**Implementing Single Sign-On with Azure AD for iOS Applications –**

**Detailed Design Document**

**Version 1.0**

**Monday, 02 March 2020**

**REVISION LOG**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes done** | **Comments** |
| 1.0 | Monday, 02 March 2020 |  | First Draft | Initial Draft |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**DETAILED DESIGN DOCUMENT SIGN OFF**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Name** | **Position/Role** | **Date** | **Signature** |
|  |  | Architecture / Project Owner |  |  |
|  |  | IT Infrastructure Manager / Project Advisor |  |  |

Contents

[1. Introduction 5](#_Toc24373275)

[1.1. Purpose 5](#_Toc24373276)

[1.2. Scope 5](#_Toc24373277)

[1.3. Project Vision and Scope Summary 6](#_Toc24373278)

[1.4. Intended Audience 6](#_Toc24373279)

[1.5. Acronyms and Definitions 7](#_Toc24373280)

[2. Design Assumption 7](#_Toc24373281)

[3. Register your Application 8](#_Toc24373282)

[5. Run the Application 18](#_Toc24373283)

[6. Authentication flow using MSAL 22](#_Toc24373284)

[7. Appendix 23](#_Toc24373285)

[7.1. Reference URLS 23](#_Toc24373286)

# Introduction

## Purpose

The purpose of this document is to outline requirement integrate an iOS app with the Microsoft identity platform. Microsoft [identity platform](https://docs.microsoft.com/azure/active-directory/develop/) also known as the Azure Active Directory v2 Endpoint.

The MSAL library gives your app the ability to begin using the [Microsoft identity platform](https://aka.ms/aaddev) by supporting [Azure Active Directory](https://azure.microsoft.com/services/active-directory/) and [Microsoft Accounts](https://account.microsoft.com/) in a converged experience using industry standard OAuth2 and OpenID Connect protocols.

The Microsoft Authentication Library (MSAL) for iOS is an Auth SDK that can be used to seamlessly integrate authentication into your iOS app and give access to the entire Microsoft ecosystem.

This document is guide to help application owners and vendors to move the apps to SSO on Azure AD

## Scope

The scope of this document is to detail out the complete design and architecture for the iOS application that uses Microsoft Authentication library for iOS (MSAL) to implement Authentication with Active Directory (Azure AD).

MSAL will automatically renew tokens, deliver single sign-on (SSO) between other apps on the device, and manage the Account(s).

This document covers how to integrate existing iOS application that uses MSAL authentication to sign in users from a single Azure Active Directory. All the design and development changes made will be covered in this document.

This document is a step-by-step guide for implementation and configuration.

the various lifecycle events of your app to achieve the following objectives.

* Sign-in a user
* Device-wide SSO and Conditional Access support through the Auth Broker
* Select between Single Account Mode and Multiple Account Mode
* Get a token for the [Microsoft Graph](https://graph.microsoft.com/)
* Sign out the user

## Project Vision and Scope Summary

The vision and primary goal of the solution is to Integrate Azure AD into an iOS application using MSAL authentication. Based on the initial project kick-off scope below is the vision and scope summary of this project

* + **Assessment**
* Detailed assessment of tenant’s Azure Active Directory environment
* Assessment of technical requirements around existing applications, frameworks and databases
  + **Designing and development**
* Design will be done for Integrating Azure AD into existing iOS application using MSAL authentication
  + **Development**
* Steps for configuring single sign-on for iOS applications using organizational authentication.
  + **Azure App Registration**
* Registration of the existing application with your Azure Active Directory tenant
  + **Security best practices**
* Single sign-on (SSO) adds security and convenience when users sign-on to applications in Azure Active Directory (Azure AD). Administrators can centralize user account management, and automatically add or remove user access to applications based on group membership.
  + **Single Sign-On in all the applications**
* Azure Active Directory Seamless Single Sign-On (Azure AD Seamless SSO) automatically signs users in when they are on their corporate devices connected to your corporate network.
* When enabled, users don't need to type in their passwords to sign in to Azure AD, and usually, even type in their usernames. This feature provides your users easy access to your cloud-based applications without needing any additional on-premises components.

