**Angular & Okta**

**Step 1: Create free developer account at okta.com**

* Visit <http://developer.okta.com>
* Sign up for free account
* Check your email to verify your account

**Step 2: Add OpenID connect client app in Okta**

* In your Okta developer account
* Create a new application
* Select option for single-page-application

**Step 3: Set up app configuration for OpenId connect**

* To connect with okta, need to provide configs
* Need the clientId and Issuer… available on Okta application details

Example

Ng generate interface common/myAppconfig

Export default {  
Oidc: {  
Clientid: ‘2321asda321654asd112123a.1das3215**’,** Issuer: ’http://dev-462324532.okta.com/oauth2/default’,  
redirectUri: ‘http://localhost:4200/login/callback’**,** scopes: [‘openid’, ‘profile’, ‘email’ ]

} }

1. **Client ID:** public identifier of the client app
2. **Issuer:** URL when authorizing with okta authorization server
3. **Redirect URI:** once user logs in successfully send them here
4. **Scopes:** OIDC scopes provides access to information about a user such as name phone, email

**Step 4: Install Okta SDK dependencies**

* We will use two Okta SDK dependencies “ Okta Sign-In widget & Okta Angular SDK ”
* Okta sign-In widget is a JS library for application login
* We don’t have to create the html… just integrate the widget into our application
* Customizable…. Use your own logo, field names and custom fields
* Okta Angular SDK provides integration with angular router for authentication and authorization
* Core features

1. Login / Logout from Okta using OAUTH 2.0 API
2. Retrieve user information and determine authentication status

* Additional Features

1. Add protected routes that require authentication
2. Subscribe to change in authentication state

**Terminal**

* Npm install @okta/okta-signin-widget
* Npm install @okta/okta-angular “okta angular sdk”

**Step 5: Integrate Okta Sign-In widget**

* “Styles”: [ “node\_modules/@okta-signin-widget/dist/css/okta-sign-in.min.css”]
* Ng generate component component/login
* Edit login.component.html

<!—Container to inject the Okta Sign-in Widget -- >

<div class=” “>

<div id=”okta-sign-in-widget” class = “ “> </div>

</div>

* Edit login.component.ts

Graphical user interface, text, application

Description automatically generated

**Authorization Code Flow with PKCE**

* **Pkce on line 15**: Proof key for code exchange
* Recommended approach for controlling access between app and auth server
* Protects against authorization code interception attacks
* Introduces concepts of dynamics secrets
* Implemented with a code verifier, code challenge method

**NgOnINIT method**

* Renders elements with given id

**Step 6: Develop login status component for login/logout buttons**

* ng generate component components/loginStatus
* code for login-status.component.ts

**Graphical user interface, text, application

Description automatically generated**

* edit login-component.html

<div class="login-status-header">

    <div \*ngIf="isAuthenticated && userFullName" class="login-status-user-info">

        {{ userFullName }}!

    </div>

<button \*ngIf="!isAuthenticated" routerLink="/login" class="security btn">Login</button>

    <button \*ngIf="isAuthenticated" (click)="logout()" class="security-btn">Logout</button>

 <button \*ngIf="isAuthenticated" routerLink="/members" class="security-btn ml-1">Member</button>

    <button \*ngIf="isAuthenticated" routerLink="/order-history" class="security-btn ml-1">Orders</button>

</div>

**Step 7: Update App module configs to connect routes**

import {

  OktaAuthGuard

} from '@okta/okta-angular';

import {

  OKTA\_CONFIG,

  OktaAuthModule,

  OktaCallbackComponent

} from '@okta/okta-angular';

const oktaConfig = Object.assign({

  onAuthRequired: (oktaAuth,injector) => {

    const router = injector.get(Router);

    // Redirect the user to your custom login page

    router.navigate(['/login']);

  }

}, myAppConfig.oidc);

const routes: Routes = [

  {path: 'login/callback', component: OktaCallbackComponent},

  {path: 'login', component: LoginComponent}]

* Okta call back component

1. Once the user is authenticated, they are redirected to your app
2. Normally you would need to parse the response and store the OAuth+OIDC tokens
3. The Okta call Back Component does this for us.

  imports: [

    RouterModule.forRoot(routes),

    BrowserModule,

    HttpClientModule,

    NgbModule,

    ReactiveFormsModule,

**OktaAuthModule**

  ],

  providers: [ProductService, {provide: **OKTA\_CONFIG**, useValue: **oktaConfig**}]