then add login and logout buttons to our application home page.

16.2.1 Auth0 Setup

The Auth0 website provides an excellent Quickstart guide that you can access from the home page of your Auth0 application. When you select the QuickStart link and choose Angular from the list of available platforms that are supported, you are greeted with the step-by-step guide shown in Figure16.6, below.

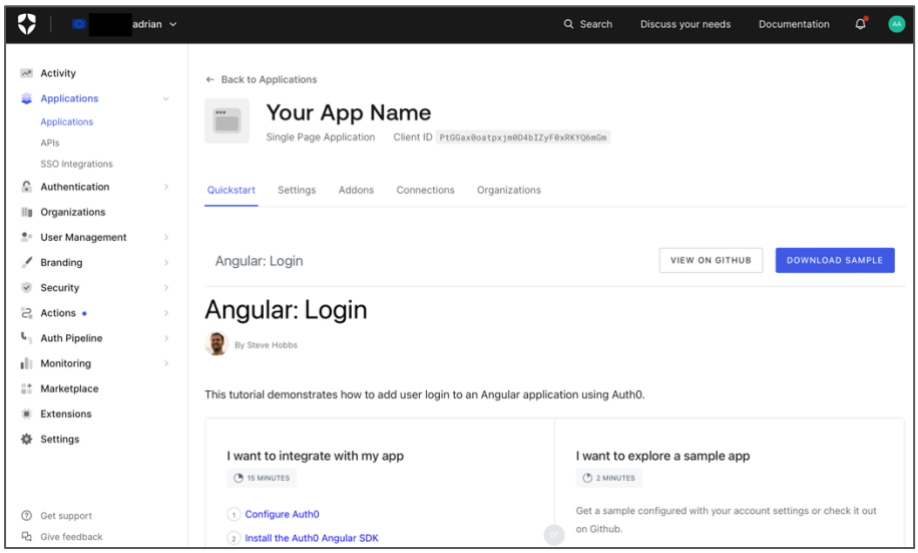

Figure 16.6 The Auth0 Quickstart Guide

Note:	In the sections that follow, you can copy and paste the code from the Quickstart guide into your application, but please note that the code presented in the code boxes in this section differs from the Quickstart code in some areas to reflect functionality already implemented in our application.

The first step is to replace the code in the file ***main.ts*** with that presented in the code box below. Note that your personal **`domain`** and **`clientId`** values will already be populated for you in the code seen in the Quickstart guide. Note also that this code differs from that in the Quickstart guide since we have had to integrate the Quickstart code with the existing functionality of our ***main.ts***.

File: bizFE/src/main.ts

```
import { bootstrapApplication } from
                                    '@angular/platform-browser';
import { AppComponent } from './app/app.component';
import { appConfig } from './app/pri';
import { provideAuth0 } from '@auth0/auth0-angular';
import { provideHttpClient } from '@angular/common/http';

bootstrapApplication(AppComponent, {
  providers: [
    provideAuth0({
      domain: 'YOUR-DOMAIN-NAME,
      clientId: 'YOUR-CLIENT-ID',
      authorizationParams: {
        redirect_uri: window.location.origin
      }
    }),
    provideHttpClient(),
    appConfig.providers
  ]
}).catch((err) => console.error(err));
```

Do it now!	Update ***main.ts*** with the code shown in the code box above. You can also run the application at this stage to verify that no errors have been introduced by Auth0 integration.

16.2.2 Create the Auth Button Component

Next, we add the **`AuthButton`** Component that will be added to the application to enable login and login, though initially, we will just consider login. As usual, the component is represented by a TypeScript class definition and a HTML template (we choose not to provide an associated CSS file).

The TypeScript file imports the standard Component package and the Auth0 Authentication Service and makes the service available by injecting it to the Component by the usual method of providing it as a parameter to the component constructor.

File: bizFE/src/app/authbutton.component.ts

```
import { Component } from '@angular/core';
import { AuthService } from '@auth0/auth0-angular';

@Component({
  selector: 'auth-button',
  templateUrl: 'authbutton.component.html',
  standalone: true
})

export class AuthButtonComponent {

  constructor(public auth: AuthService) {}
}
```

In the HTML template, we define the button as a standard Bootstrap button and bind the button click event to the **`AuthService loginWithRedirect()`** method.