## Intended Audience

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Department/**  **Team** |
| Application owners |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Acronyms and Definitions

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| AD | Active Directory |
| AAD | Azure Active Directory |
| SSO | Single Sign-On |
| MSAL | Microsoft Authentication library |
| SSL | Secure Sockets Layer |
| URL / URI | Uniform Resource Locator / Identifier |
| B2C | business-to-consumer |

# Design Assumption

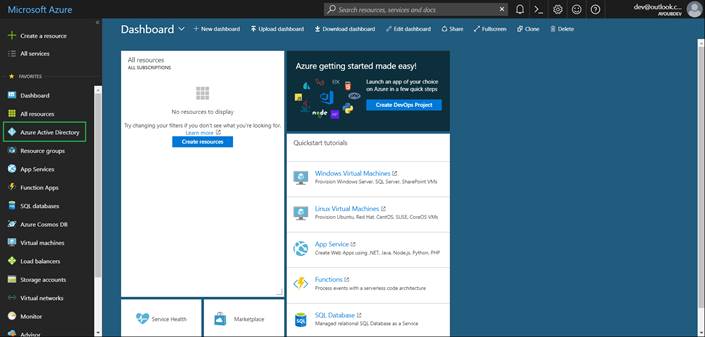
The following assumptions have been made while suggesting the proposed Azure Active Directory Seamless Single Sign-On (Azure AD Seamless SSO)

| Assumption |
| --- |
| iOS Application |
| XCode version 10.x or greater |
| Access to code for modification |
| Access to database for testing |
| Microsoft Azure Subscription |
| Microsoft Azure Active Directory |
| Admin access to Azure portal |

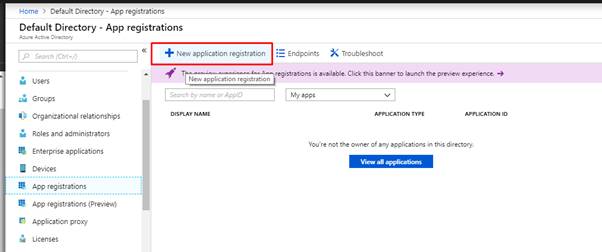
# Register your Application

Follow the below-listed steps to register the application.

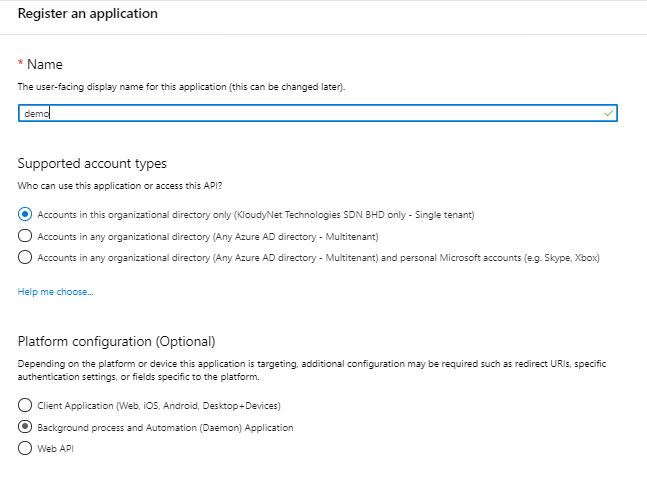
* Sign in to the [Azure portal](https://portal.azure.com/) using a work Microsoft account.
* If your account gives you access to more than one tenant, select your account in the top right corner, and set your portal session to the Azure AD tenant that you want.
* In the left-hand navigation pane, select the Azure Active Directory service.



