**Implementing Single Sign-On with Azure AD for Android 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](#_Toc24373084)

[1.1. Purpose 5](#_Toc24373085)

[1.2. Scope 5](#_Toc24373086)

[1.3. Project Vision and Scope Summary 6](#_Toc24373087)

[1.4. Intended Audience 6](#_Toc24373088)

[1.5. Acronyms and Definitions 7](#_Toc24373089)

[2. Design Assumption 7](#_Toc24373090)

[3. Register your Application 8](#_Toc24373091)

[5. Run the Application 16](#_Toc24373092)

[6. Authentication flow using MSAL 18](#_Toc24373093)

[7. Appendix 19](#_Toc24373094)

[7.1. Reference URLS 19](#_Toc24373095)

# Introduction

## Purpose

The purpose of this document is to outline requirement integrate an Android 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 Android 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 Android is an Auth SDK that can be used to seamlessly integrate authentication into your Android 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 Android application that uses Microsoft Authentication library for Android (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 Android 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 Android 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 Android application using MSAL authentication
  + **Development**
* Steps for configuring single sign-on for Android 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

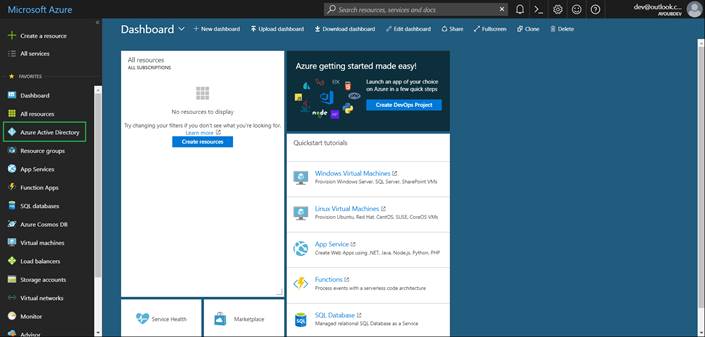
The following assumptions have been made while suggesting the proposed Azure Active Directory Seamless Single Sign-On (Azure AD Seamless SSO)

| 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 |

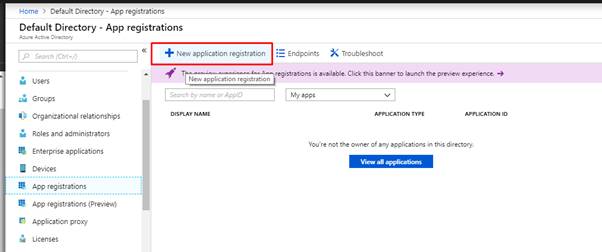
# Register your Application

Follow the below-listed steps to register the application.

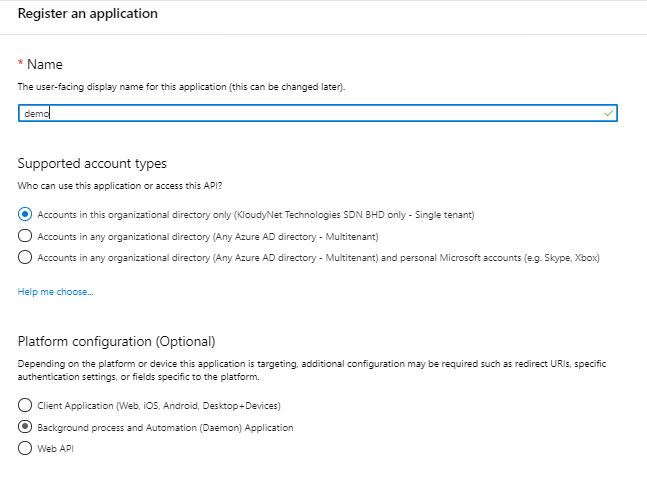
* Sign in to the [Azure portal](https://portal.azure.com/) using a work Microsoft account.
* If your account gives you access to more than one tenant, select your account in the top right corner, and set your portal session to the Azure AD tenant that you want.
* In the left-hand navigation pane, select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



