

Microsoft Power BI

Integrating Azure AD B2C with App-Owns-Data Embedding

Ted Pattison

Principal Program Manager

Power BI Customer Advisory Team (PBICAT)

Agenda

- Understanding Azure AD B2C Architecture
 - Building and Testing User Flows
 - Acquiring B2C tokens using msal-browser.js
 - Managing B2C User Accounts using Microsoft Graph API
 - Developing Custom Policies using Visual Studio Code
 - Using Azure functions to implement and deploy API Connectors

What is Azure AD B2C?

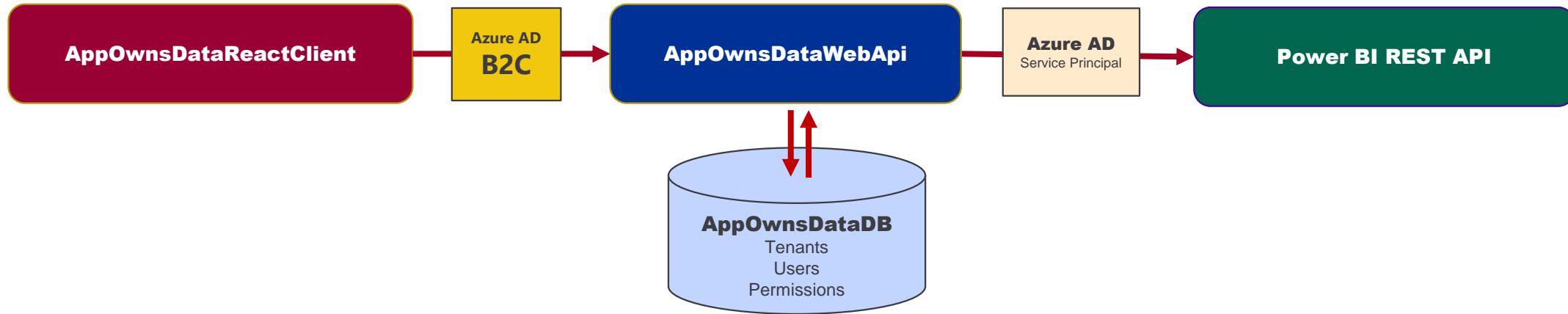
- Identity Provider for the Internet
 - Reach any potential customer on the Internet who has a valid email address
 - Consumer users can signup or sign in using social accounts or enterprise accounts
 - Consumer users can signup by creating local account
 - Azure AD B2C handles self-service features for email verification and password reset
- Azure AD B2C is a **Customer Identity and Access Management (CIAM)** solution
 - Azure AD B2C provides consumer-centric authorization and identity as a service
 - Azure AD B2C competes against other CIAM solution including **OneLogin**, **Auth0**, **Ping** and **Okta**

Comparing Azure AD B2B versus Azure AD B2C

- Azure AD B2C is a separate service from Azure AD
 - Designed to support customer-facing applications
 - Supports federating user accounts from external identity providers
 - Supports creating local user account for any human with email address
 - Provides self-service features for signup and sign in
 - Provides self-service features for profile editing and password reset
 - Allows for white labeling user signup/sign in experience
- B2C consumer accounts **cannot** access Azure resources
 - AAD B2C only provides authentication support to establish user identity
 - AAD B2C does not provide features for permissions management or authorization