File: bizFE/src/app/authbutton.component.html

```
<button class="btn btn-primary"
        (click)="auth.loginWithRedirect()">
    Log in
</button>
```

Now we can add the **`AuthButton`** Component to the home page of our application by importing it into the TypeScript file and adding it to the list of imports.

File: bizFE/src/app/home.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { AuthButtonComponent } from "./authbutton.component";

@Component({
  selector: 'home',
  standalone: true,
  imports: [RouterOutlet, AuthButtonComponent],
  templateUrl: './home.component.html',
  styleUrl: './home.component.css'
})
export class HomeComponent {}
```

Finally, we can display the button on the home page by adding its selector tag below the existing heading.

File: bizFE/src/app/home.component.html

```
<h1 style="margin-top: 70px">
    Biz Directory
</h1>

<auth-button></auth-button>
```

Do it now!	Add the **`AuthButton`** to the home page by making the modification shown in the four code boxes above. Verify that the button has been added by visiting the home page of the application and observing the display in the browser as shown by Figure 16.7, below.

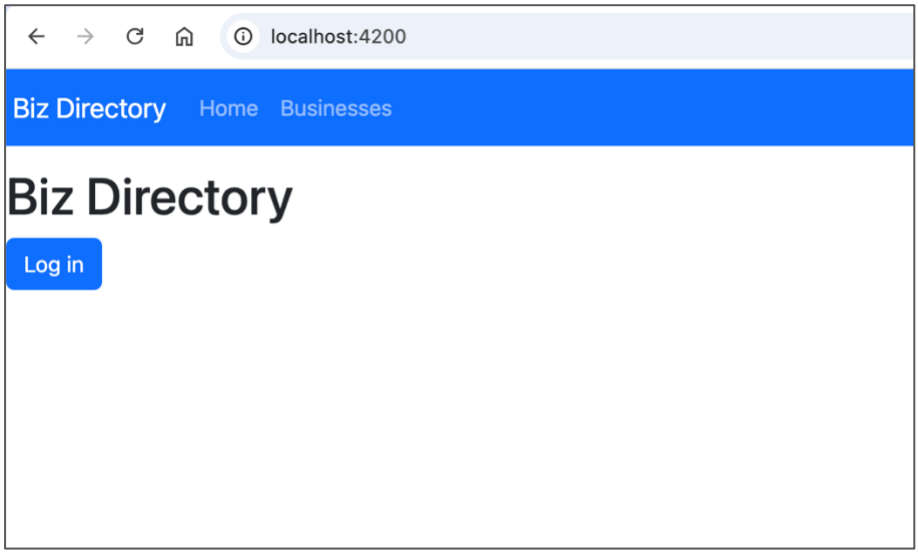

Figure 16.7 Login Button added to Home Page

Do it now!	At this stage, the button is already functional so click it and verify that you are redirected to the Auth0 Log in and Sign up screen. If you sign up for the application, using either a username and password or a social media account, you should find that you are redirected to the home page of the application.

Note:	It is important to distinguish between signing up for Auth0 as you did earlier and signing up for the application. The first is opening a developer account with Auth0 to enable you to deploy their authentication services in your applications. The second is signing up as a user of the application being protected by the authentication service.

16.2.3 Adding Logout Functionality

Now that users can sign in to the application, we to provide a way for them to sign out. This is implemented by using the **`logout()`** method provided by the **`AuthService`** service. Rather than specify separate components for login and logout, we will extend the functionality of our existing **`AuthButton`** so that is can cope with both uses.

The **`AuthService logout()`** method requires access to the Document Object Model to redirect the browser back to the specified origin. This is provided by the Angular **`DOCUMENT`** service that we need to import into the **`AuthButton`** TypeScript file and inject into the Component. This is achieved by the additional code highlighted in the code box below. The remaining new imports are to enable new functionality in the HTML template that we will see shortly.

File: bizFE/src/app/authbutton.component.ts

```
import { Component, Inject } from '@angular/core';
import { AuthService } from '@auth0/auth0-angular';
import { DOCUMENT } from '@angular/common';
import { AsyncPipe } from '@angular/common';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'auth-button',
  templateUrl: 'authbutton.component.html',
  standalone: true,
  imports: [AsyncPipe, CommonModule]
})

export class AuthButtonComponent {

  constructor(@Inject(DOCUMENT) public document: Document,
              public auth: AuthService) {}
}
```

The enhanced version of the **AuthButton** HTML template demonstrates how we can have a single element with two separate functions. The **ng-container** directive provides us with a way of delimiting a bunch of page content as a single entity – much as the **div** and **span** tags in HTML. However, when used in conjunction with ***ngIf** and **ng-template**, it provides an easy way of specifying multiple content blocks, only one of which will be rendered according to some condition. In effect, the **ng-template** element acts as a reserve, ready to replace the **ng-container** content when required. The code here specifies the logout button as the default content to be displayed during "normal" operation, i.e. when a user is logged in to the application. However, the **ng-container** includes an ***ngIf** directive