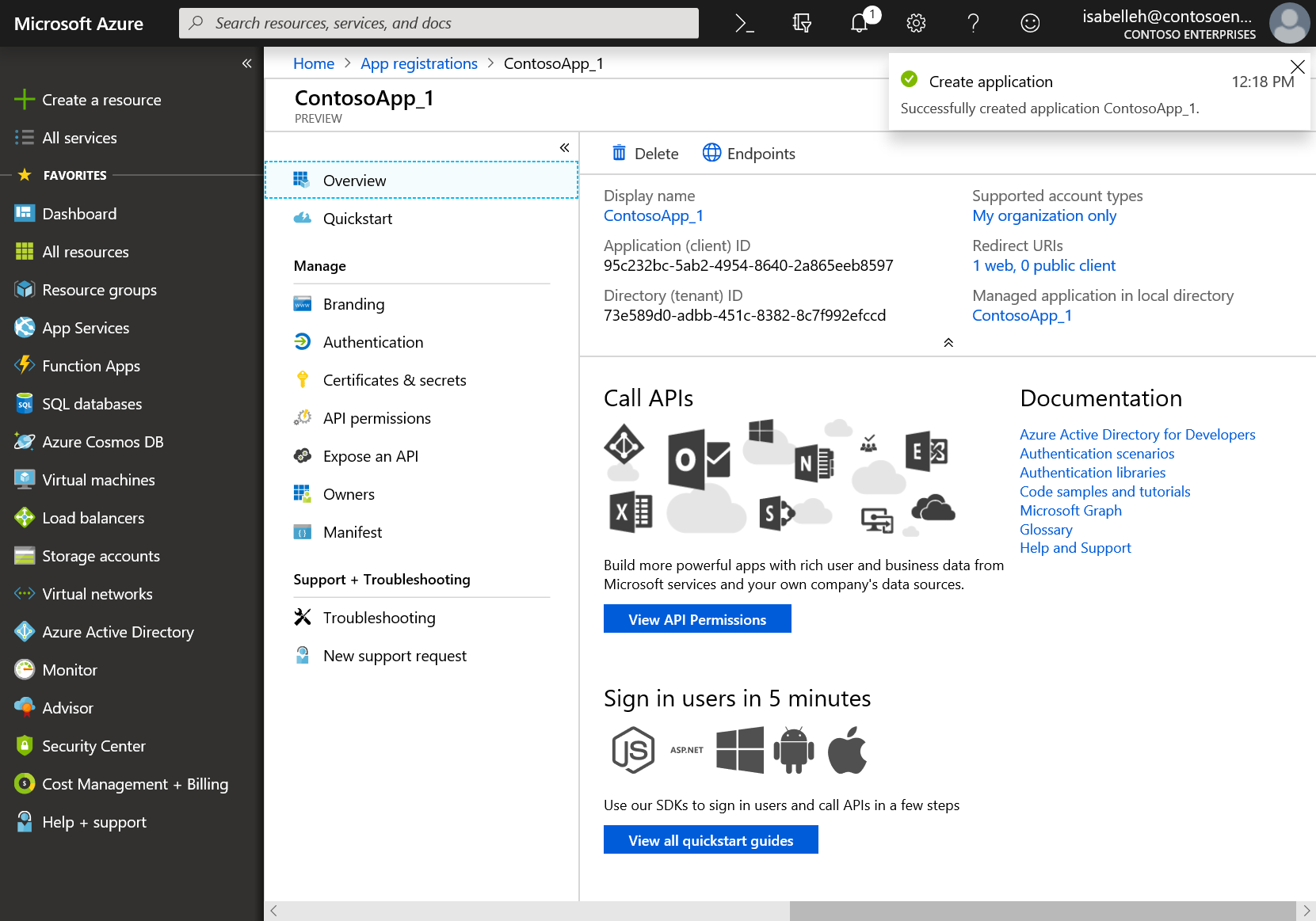
* Then select App registrations > New registration.



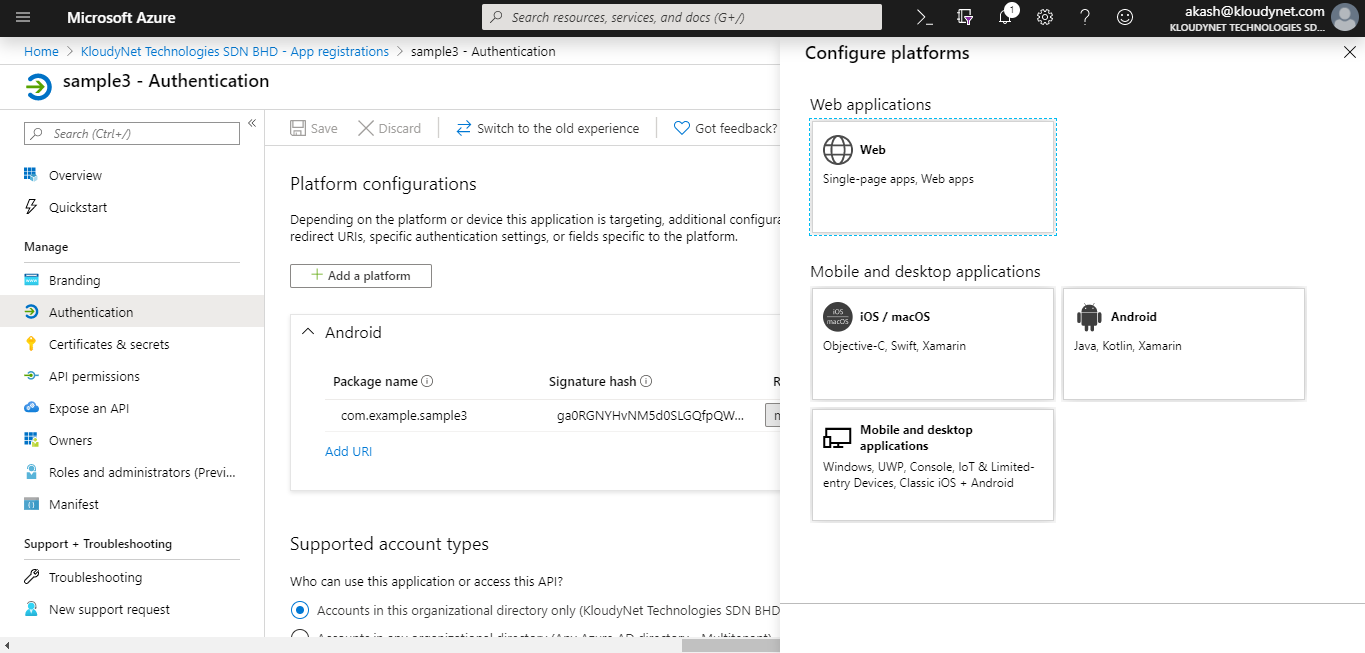
* When we Register an application page appears, enter your application's registration information:
  + **Name** - Enter a meaningful application name that will be displayed to users of the app.
  + **Supported account types** - Select which accounts you would like your application to support.
  + **Platform configuration** (Optional)- Ignore this section and without setting a Redirect URI, click **Register**.



