<https://medium.com/@ankit4b/a-simplified-guide-to-msal-integration-with-angular-16-24c15cc8130e>

<https://learn.microsoft.com/en-us/credentials/certifications/azure-developer/?practice-assessment-type=certification>

A screenshot of a computer

Description automatically generated

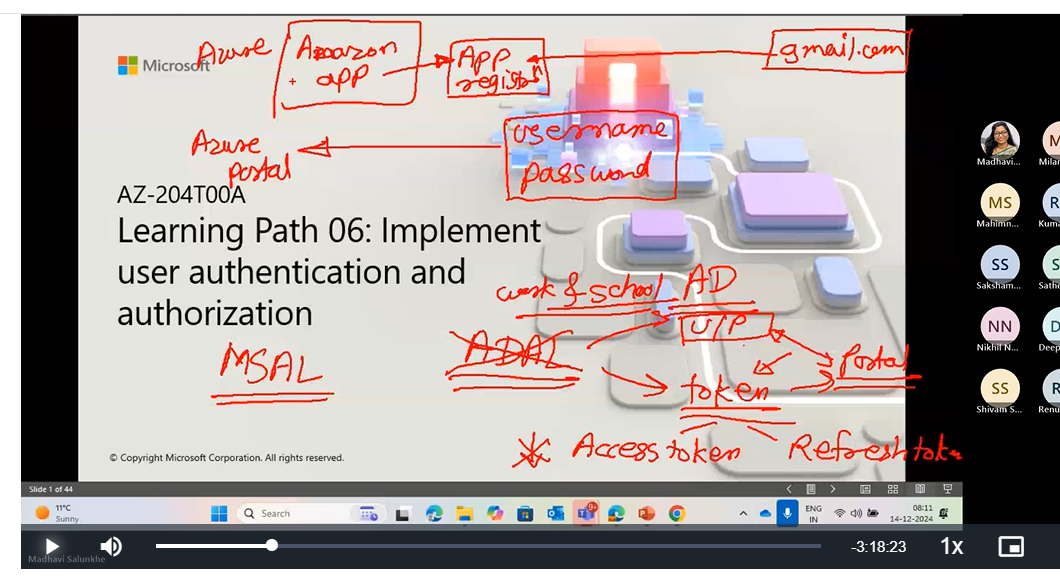
EntraId -🡪 earlier active directory

ADAL-Azure Active Directory Authentication Library🡪 work and school account

A computer screen with a computer screen and text

Description automatically generated with medium confidence

MSAL -> The Microsoft Authentication Library (**MSAL**) enables application developers to acquire tokens in order to call secured web APIs. Can work with all types of identity.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of a company identity platform

Description automatically generated

A diagram of a security system

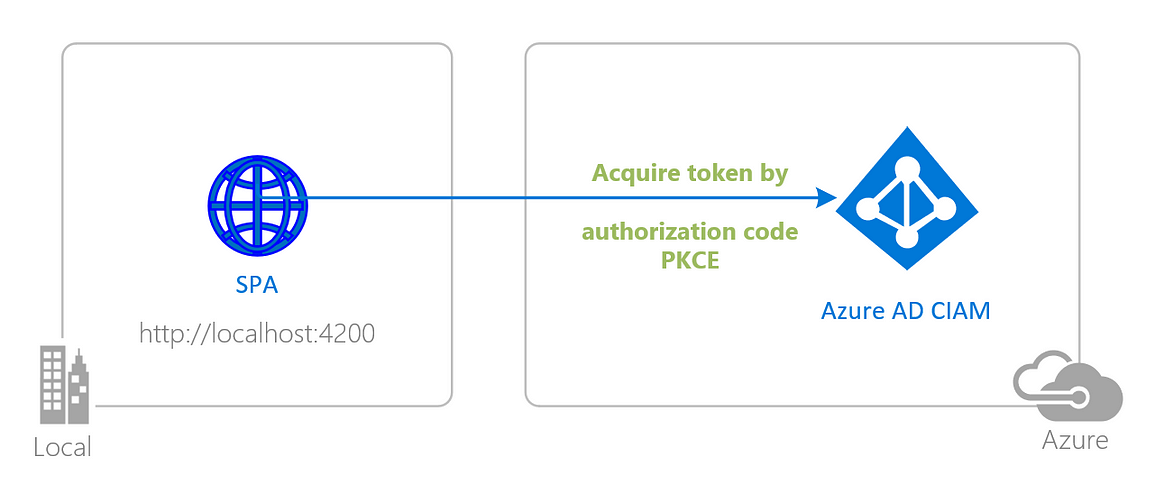
Description automatically generated

MSAL.NET (Microsoft. Identity. Client) is an authentication library that enables you to acquire tokens from Microsoft Entra ID to access protected web APIs (Microsoft APIs or applications registered with Microsoft Entra ID).