```
*ngIf = "authService.isAuthenticated$ | async;
else loggedOut"
```

which is read as "if the **isAuthenticated$** property of the **authService** is true, then display this content, otherwise replace this content with that in the **loggedOut** template". The final thing to note is the trailing **$** on the **isAuthenticated$** property name. This is an Angular convention (though not a requirement) to denote that the property is an Observable. As such, we cannot its value directly but need to subscribe to it. The **async** pipe provides a convenient way to subscribe to an Observable in the HTML template. The alternative would be to use the **subscribe()** method in the TypeScript file (as we have already seen) and update a component property with the value.

The full updated code for ***authbutton.component.html*** is provided in the code box below.

File: bizFE/src/app/authbutton.component.html

```
<ng-container *ngIf="auth.isAuthenticated$ | async;
               else loggedOut">
     <button class="btn btn-primary"
             (click)="auth.logout({
                         logoutParams: {
                            returnTo: document.location.origin }
                      })">
          Log out
     </button>
 </ng-container>

 <ng-template #loggedOut>
     <button class="btn btn-primary"
             (click)="auth.loginWithRedirect()">Log in</button>
```

```
    </ng-template>
```

Do it now!	Update the AuthButton component TypeScript and HTML files according to the code presented above. Then run the application and navigate to the home page. Click on the login button and see how you are transferred to the Auth0 sign in page. Provide the sign in details that you created earlier and note now you are returned to the application home page, but the button is now a logout button. Finally, click on logout and verify that you are returned to the home page with a login button. The sequence of browser content should be as that shown in Figure 16.8, below.

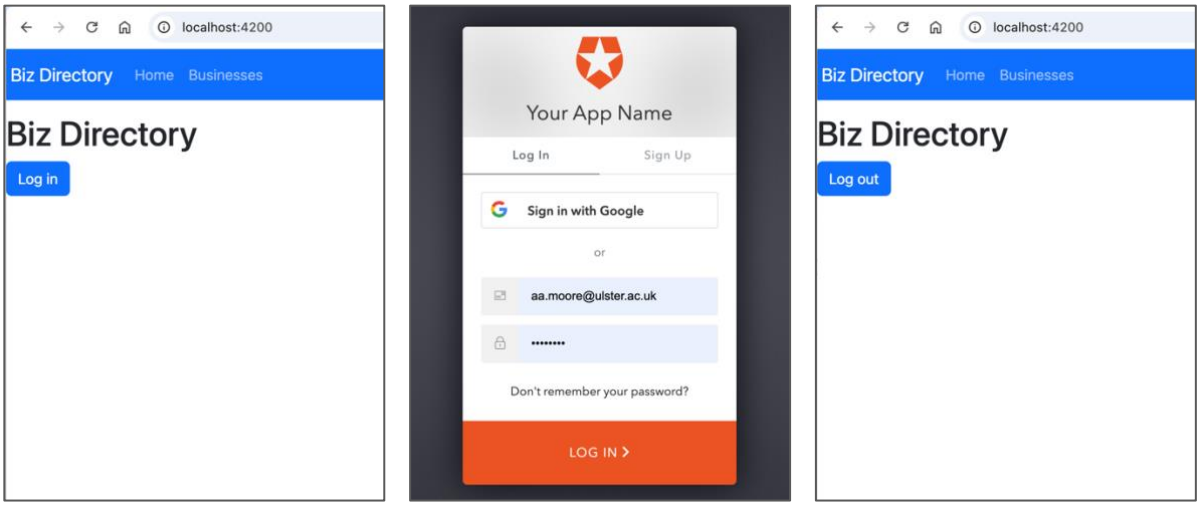

Figure 16.8 Ready to login, Auth0 Sign in, and Ready to Log out

16.2.4 User Profile

Now that users can register and login to our application, we can retrieve selected properties from the Auth0 User Profile object to personalize the application for the logged in user. The Auth0 **`AuthService`** provides a **`user$`** Observable that contains a range of fields. The full set of fields can be seen at the link in Section 16.4, but here, we will display the registered name and email address of the logged in user.

First, we define a new **`AuthUser`** component which will be represented by the selector auth-user and makes the **`AuthService`** available by injecting it into the component as usual. The following code box presents the source for ***authuser.component.ts***.

File: bizFE/src/app/authuser.component.ts