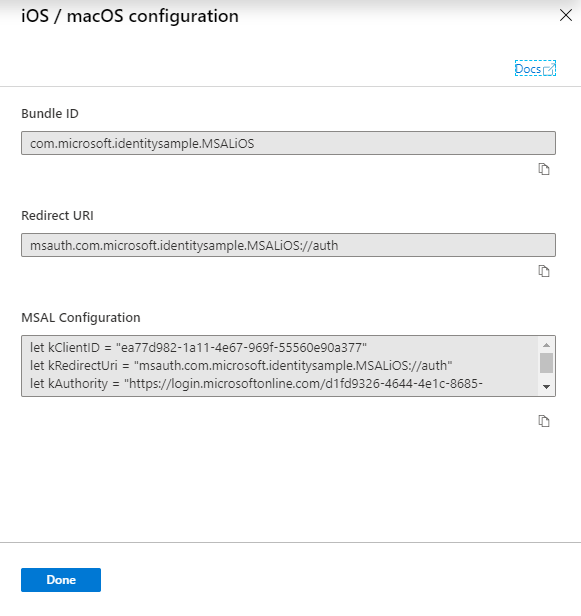
* When finished, select Register.



* In the Manage section of the pane that appears, select Authentication > + Add a platform > iOS/macOS. (You may have to select "Switch to the new experience" near the top of the blade to see this section)



* Enter your project's Bundle ID
* The bundle identifier appears in the Identity section
* Click Configure and save the MSAL Configuration that appears in the iOS configuration page so you can enter it when you configure your app later. Click Done.



1. **Configuring your Application**
   1. Installing the MSAL library in your app

* Install MSAL by first creating an empty file called podfile in the same folder as your project's .xcodeproj file. Add the following to podfile:

use\_frameworks!

target '<your-target-here>' do

pod 'MSAL'

end

* Replace <your-target-here> with the name of your project.
* In a terminal window, navigate to the folder that contains the podfile you created and run pod install to install the MSAL library.
* Close Xcode and open <your project name>.xcworkspace to reload the project in Xcode.
  1. Adding MSAL to your project
* Add the following import statement to the top of the ViewController.swift and AppDelegate.swift files

import MSAL

Add the following code to ViewController.swift prior to viewDidLoad()

let kClientID = "Your\_Application\_Id\_Here"

// Additional variables for Auth and Graph API

let kGraphURI = "https://graph.microsoft.com/v1.0/me/"

let kScopes: [String] = ["https://graph.microsoft.com/user.read"]

let kAuthority = "https://login.microsoftonline.com/common"

var accessToken = String()

var applicationContext : MSALPublicClientApplication?

var webViewParamaters : MSALWebviewParameters?

* Modify the value assigned to kClientID to be your Application ID. This value is part of the MSAL Configuration data that you saved during the step at the beginning to register the application in the Azure portal.
  1. Configuring URL schemes
* Register CFBundleURLSchemes so that the user can be redirected back to the app after sign in.
* LSApplicationQueriesSchemes allows your app to make use of Microsoft Authenticator.
* In Xcode, open Info.plist as a source code file, and add the following inside of the <dict> section.
* Replace [BUNDLE\_ID] with the value you used in the Azure portal, the bundle identifier appears in the Identity section.