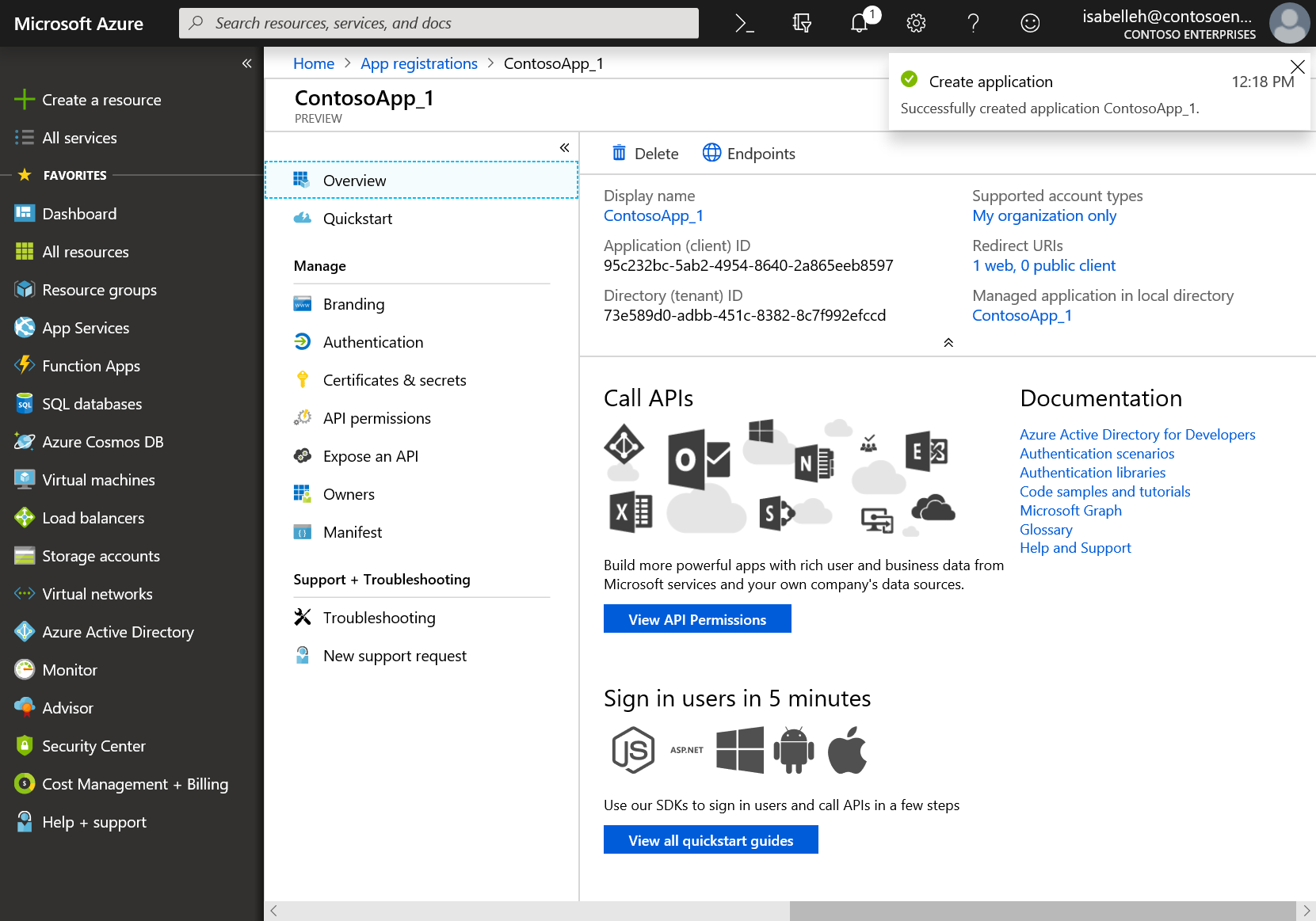
* Then select App registrations > New registration.