```
import { Component } from '@angular/core';
import { AuthService } from '@auth0/auth0-angular';
import { AsyncPipe } from '@angular/common';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'user-profile',
  templateUrl: 'authuser.component.html',
  standalone: true,
  imports: [AsyncPipe, CommonModule]
})

export class AuthUserComponent {
  constructor(public auth: AuthService) {}
}
```

Now we can define the representation of the **`AuthUser`** component by specifying the HTML template. Here, we check that the **`AuthService user$`** Observable has a value (i.e. that someone is currently logged in), and if so, we display their name and email attributes.

File: bizFE/src/app/authuser.component.html

```
<h3 style="margin-top:30px">
    Current User Details
</h3>

<ul *ngIf="auth.user$ | async as user">
   <li>{{ user.name }}</li>
   <li>{{ user.email }}</li>
</ul>
```

Now that the **`AuthUser`** component is available, we can import it into the **`HomeComponent`** to enable the user details to be added to the home page.

File: bizFE/src/app/home.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { AuthButtonComponent } from "./authbutton.component";
import { AuthUserComponent } from "./authuser.component";

@Component({
  selector: 'home',
  standalone: true,
  imports: [RouterOutlet, AuthButtonComponent,
                                              AuthUserComponent]
  templateUrl: './home.component.html',
  styleUrl: './home.component.css'
})
export class HomeComponent {}
```

Finally, we are able to add the **`AuthUser`** selector to the HTML template using the **`<user-profile>`** tag so that the details of a logged in user will be displayed when available.

File: bizFE/src/app/home.component.html

```
<h1 style="margin-top: 70px">
    Biz Directory
</h1>

<auth-button></auth-button>
<user-profile></user-profile>
```

Do it now!	Create the **`AuthUser`** component and add it to the home page by making the modifications described above. Run the application and log in to confirm that the user details are displayed as illustrated in Figure 16.9, below.

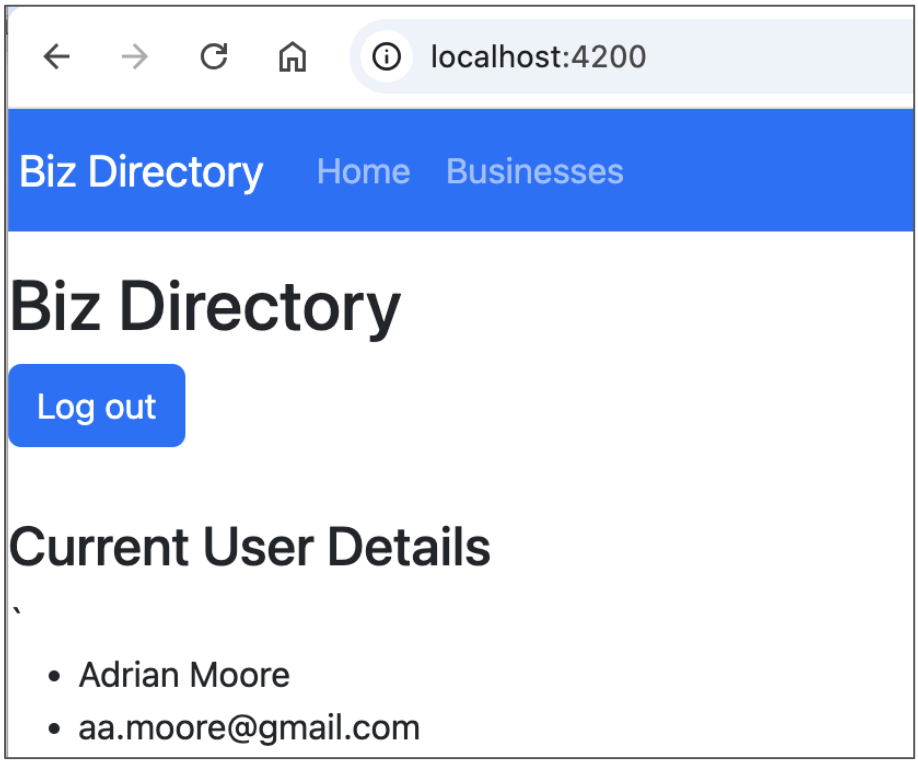

Figure 16.9 Displaying the User Profile

16.3 Add Authentication Functionality to the NavBar

Our authentication functionality is now in place, but the home paged is not the best place to implement it. We will want our users to be able to login from any part of our application and to provide login and logout on each page will require significant repetition of code.