<key>CFBundleURLTypes</key>

<array>

<dict>

<key>CFBundleURLSchemes</key>

<array>

<string>msauth.[BUNDLE\_ID]</string>

</array>

</dict>

</array>

<key>LSApplicationQueriesSchemes</key>

<array>

<string>msauthv2</string>

<string>msauthv3</string>

</array>

* Inside the ViewController class, replace the viewDidLoad() method with

override func viewDidLoad() {

super.viewDidLoad()

initUI()

do {

try self.initMSAL()

} catch let error {

self.updateLogging(text: "Unable to create Application Context \(error)")

}

}

* 1. Initializing MSAL
* Add the following initMSAL method to the ViewController class

func initMSAL() throws {

guard let authorityURL = URL(string: kAuthority) else {

self.updateLogging(text: "Unable to create authority URL")

return

}

let authority = try MSALAADAuthority(url: authorityURL)

let msalConfiguration = MSALPublicClientApplicationConfig(clientId: kClientID, redirectUri: nil, authority: authority)

self.applicationContext = try MSALPublicClientApplication(configuration: msalConfiguration)

self.initWebViewParams()

}

* Add the following after initMSAL method to the ViewController class.

func initWebViewParams() {

self.webViewParamaters = MSALWebviewParameters(parentViewController: self)

}

* Open the AppDelegate.swift file. To handle the callback after sign-in, add MSALPublicClientApplication.handleMSALResponse to the appDelegate class

// Inside AppDelegate...

func application(\_ app: UIApplication, open url: URL, options: [UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {

return MSALPublicClientApplication.handleMSALResponse(url, sourceApplication: options[UIApplication.OpenURLOptionsKey.sourceApplication] as? String)

}

* **If you are using Xcode 11**, you should place MSAL callback into the SceneDelegate.swift instead. If you support both UISceneDelegate and UIApplicationDelegate for compatibility with older iOS, MSAL callback would need to be placed into both files.

func scene(\_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>) {

guard let urlContext = URLContexts.first else {

return

}

let url = urlContext.url

let sourceApp = urlContext.options.sourceApplication

MSALPublicClientApplication.handleMSALResponse(url, sourceApplication: sourceApp)

}

* 1. Acquiring tokens interactively

Implementing the application's UI processing logic and get tokens interactively through MSAL.

* acquireTokenInteractively() always shows UI when attempting to sign in the user. It may use session cookies in the browser or an account in the Microsoft authenticator to provide an interactive-SSO experience.
* Add the following code to the ViewController class:

@objc func callGraphAPI(\_ sender: AnyObject) {

guard let currentAccount = self.currentAccount() else {

// We check to see if we have a current logged in account.

// If we don't, then we need to sign someone in.

acquireTokenInteractively()

return

}

acquireTokenSilently(currentAccount)

}

func currentAccount() -> MSALAccount? {

guard let applicationContext = self.applicationContext else { return nil }

// We retrieve our current account by getting the first account from cache

// In multi-account applications, account should be retrieved by home account identifier or username instead

do {

let cachedAccounts = try applicationContext.allAccounts()

if !cachedAccounts.isEmpty {

return cachedAccounts.first

}

} catch let error as NSError {

self.updateLogging(text: "Didn't find any accounts in cache: \(error)")

}

return nil

}

* 1. Getting the token interactively
* The code below gets a token for the first time by creating an MSALInteractiveTokenParameters object and calling acquireToken.
  + - * 1. Creates MSALInteractiveTokenParameters with scopes.
        2. Calls acquireToken() with the created parameters.
        3. Handles errors.
        4. Handles the successful case.
* Add the following code to the ViewController class.

func acquireTokenInteractively() {

guard let applicationContext = self.applicationContext else { return }

guard let webViewParameters = self.webViewParamaters else { return }

// #1

let parameters = MSALInteractiveTokenParameters(scopes: kScopes, webviewParameters: webViewParameters)

// #2

applicationContext.acquireToken(with: parameters) { (result, error) in

// #3

if let error = error {

self.updateLogging(text: "Could not acquire token: \(error)")

return

}

guard let result = result else {

self.updateLogging(text: "Could not acquire token: No result returned")

return

}

// #4

self.accessToken = result.accessToken

self.updateLogging(text: "Access token is \(self.accessToken)")

self.updateSignOutButton(enabled: true)

self.getContentWithToken()

}

}

* 1. Calling the Microsoft Graph API
* Once you have a token, your app can use it in the HTTP header to make an authorized request to the Microsoft Graph:

| header key | value |
| --- | --- |
| Authorization | Bearer <access-token> |

* Add the following code to the ViewController class

func getContentWithToken() {

// Specify the Graph API endpoint

let url = URL(string: kGraphURI)

var request = URLRequest(url: url!)

