**Enable authentication React Application by using Azure Active Directory B2C**

## **Step 1: Install the dependencies.**

To install the **MSAL Browser** and **MSAL React** libraries in your application, run the following command in your command line (Inside the ClientApp folder)

npm i @azure/msal-browser @azure/msal-react

## **Step 2: Add the authentication components.**

We need to add/modify the following components.

* src/authConfig.js
* src/index.js
* src/App.js
* src/components/NavMenu.js

**authConfig.js**

A configuration file that contains information about your Azure AD B2C identity provider and the web API service. **The React app uses this information to establish a trust relationship with Azure AD B2C**, sign in and sign out the user, acquire tokens, and validate the tokens. Sample code snippet attached below.

**In the b2cPolicies we have the user flow names**

export const b2cPolicies = {

    names: {

        signUpSignIn: 'B2C\_1\_Signin\_Signup\_MarsCommerce',

        forgotPassword: 'B2C\_1\_Profile\_Reset\_MarsCommerce',

        editProfile: 'B2C\_1\_Profile\_Edit\_MarsCommerce',

    },

    authorities: {

        signUpSignIn: {

            authority: 'https://marscad.b2clogin.com/marscad.onmicrosoft.com/B2C\_1\_Signin\_Signup\_MarsCommerce',

        },

        forgotPassword: {

            authority: 'https://marscad.b2clogin.com/marscad.onmicrosoft.com/B2C\_1\_Profile\_Reset\_MarsCommerce',

        },

        editProfile: {

            authority: 'https://marscad.b2clogin.com/marscad.onmicrosoft.com/B2C\_1\_Profile\_Edit\_MarsCommerce',

        },

    },

    authorityDomain: 'marscad.onmicrosoft.com',

};

**In the b2cPolicies we have the user flow names**

export const msalConfig = {

    auth: {

        clientId: '779516fb-0c12-41c4-898a-92378b402ce8', //ONLY mandatory field that you need to supply.

        authority: b2cPolicies.authorities.signUpSignIn.authority, //Choose SUSI as your default authority.

        knownAuthorities: [b2cPolicies.authorityDomain, 'marscad.b2clogin.com'], //B2C tenant's domain as trusted.

        redirectUri: 'https://localhost:44462', // You must register this URI on Azure Portal/App Registration.

        postLogoutRedirectUri: '', //Indicates the page to navigate after logout.

        navigateToLoginRequestUrl: true, //It will navigate back to the original request location before processing

    }

}

**Adding Scopes for protected API**

Add here the endpoints and scopes when obtaining an access token for protected web APIs.

Scopes you add here will be prompted for user consent during sign-in. By default, MSAL.js will add OIDC scopes (openid, profile, email) to any login request.

export const protectedResources = {

    apiscope: {

        endpoint: 'http://localhost:44462/apitoken',

        scopes: {

            read: ['https://marscad.onmicrosoft.com/apitoken/demo.read'],

            write: ['https://marscad.onmicrosoft.com/apitoken/demo.write'],

        },

    },

};

export const loginRequest = {

    scopes: [...protectedResources.apiscope.scopes.read, ...protectedResources.apiscope.scopes.write],

};

**index.js**

The JavaScript entry point to your application. This JavaScript file:

Mounts the App as the root component into the public/index.html page's <div id="root"> element.

Initiates the MSAL PublicClientApplication library with the configuration defined in the authConfig.js file.

The MSAL React should be instantiated outside of the component tree to prevent it from being

re-instantiated on re-renders.

The following headers need to import.

import { PublicClientApplication, EventType } from "@azure/msal-browser";

import { msalConfig } from './authConfig';

After instantiation of the MSAL library, **the JavaScript code passes the msalInstance as props to your application components**. For example, <App instance={msalInstance} />.

const msalInstance = new PublicClientApplication(msalConfig);

// Default to using the first account if no account is active on page load

if (!msalInstance.getActiveAccount() && msalInstance.getAllAccounts().length > 0) {

    // Account selection logic is app dependent. Adjust as needed for different use cases.

    msalInstance.setActiveAccount(msalInstance.getAllAccounts()[0]);

}

msalInstance.addEventCallback((event) => {

    if (

        (event.eventType === EventType.LOGIN\_SUCCESS ||

            event.eventType === EventType.ACQUIRE\_TOKEN\_SUCCESS ||

            event.eventType === EventType.SSO\_SILENT\_SUCCESS) &&

        event.payload.account

    ) {

        msalInstance.setActiveAccount(event.payload.account);

    }

});

root.render(

    <React.StrictMode>

        <BrowserRouter>

**<App instance={msalInstance} />**

        </BrowserRouter>

    </React.StrictMode>

);

**App.js**

The App component is the top level component of your app. It wraps everything between MsalProvider component. All components underneath MsalProvider will have access to the PublicClientApplication instance via context and all hooks and components provided by MSAL React. The App component mounts the PageLayout and its Pages child element.

The Pages component registers and unregister the MSAL event callbacks. The events are used to handle MSAL errors. It also defines the routing logic of the app.

If the App component file name is App.js, change it to App.jsx.