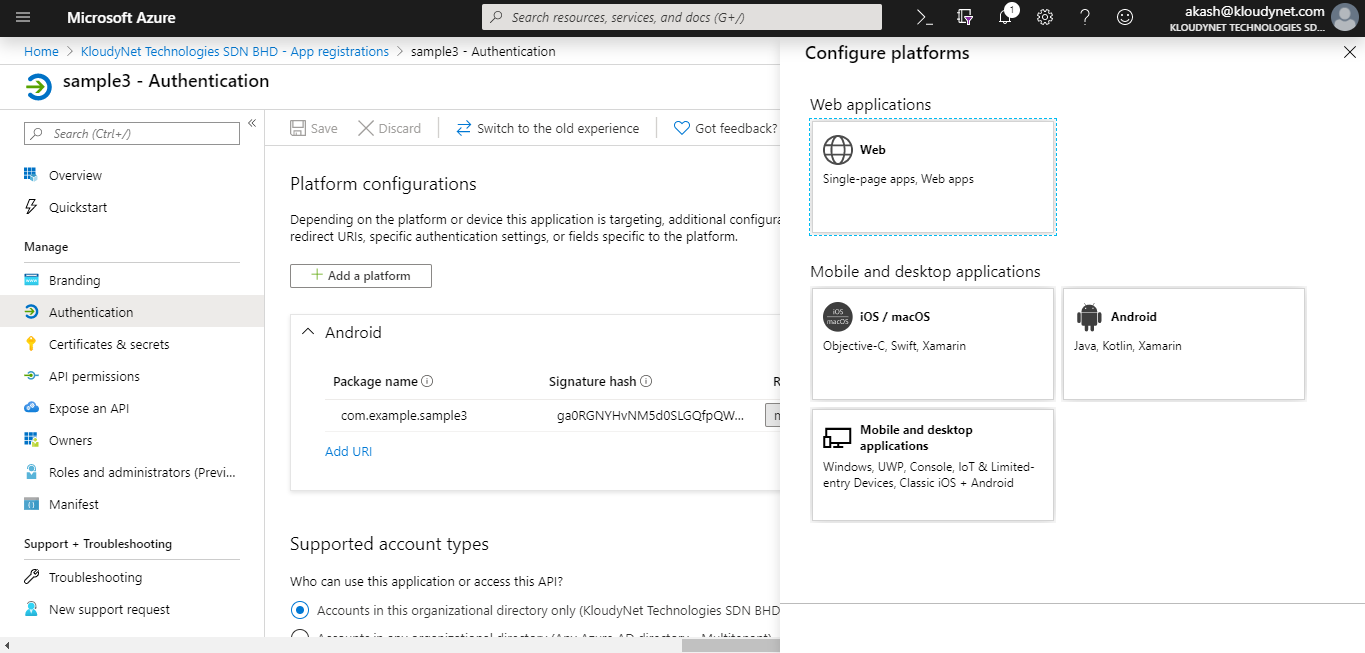
* When we Register an application page appears, enter your application's registration information:
  + **Name** - Enter a meaningful application name that will be displayed to users of the app.
  + **Supported account types** - Select which accounts you would like your application to support.
  + **Platform 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 > Android. (You may have to select "Switch to the new experience" near the top of the blade to see this section)



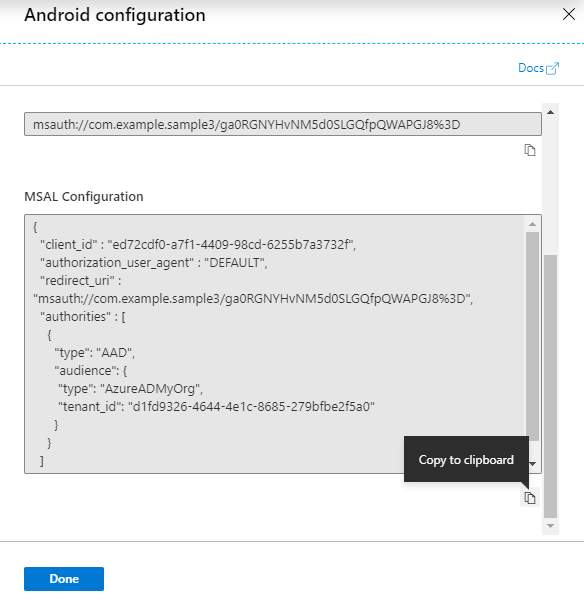
* Install the OpenSSL tool in your C drive (C:\) from below URL.

<https://code.google.com/archive/p/openssl-for-windows/downloads>

* Find the JRE path on your computer to run the command for the next step, as KeyTool.exe is installed as part of the Java Development Kit (JDK)
* Replace the correct username and paths of your computer with the command below and then run the same.