// Set the Authorization header for the request. We use Bearer tokens, so we specify Bearer + the token we got from the result

request.setValue("Bearer \(self.accessToken)", forHTTPHeaderField: "Authorization")

URLSession.shared.dataTask(with: request) { data, response, error in

if let error = error {

self.updateLogging(text: "Couldn't get graph result: \(error)")

return

}

guard let result = try? JSONSerialization.jsonObject(with: data!, options: []) else {

self.updateLogging(text: "Couldn't deserialize result JSON")

return

}

self.updateLogging(text: "Result from Graph: \(result))")

}.resume()

}

* 1. MSAL for Sign Out
* Signing out with MSAL removes all known information about a user from the application, but the user will still have an active session on their device. If the user attempts to sign in again they may see sign-in UI, but may not need to reenter their credentials because the device session is still active.
* To add sign-out capability, add the following code inside the ViewController class. This method cycles through all accounts and removes them:

@objc func signOut(\_ sender: AnyObject) {

guard let applicationContext = self.applicationContext else { return }

guard let account = self.currentAccount() else { return }

do {

/\*\*

Removes all tokens from the cache for this application for the provided account

- account: The account to remove from the cache

\*/

try applicationContext.remove(account)

self.updateLogging(text: "")

self.updateSignOutButton(enabled: false)

self.accessToken = ""

} catch let error as NSError {

self.updateLogging(text: "Received error signing account out: \(error)")