The following headers need to import.

import { MsalProvider, useMsal } from '@azure/msal-react';

import { EventType } from '@azure/msal-browser';

import { compareIssuingPolicy } from './utils/claimUtils';

import { b2cPolicies } from './authConfig';

import { useIsAuthenticated } from "@azure/msal-react";

const Pages = () => {

    const baseURL = process.env.REACT\_APP\_BASE\_URL;

     //useMsal is hook that returns the PublicClientApplication instance,

     //an array of all accounts currently signed in and an inProgress value

    const { instance } = useMsal();

    const isAuthenticated = useIsAuthenticated();

    useEffect(() => {

        const callbackId = instance.addEventCallback((event) => {

            if (

                (event.eventType === EventType.LOGIN\_SUCCESS || event.eventType === EventType.ACQUIRE\_TOKEN\_SUCCESS) &&

                event.payload.account

            ) {

                 //For the purpose of setting an active account for UI update, we want to consider only the auth

                 //response resulting from SUSI flow. "tfp" claim in the id token tells us the policy (NOTE: legacy

                 //policies may use "acr" instead of "tfp"). To learn more about B2C tokens.

                if (compareIssuingPolicy(event.payload.idTokenClaims, b2cPolicies.names.editProfile)) {

                    // retrieve the account from initial sing-in to the app

                    const originalSignInAccount = instance

                        .getAllAccounts()

                        .find(

                            (account) =>

                                account.idTokenClaims.oid === event.payload.idTokenClaims.oid &&

                                account.idTokenClaims.sub === event.payload.idTokenClaims.sub &&

                                compareIssuingPolicy(account.idTokenClaims, b2cPolicies.names.signUpSignIn)

                        );

                    let signUpSignInFlowRequest = {

                        authority: b2cPolicies.authorities.signUpSignIn.authority,

                        account: originalSignInAccount,

                    };

                    // silently login again with the signUpSignIn policy

                    instance.ssoSilent(signUpSignInFlowRequest);

                }

                 //Below we are checking if the user is returning from the reset password flow.

                 //If so, we will ask the user to reauthenticate with their new password.

                 //If you do not want this behavior and prefer your users to stay signed in instead,

                 //you can replace the code below with the same pattern used for handling the return from

                if (compareIssuingPolicy(event.payload.idTokenClaims, b2cPolicies.names.forgotPassword)) {

                    let signUpSignInFlowRequest = {

                        authority: b2cPolicies.authorities.signUpSignIn.authority,

                    };

                    instance.loginRedirect(signUpSignInFlowRequest);

                }

                //code for inserting user registration

                //Axios call for inserting the userinfo

            }

            if (event.eventType === EventType.LOGIN\_FAILURE) {

                // Check for forgot password error

                // Learn more about AAD error codes at https://docs.microsoft.com/en-us/azure/active-directory/develop/reference-aadsts-error-codes

                if (event.error && event.error.errorMessage.includes('AADB2C90118')) {

                    const resetPasswordRequest = {

                        authority: b2cPolicies.authorities.forgotPassword.authority,

                        scopes: [],

                    };

                    instance.loginRedirect(resetPasswordRequest);

                }

            }

        });

        return () => {

            if (callbackId) {

                instance.removeEventCallback(callbackId);

            }

        };

    }, [instance]);

    return (

        <Routes>

            {

                AppRoutes.map((route, index) => {

                    const { element, ...rest } = route;

                    return <Route key={index} {...rest} element={element} />;

                })

            }

        </Routes >

    );

};

function App({ instance }) {

    return (

        <MsalProvider instance={instance}>

            <Layout>

                <Pages></Pages>

            </Layout>

        </MsalProvider>

    );

}

export default App;

**NavMenu.js**

The app top navigation bar with the sign-in, sign-out, edit profile and call REST API reset buttons.

It uses the **AuthenticatedTemplate** and **UnauthenticatedTemplate**, which only render their children if a user is authenticated or unauthenticated, respectively.

Handle the login and sign out with redirection and popup window events.

The following headers need to import.

import { InteractionStatus } from "@azure/msal-browser";

import { AuthenticatedTemplate, UnauthenticatedTemplate, useMsal } from '@azure/msal-react';

import { b2cPolicies, loginRequest } from '../authConfig';

Msal library functions for login logout and edit profile.

    const handleLoginPopup = () => {

        instance

            .loginPopup({

                ...loginRequest,

                redirectUri: '',

            })

            .catch((error) => console.log(error));

    };

    const handleLogoutPopup = () => {

        instance.logoutPopup({

            mainWindowRedirectUri: '/', // redirects the top level app after logout

        });

    };

    const handleProfileEdit = () => {

        if (inProgress === InteractionStatus.None) {

            instance.acquireTokenRedirect(b2cPolicies.authorities.editProfile);

        }

    };

For the authenticated user *<AuthenticatedTemplate>* should use

<AuthenticatedTemplate>

    <UncontrolledDropdown nav inNavbar>

        <DropdownToggle nav caret>

        </DropdownToggle>

        <DropdownMenu>

            <DropdownItem as="button" onClick={handleProfileEdit}>

                Edit profile

            </DropdownItem>

            <DropdownItem as="button" tag={Link} to="/addresses">

                Your addresses

            </DropdownItem>

            <DropdownItem as="button" onClick={handleLogoutPopup}>

                Sign out

            </DropdownItem>

        </DropdownMenu>

    </UncontrolledDropdown>

</AuthenticatedTemplate>

For the Unauthenticated user *<UnauthenticatedTemplate>* should use

<UnauthenticatedTemplate>

    <div className="collapse navbar-collapse justify-content-end">

        <UncontrolledDropdown nav inNavbar>

            <DropdownToggle nav caret  >

                <img style={{ width: 40, height: 40 }} src="./AdminIcon.png" alt="Admin" />

            </DropdownToggle>

            <DropdownMenu variant="secondary" className="ml-auto" drop="start" title="Sign In" end>

                <DropdownItem as="button" onClick={handleLoginPopup}>

                    Sign in

                </DropdownItem>

            </DropdownMenu>

        </UncontrolledDropdown>

    </div>

</UnauthenticatedTemplate>