Fortunately, we have a **`NavComponent`** that is constantly present regardless of our users' location within the application. If we move the authentication functionality to there, we can implement once and have it available from any page.

16.3.1 Move Authentication Functionality

Moving the functionality is quite straightforward and only requires that the **`AuthButton`** and **`AuthUser`** components are imported into the **`NavComponent`** and removed from the **`HomeComponent`**.

First, we import the required components into the **`NavComponent`**.

File: bizFE/src/app/nav.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet, RouterModule } from '@angular/router';
import { AuthButtonComponent } from "./authbutton.component";
import { AuthUserComponent } from "./authuser.component";

@Component({
  selector: 'navigation',
  standalone: true,
  imports: [RouterOutlet, RouterModule, AuthButtonComponent,
            AuthUserComponent],
  templateUrl: './nav.component.html'
})
export class NavComponent {}
```

Then, we remove the same content from the **`HomeComponent`**.

File: bizFE/src/app/home.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
~~import { AuthButtonComponent } from "./authbutton.component";~~
~~import { AuthUserComponent } from "./authuser.component";~~

@Component({
  selector: 'home',
  standalone: true,
  imports: [RouterOutlet~~, AuthButtonComponent,~~
            ~~AuthUserComponent~~],
  templateUrl: './home.component.html',
  styleUrl: './home.component.css'
})
export class HomeComponent {}
```

Do it now!	Move the imports related to authentication from the **`HomeComponent`** to the **`NavComponent`** as described above.

Note:	Note that removing the imports from the **`HomeComponent`** before removing the corresponding HTML template content will cause an error in the application if it is currently running, but this will be resolved in the next section.

16.3.2 Adding an Inline Button

Now that the components have been updated, we can make the corresponding changes to the HTML templates. First, we will add the **`AuthButton`** component to the navigation bar by specifying it in an inline **`<div>`** element. This is positioned below the **`<ul>`** element that contains the navbar links, but inside the **`<div>`** element that contains the navbar.

Note the class **`ms-auto`**, which specifies that this form should be right-aligned (i.e. allow the margin-start [**`ms`**] to automatically take as much space as is available).

File: bizFE/src/app/nav.component.html

```
...

    <div class="form-inline ms-auto"
         style="margin-right: 10px">
         <auth-button></auth-button>
    </div>
</div>
```

We also take the opportunity to change the appearance of the button by describing it using the Bootstrap **`btn-outline-light`** class. This requires the following change in the **`AuthButton`** HTML template.

File: bizFE/src/app/authbutton.component.html

```
...

<button class="btn btn-outline-light"

...

<button class="btn btn-outline-light"

...
```

Finally, we can remove the authentication content from the **`HomeComponent`** HTML template.

File: bizFE/src/app/home.component.html

```
<h1 style="margin-top: 70px">
    Biz Directory
</h1>

<auth-button></auth-button>

<user-profile></user-profile>
```

Do it now!	Make the changes described above to move the template authentication functionality from the **`HomeComponent`** to the **`NavComponent`**. Run the application and confirm that you see the login/logout button in the navigation bar as illustrated in Figure 16.10, below.