}

}

* 1. Adding Helper Methods

Add the following helper methods to the ViewController class

func updateLogging(text : String) {

if Thread.isMainThread {

self.loggingText.text = text

} else {

DispatchQueue.main.async {

self.loggingText.text = text

}

}

}

func updateSignOutButton(enabled : Bool) {

if Thread.isMainThread {

self.signOutButton.isEnabled = enabled

} else {

DispatchQueue.main.async {

self.signOutButton.isEnabled = enabled

}

}

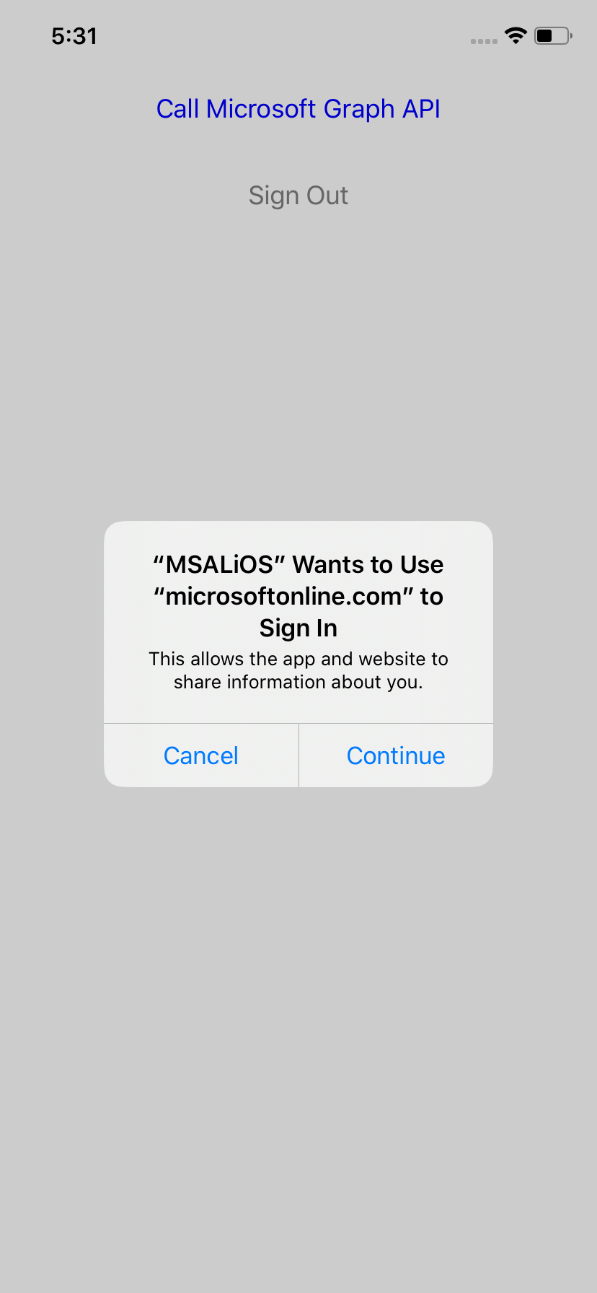
}

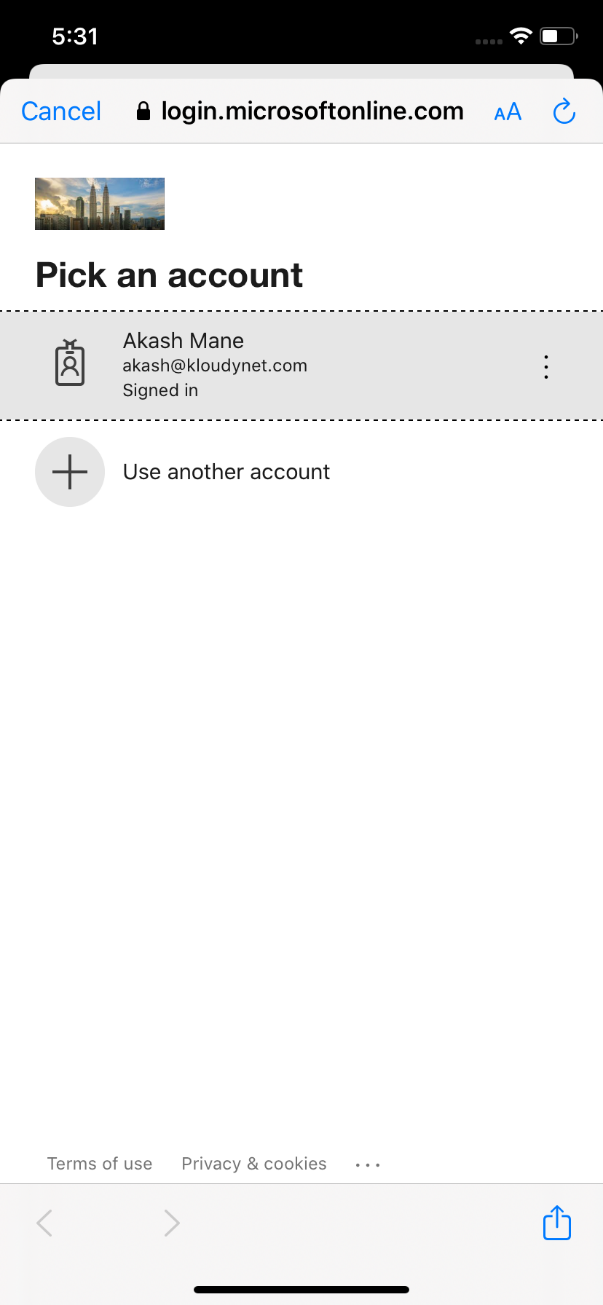
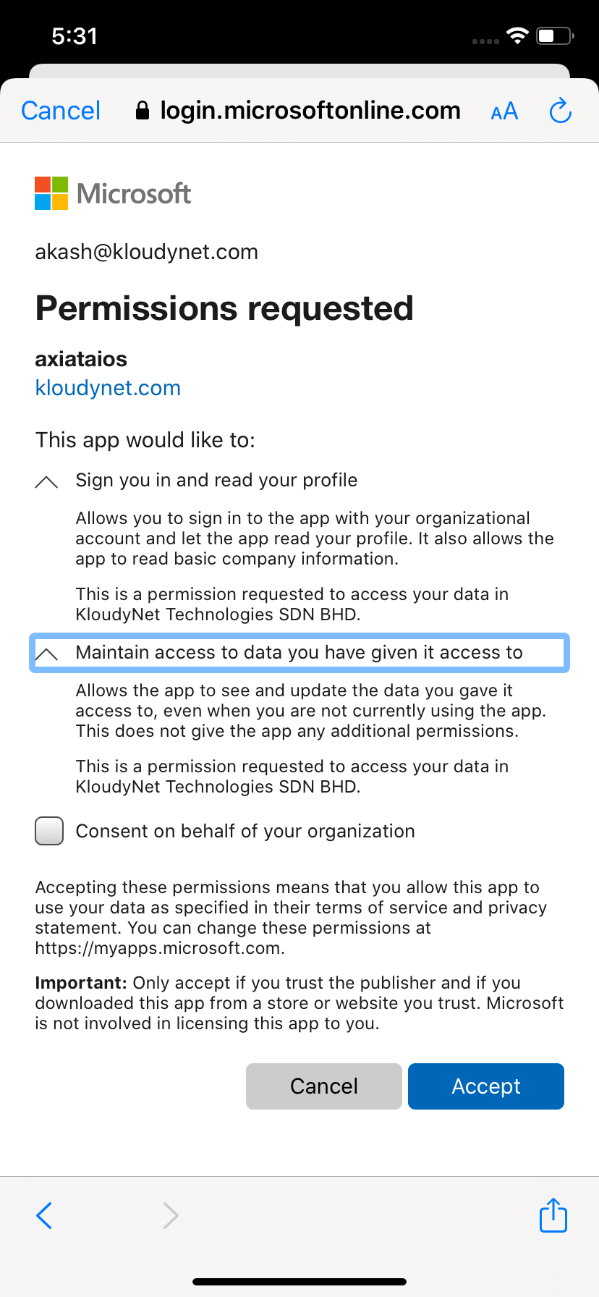
# Run the Application