Developers can obtain security tokens from the Microsoft Identity Platform to authenticate users and access secure web APIs through the Microsoft Authentication Library (MSAL). This provides fast access to Microsoft Graph, other Microsoft APIs, third-party web APIs, or your web APIs. MSAL is compatible with many application platforms and architectures, including .NET, JavaScript, Java, Python, Android, and iOS.

**A Simplified Guide to MSAL Integration with Angular 16**

This is a step-by-step guide to implementing sign-in with Azure Active Directory (Azure AD) in an Angular single-page application (SPA) using the Microsoft Authentication Library for Angular 16 (MSAL Angular).



**1. Install the dependencies**

Install MSAL Browser and MSAL Angular library in your Angular application.

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

**2. Environment file setup**

Keep all authentication keys in the environment.ts file.

Below Within the ‘msalConfig’ object, authentication details are provided such as  
- ‘clientId’ : representing the unique identifier for the application  
- ‘authority’ : specifying the Azure Active Directory tenant  
- ‘redirectUri’ indicating the URL to redirect users after authentication  
Developers can customize these settings based on their specific application requirements and deployment environment.

export const environment = {  
 //...  
 // msal config details  
 mslConfig: {  
 clientId: "<clientId>",  
 authority: "https://login.microsoftonline.com/<Tenant\_Id>",  
 redirectUri: 'http://localhost:4200/' // change redirect url based on where u want to redirect after the authentication  
 }  
};

**3. app.module.ts file changes for configuration**

Within the ‘app.module.ts’ file, the MSAL configuration is initialized using the provided ‘msalConfig’ object from the environment. This includes specifying the client ID, authority URL, and redirect URI. Additionally, interactions with MSAL are configured for authentication requests and resource protection using ‘MsalModule.forRoot()’

* cacheLocation: “sessionStorage” | “localStorage” — this parameter refers where the developer wants to store the authentication tokens.

import { environment } from 'src/environments/environment';  
import { PublicClientApplication, InteractionType } from "@azure/msal-browser";  
import { MsalModule, MsalRedirectComponent, MsalGuard,  
 MsalInterceptor } from "@azure/msal-angular";  
  
const isIE =  
 window.navigator.userAgent.indexOf("MSIE ") > -1 ||  
 window.navigator.userAgent.indexOf("Trident/") > -1;  
  
const msalConfig = environment.mslConfig;  
  
  
@NgModule({  
 declarations: [  
 ...  
 ],  
 imports: [  
 ...,  
 MsalModule.forRoot(  
 new PublicClientApplication({  
 auth: {  
 clientId: msalConfig.clientId,  
 authority: msalConfig.authority,  
 redirectUri: msalConfig.redirectUri  
 },  
 cache: {  
 cacheLocation: "localStorage",  
 storeAuthStateInCookie: isIE,  
 },  
 }),  
 {  
 interactionType: InteractionType.Redirect, // Msal Guard Configuration  
 authRequest: {  
 scopes: ["user.read"],  
 },  
 },  
 {  
 interactionType: InteractionType.Redirect, // MSAL Interceptor Configuration  
 protectedResourceMap: new Map([  
 ["https://graph.microsoft.com/v1.0/me", ["user.read"]]  
 ]),  
 }  
 ),  
],  
  
 providers: [  
 {  
 provide: HTTP\_INTERCEPTORS,  
 useClass: MsalInterceptor,  
 multi: true,  
 },  
 MsalGuard  
 ],  
 bootstrap: [AppComponent, MsalRedirectComponent]  
})  
export class AppModule { }

**4. Setting up inside components:**

Here we will write the required authentication and login page integration.

**i. app.component.ts file changes:**

Upon loading our project, the following code is executed to check if the user was previously logged in. If the user was logged in, their information is fetched, and the necessary code is executed before redirecting to the respective page. If the user was not logged in, they are redirected to the login component.

import { MsalBroadcastService, MsalService } from '@azure/msal-angular';  
import { EventMessage, EventType, InteractionStatus } from '@azure/msal-browser';  
import { Subject, filter, takeUntil } from 'rxjs';  
  