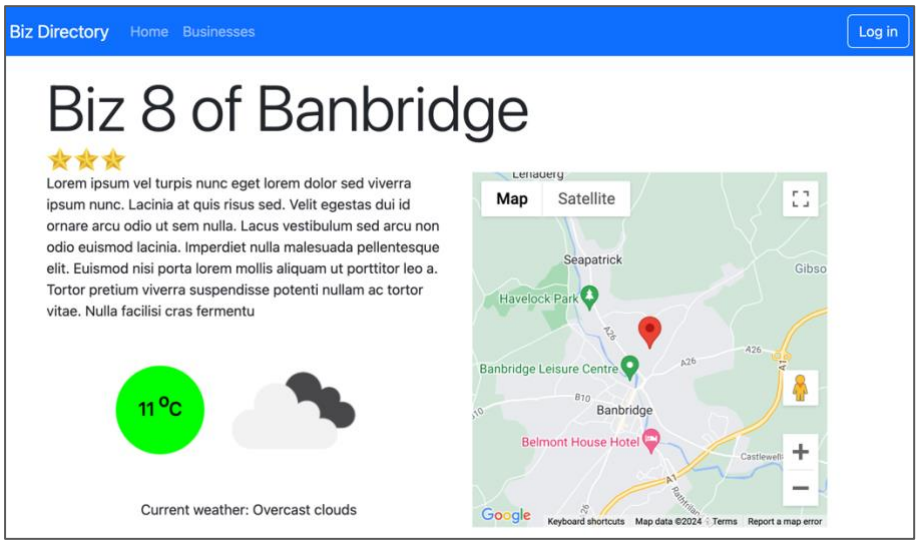

Figure 16.10 Authentication in the NavBar

16.3.2 Redirecting After Login

The login and logout mechanism implemented so far works well, but the user is redirected to the (currently empty) home page after every login/logout event. This is perhaps reasonable for logout, as the use may be on a page that is only available for logged in users, but there is no reason for a user to be redirected following a login event. Indeed, in our case it is desirable that the user remains on the current page, since they may have logged in order to leave a review for the specific business being viewed at that time (we will restrict the provision of reviews to logged in users in the next section).

Auth0 supports redirection after login by allowing us to specify the route to which the user is to be redirected as a parameter to the **`AuthService`** **`loginWithRedirect()`** method. The modification that we need to make to our **`AuthButton`** Component is illustrated below.

File: bizFE/src/app/authbutton.component.html

```
...

<ng-template #loggedOut>
    <button class="btn btn-outline-light"
            (click)="auth.loginWithRedirect( {
                appState: { target: this.router.url }
            })">Log in</button>
</ng-template>
```

To access the **`url`** attribute of the **`router`** service, we also need to import it into the **`AuthButton`** component TypeScript file, add it to the (currently not present) list of providers, and inject it into the Component constructor.

File: bizFE/src/app/authbutton.component.ts

```
...

import { Router } from '@angular/router';

@Component({
  selector: 'auth-button',
  templateUrl: 'authbutton.component.html',
  standalone: true,
  imports: [AsyncPipe, CommonModule],
  providers: [Router]
})

export class AuthButtonComponent {

    constructor( ...
        public auth: AuthService,
        public router: Router) {}

}
```

Do it now!	Make the changes to the **`AuthButton`** component described above to redirect a logged in user back to the page they were previously viewing. Run the application (logging out if you are currently logged in) and navigate to a single business. Now log in and confirm that you are returned to the page on which you initiated the login event.

16.3.3 Restricting Access to Selected Content

Ideally, 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 **`AuthButton`** component, using the **`AuthService isAuthenticated$`** Observable 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 that instructs users that they should log in to leave a review.

This time, the **`ng-container`** contains the form which the user completes to submit a review, while the **`ng-template`** that replaces it in the event of no user being logged in is a message requesting that the user logs in to review the business.

File: bizFE/src/app/business.component.html

```
...

   <ng-container
       *ngIf = "authService.isAuthenticated$ | async;
        else loggedOut">

        <h2>Please review this business</h2>
        <form [formGroup]="reviewForm"
              (ngSubmit)=onSubmit()">

        ...

   </ng-container>

   <ng-template #loggedOut>
        <h2>Please login to review this business</h2>
   </ng-template>

...
```

As usual, to make the **`AuthService`** functionality available in the **`BusinessComponent`**, we need to import it into the Component and inject it into the constructor.

File: bizFE/src/app/business.component.ts

```
...

import { AuthService } from '@auth0/auth0-angular';

...

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

...
```

The effect of this is that only users that are logged in will see the review form, while others will see a message as shown in Figures 16.11 and 16.12, below.