C:\Program Files\Android\Android Studio\jre\bin>keytool -exportcert -alias androiddebugkey -keystore C:\Users\<compuer username>\.android\debug.keystore | C:\openssl-0.9.8k\_X64\bin\openssl sha1 -binary | C:\openssl-0.9.8k\_X64\bin\openssl base64

* After running the command, you will be presented with the Signature has. Note down the signature hash.
* On the 'Configure your Android App' page
  1. Enter the Package Name from your Android Manifest.
  2. Enter the Signature Hash.
  3. Click Configure at the bottom of the page.
* Take note of the MSAL Configuration as it is used later.

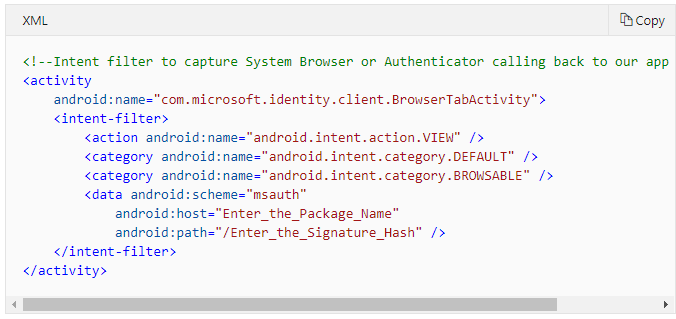


1. **Configuring your Application**
   1. Adding you App Registration

* In Android Studio's project pane, navigate to app\src\main\res.
* If folder raw not present under res the create by right-clicking res and choose New > Directory. Enter raw as the new directory name and click OK.
* In app > src > main > res > raw, create a new JSON file called auth\_config.json and paste the MSAL Configuration that you saved earlier.
* `auth\_config.json` contains this information as a reference inside the `redirect\_uri` field.
* The Signature Hash should NOT be URL encoded in the `AndroidManifest.xml`.



* In **app** > **src** > **main** > **AndroidManifest.xml**, add the BrowserTabActivity activity below to the application body. This entry allows Microsoft to call back to your application after it completes the authentication:



* Substitute the package name you registered in the Azure portal for the android:host= value.
* Substitute the key hash you registered in the Azure portal for the android:path= value.
* The Signature Hash should not be URL encoded.
* Inside the AndroidManifest.xml, just above the <application> tag, add the following permissions:

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

* 1. Adding MSAL to your project
* In the Android Studio project window, navigate to app > src > build.gradle.
* Under Dependencies, paste the following:

implementation **'com.android.volley:volley:1.1.1'  
  
if** (findProject(**':msal'**) != **null**) {  
 *// For developer team only.* localImplementation project(**':msal'**)  
 externalImplementation **'com.microsoft.identity.client:msal:1.0.+'**}  
**else** {  
 *// Downloads and Builds MSAL from maven central.* implementation **'com.microsoft.identity.client:msal:1.0.+'**}

* 1. Adding the Sign-In experience to your existing implementation
* Add the following imports near the top of your java file where your existing sign in logic is coded:

import android.app.Activity;

import android.content.Intent;

import android.util.Log;

import android.view.View;

import android.widget.Button;

import android.widget.TextView;

import android.widget.Toast;

import com.android.volley.\*;

import com.android.volley.toolbox.JsonObjectRequest;

import com.android.volley.toolbox.Volley;

import org.json.JSONObject;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.microsoft.identity.client.\*;

import com.microsoft.identity.client.exception.\*;

* + Create a PublicClientApplication in the same java file:

PublicClientApplication.*createSingleAccountPublicClientApplication*(getContext(),  
 R.raw.*auth\_config\_single\_account*,  
 new IPublicClientApplication.ISingleAccountApplicationCreatedListener() {  
 @Override  
 public void onCreated(ISingleAccountPublicClientApplication application) {  
 */\*\*  
 \* This test app assumes that the app is only going to support one account.  
 \* This requires "account\_mode" : "SINGLE" in the config json file.  
 \*\*/* mSingleAccountApp = application;  
 loadAccount();  
 }  
  
 @Override  
 public void onError(MsalException exception) {  
 displayError(exception);  
 }  
 });

* Signing in a user, adding the sign-in logic inside your existing SignIn button logic

signInButton.setOnClickListener(new View.OnClickListener() {  
 public void onClick(View v) {  
 if (mSingleAccountApp == null) {  
 return;  
 }  
  
 mSingleAccountApp.signIn(getActivity(), null, getScopes(), getAuthInteractiveCallback());  
 }  
});