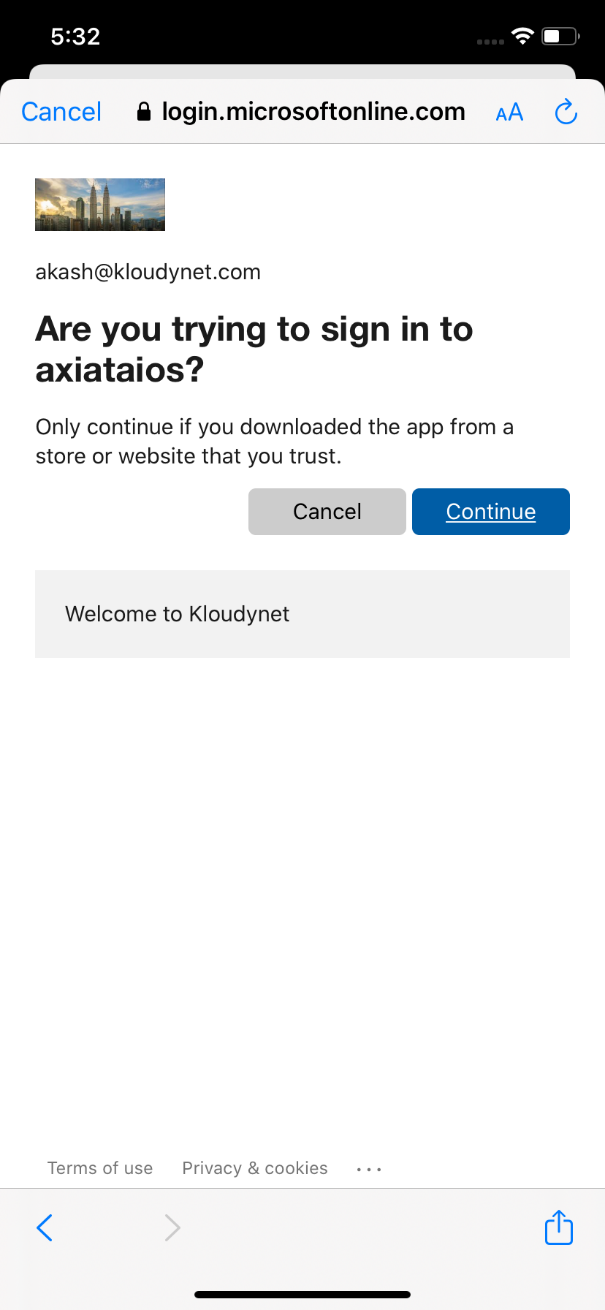
Clean the solution, rebuild the solution, and run it.

Build and deploy the app to a test device or simulator. You should be able to sign in and get tokens for Azure AD or personal Microsoft accounts.

The first time a user signs into your app, they will be prompted by Microsoft identity to consent to the permissions requested.

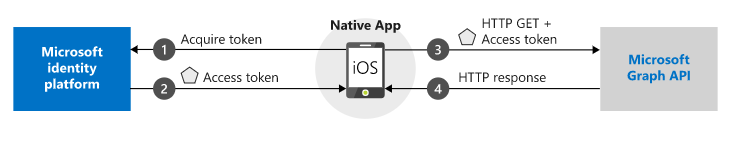
 

# Authentication flow using MSAL

The most basic sign-in flow contains the following steps - each of them is described in detail below diagram



# Appendix

# Reference URLS

<https://docs.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-ios>