  
@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css'  
]  
})  
export class AppComponent {  
 ...  
 loginStatus: boolean = false;  
 private readonly \_destroying$ = new Subject<void>();  
  
 constructor(private broadcastService: MsalBroadcastService, private msalService: MsalService, private route: Router) {  
  
 this.broadcastService.msalSubject$  
 .pipe(  
 filter((msg: EventMessage) => msg.eventType === EventType.ACCOUNT\_ADDED || msg.eventType === EventType.ACCOUNT\_REMOVED),  
 )  
 .subscribe((result: EventMessage) => {  
 if (this.msalService.instance.getAllAccounts().length === 0) {  
 // Redirect to Login page  
 this.route.navigate(['/login']);  
 } else {  
 this.setLoginDisplay();  
 }  
 });  
  
 this.broadcastService.inProgress$  
 .pipe(  
 filter((status: InteractionStatus) => status === InteractionStatus.None),  
 takeUntil(this.\_destroying$)  
 )  
 .subscribe(() => {  
 this.setLoginDisplay();  
 })  
 }  
  
  
 setLoginDisplay() {  
 this.loginStatus = this.msalService.instance.getAllAccounts().length > 0;  
  
 if(this.loginStatus){  
 const profile\_data = this.msalService.instance.getAllAccounts()[0];  
 this.msalService.instance.setActiveAccount(profile\_data)  
 } else {  
 console.log("Not logged-in");  
 // Redirect to Login page  
 this.route.navigate(['/login']);  
 }  
  
 }  
  
  
 ngOnDestroy(): void {  
 this.\_destroying$.next(undefined);  
 this.\_destroying$.complete();  
 }  
}

**ii. auth.service.ts file changes**

We also require an auth-service file. You can generate this service file using the following command: “ng g s services/auth”. Afterwards, add the methods below that we will utilize further.

import { Inject, Injectable } from '@angular/core';  
import { MSAL\_GUARD\_CONFIG, MsalGuardConfiguration, MsalService } from '@azure/msal-angular';  
import { RedirectRequest } from '@azure/msal-browser';  
  
  
@Injectable({  
 providedIn: 'root'  
})  
export class AuthService {  
 profile: any;  
  
 constructor(@Inject(MSAL\_GUARD\_CONFIG) private msalGuardConfig: MsalGuardConfiguration, private msalService: MsalService) { }  
  
 getUserDetails(){  
 const profileData = this.msalService.instance.getAllAccounts()[0];  
 this.profile = {name: profileData.name, mail: profileData.username}  
 return this.profile;  
 }  
  
 microsoftLogin() {  
 if (this.msalGuardConfig.authRequest){  
 this.msalService.loginRedirect({...this.msalGuardConfig.authRequest} as RedirectRequest);  
 } else {  
 this.msalService.loginRedirect();  
 }  
 }  
  
 microsoftLogout() {  
 this.msalService.logoutRedirect({  
 postLogoutRedirectUri: location.origin  
 });  
 }  
}

**iii. Login-component implementation**

On this page, we will create a Login button. Upon clicking this button, users will be redirected to Microsoft login. Upon successful login, they will be redirected back to the configured redirect URL.

// login.component.ts  
  
import { Component } from '@angular/core';  
import { AuthService } from 'path-to-your-auth-service'; // Import the AuthService  
  
@Component({  
 selector: 'app-login',  
 templateUrl: './login.component.html',  
 styleUrls: ['./login.component.css']  
})  
export class LoginComponent {  
 constructor(private authService: AuthService) {} // Inject the AuthService  
  
 MsLogin() {  
 this.authService.microsoftLogin(); // Call the microsoftLogin() method from the AuthService  
 }  
}  
  
  
<!-- login.component.html -->  
  
<button (click)="MsLogin()">Login</button>

**5. Routing Module Setup**

In the Routing Module, you can define navigation paths/routes according to your requirements, specifying which pages require MsalGuard and which paths should be accessible to all users.

import { NgModule } from '@angular/core';  
import { RouterModule, Routes } from '@angular/router';  
//... import required components  
import { HomeComponent } from './home/home.component';  
import { LoginComponent } from './login/login.component';  
import { MsalGuard } from '@azure/msal-angular';  
  
const routes: Routes = [  
 {  
 path: '',  
 component: HomeComponent,  
 canActivate: [MsalGuard] // Use MsalGuard for the Home component  
 },  
 {  
 path: 'login',  
 component: LoginComponent  
 }  
];  
  
@NgModule({  
 imports: [RouterModule.forRoot(routes, { useHash: false })],  
 exports: [RouterModule]  
})  
export class AppRoutingModule { }

**6. Add <app-redirect> on index.html**

<app-redirect></app-redirect> component is often used as a placeholder for handling the redirect flow after authentication.

<!doctype html>  
<html lang="en">  
<head>  
…  
</head>  
<body>  
 <app-root></app-root>  
 // Add app-redirect tag  
 <app-redirect></app-redirect>  
</body>  
</html>