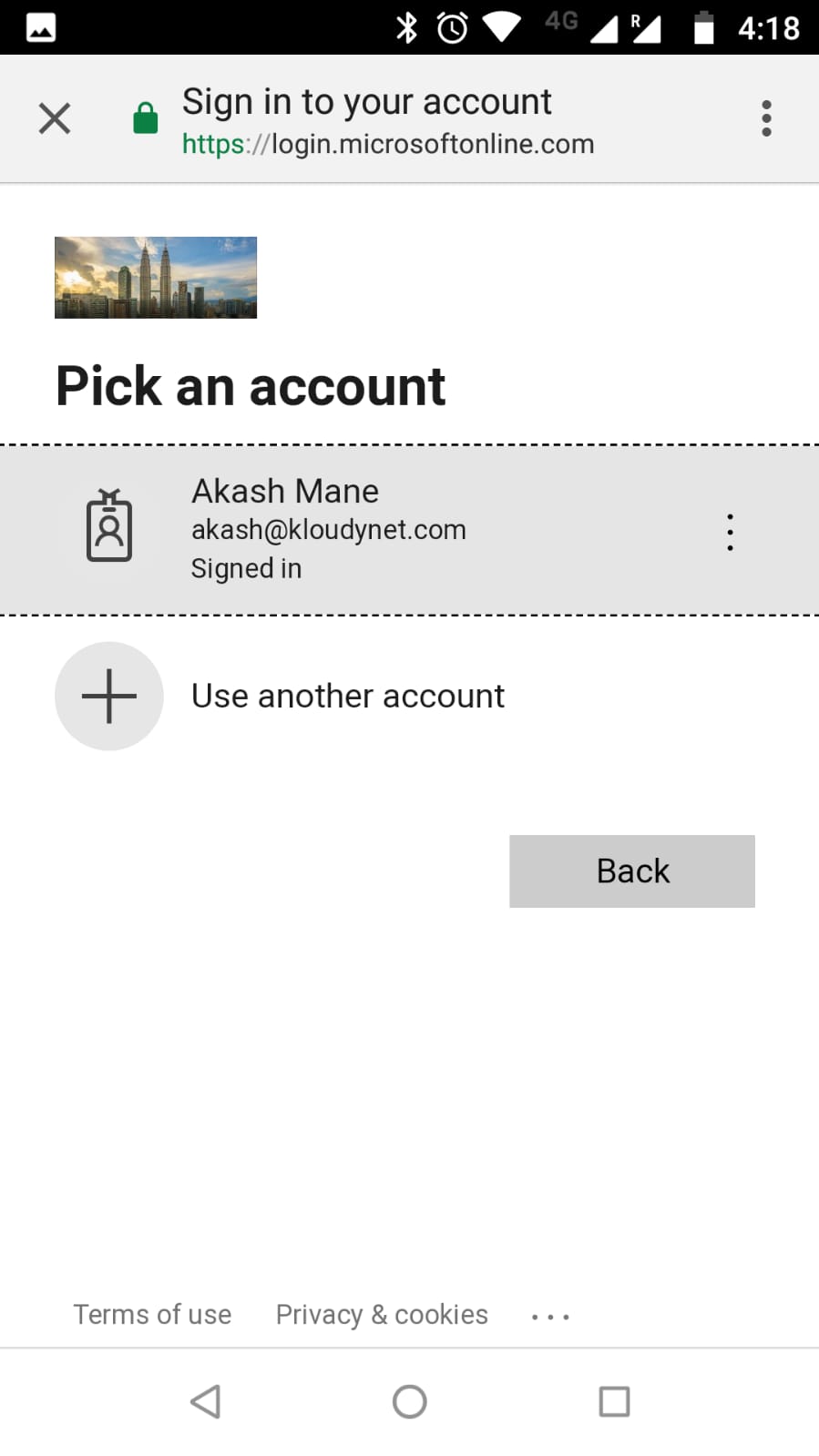
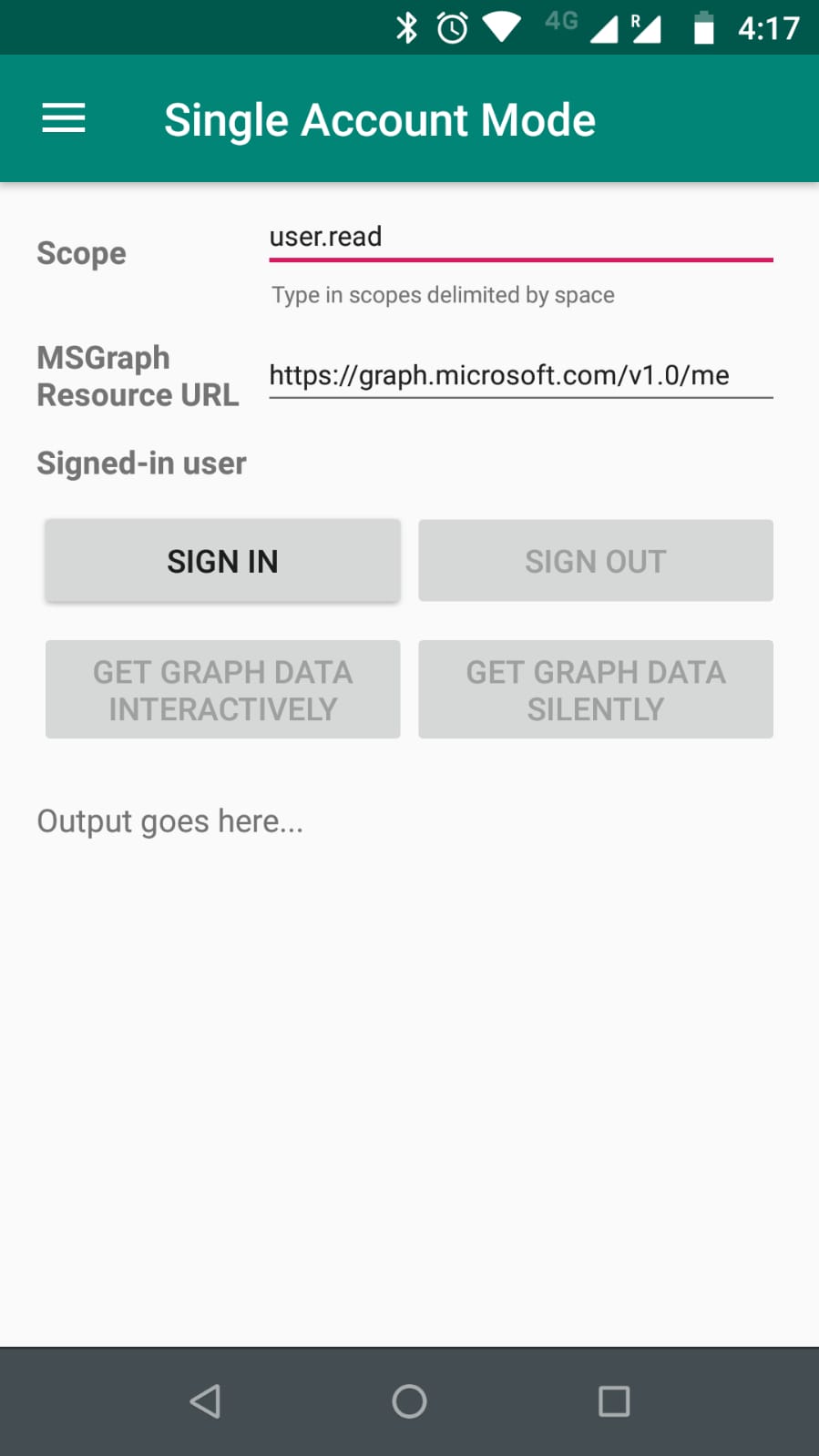
* + Adding the sign-out logic inside your existing sign out button logic