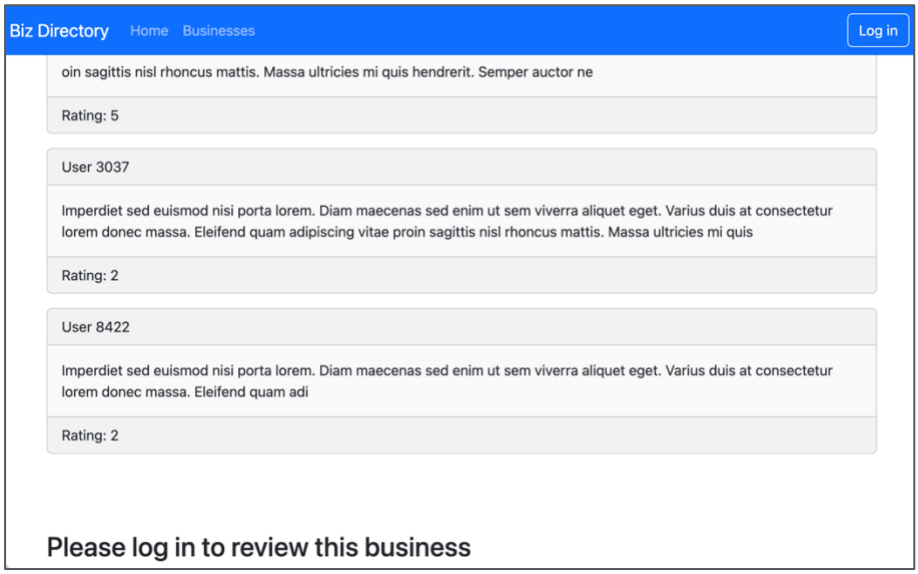

Figure 16.11 Logged out User Cannot Review

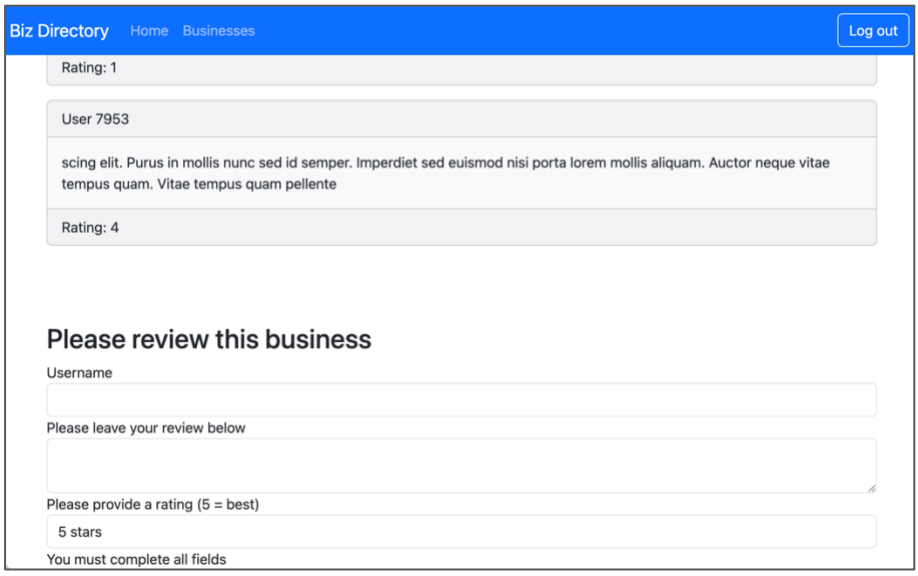

Figure 16.12 Logged in User Invited to Review

Do it now!	Modify the **`BusinessComponent`** so that the review form is enclosed in an **`ng-component`** directive as shown above, with an **`ng_template`** directive as an alternative "Please login" message for non-logged in users. Use the **`AuthService isAuthenticated$`** Observable to select between the content blocks by use of an **`*ngIf`** directive. Ensure that you are able to generate output such as that shown in Figures 16.11 and 16.12

16.4 Further Information

- https://coreui.io/docs/components/forms/
  Bootstrap Forms

- https://www.w3schools.com/Bootstrap/bootstrap_forms.asp
  Bootstrap Forms – W3Schools

- https://angular.dev/guide/forms/reactive-forms
  Angular ReactiveForms – the online manual

- https://malcoded.com/posts/angular-fundamentals-reactive-forms/
  Reactive Forms with Angular – using easy examples!

- https://angular.dev/api/forms/FormBuilder
  Angular FormBuilder class

- https://coryrylan.com/blog/angular-form-builder-and-validation-management
  Angular Form Builder and Validation Management

- https://malcoded.com/posts/angular-reactive-form-validation/
  Validating Reactive forms in Angular

- https://angular.dev/guide/forms/form-validation
  Angular Form Validation

- https://angular.dev/guide/templates/two-way-binding
  Two-way binding in Angular