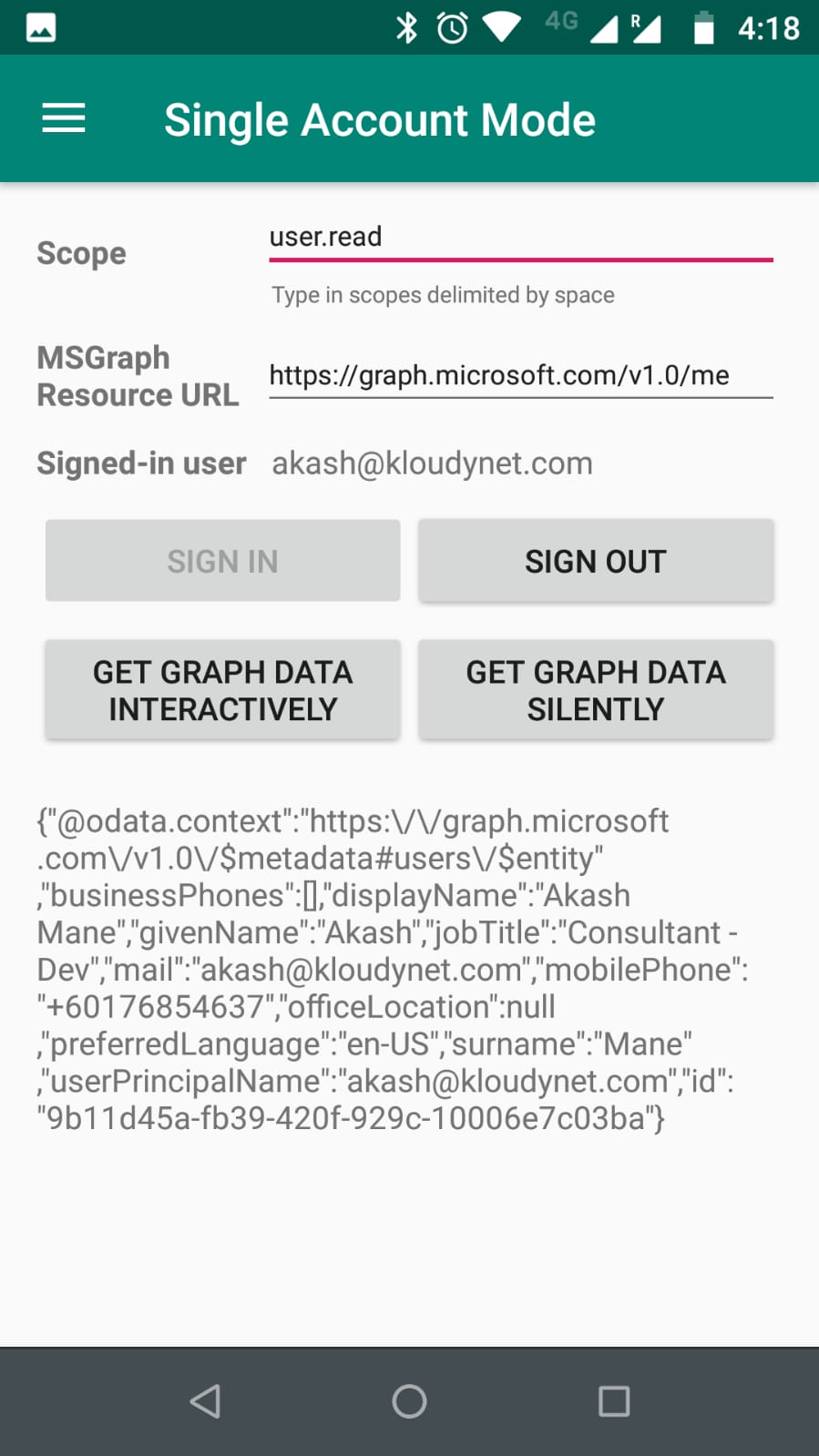
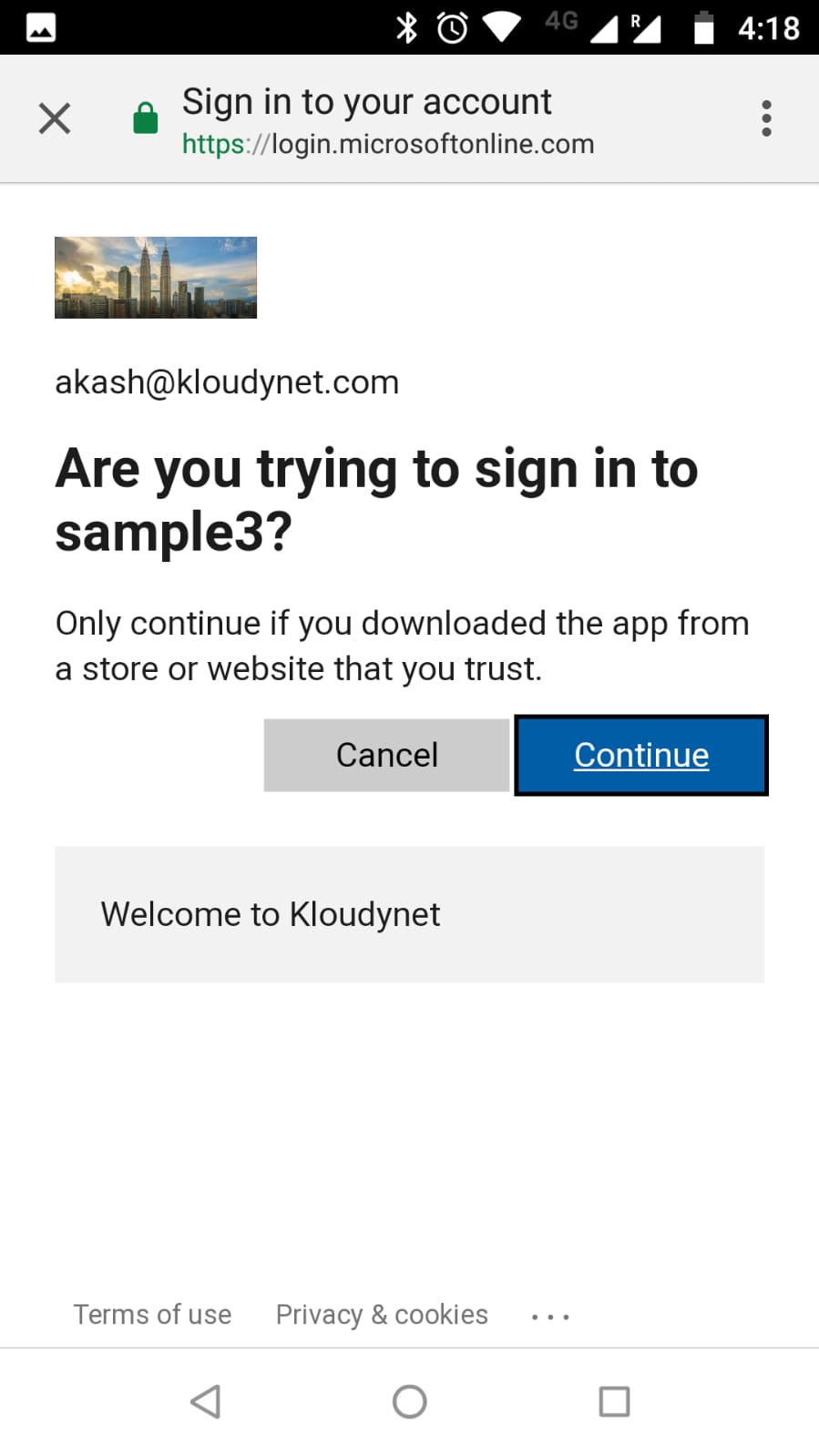
**signOutButton**.setOnClickListener(**new** View.OnClickListener() {  
 **public void** onClick(View v) {  
 **if** (**mSingleAccountApp** == **null**) {  
 **return**;  
 }  
  
 */\*\*  
 \* Removes the signed-in account and cached tokens from this app (or device, if the device is in shared mode).  
 \*/* **mSingleAccountApp**.signOut(**new** ISingleAccountPublicClientApplication.SignOutCallback() {  
 @Override  
 **public void** onSignOut() {  
 updateUI(**null**);  
 performOperationOnSignOut();  
 }  
  
 @Override  
 **public void** onError(@NonNull MsalException exception) {  
 displayError(exception);  
 }  
 });  
 }  
});

# Run the Application

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

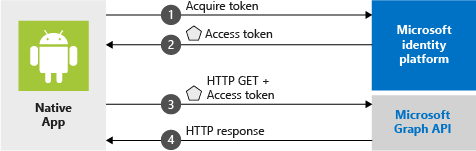
Click the sign-in link on the homepage of the application to sign in. On the Azure AD sign-in page, enter the name and password of a user account that is in your Azure AD tenant.





# 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/dotnet/framework/security/how-to-build-claims-aware-aspnet-web-forms-app-using-wif>

<https://github.com/AzureAD/azure-activedirectory-powershell-tokenkey>