Authenticating Users for App-Owns-Data Embedding

- Let's walk through a typical App-Owns-Data embedding solution

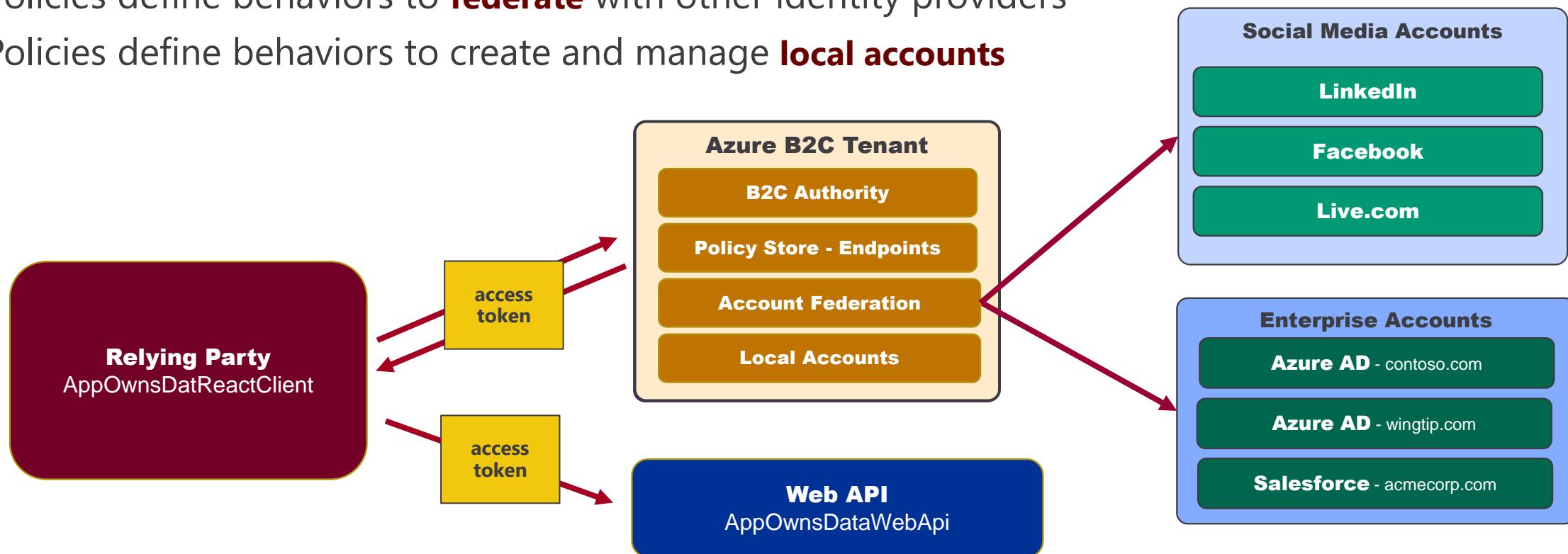


- Azure AD B2C fits in perfectly in an App-Owns-Data embedding solution
 - App-Owns-Data solution provides authorization layer (but it needs help with authentication)
 - Azure AD B2C provides authentication and identity management (but no help with authorization)



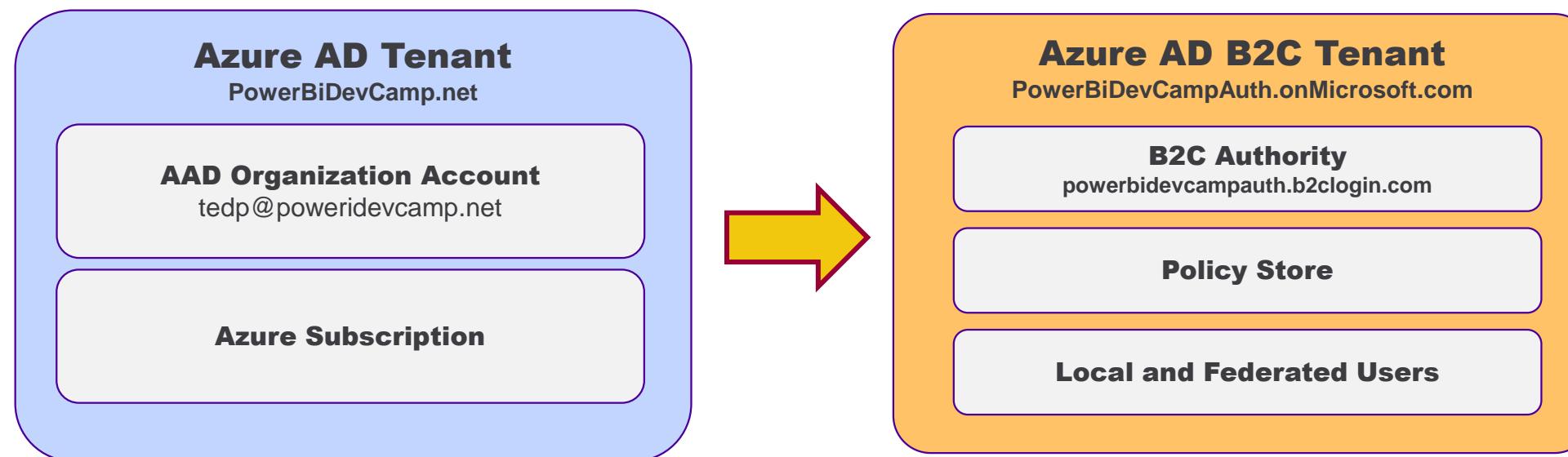
Understanding Azure AD B2C Architecture

- Azure AD B2C plays role of **identity provider**
 - B2C tenant provides **access tokens** to client applications (i.e. relying parties)
 - B2C tenant contains its own **Authority** which exposes public endpoints for user authentication
 - Each public endpoint has behavior defined by **policy** loaded from **policy store**
 - Policies define behaviors to **federate** with other identity providers
 - Policies define behaviors to create and manage **local accounts**



Creating an Azure AD B2C New Tenant

- Requirements for creating a new Azure AD B2C tenant
 - Azure AD organizational account
 - Azure subscription



Creating an Azure AD B2C Tenant in the Azure Portal

The image shows two screenshots illustrating the process of creating an Azure AD B2C tenant.

Left Screenshot: Microsoft Azure Marketplace

- The top navigation bar shows "Microsoft Azure" and a search bar with "Search resources, services, and docs (G+/)".
- The breadcrumb path is "Home > Resource groups > app-owns-data-smarter-kit > Marketplace".
- A search bar on the left contains the text "b2c".
- The results list shows one item: "Managed Azure AD B2C" by Amesto Fortytwo AS.
- Below the results, there are "Create" buttons.

Right Screenshot: Create a tenant - Configuration

- The title is "Create a tenant" with tabs for "Basics", "Configuration" (which is selected), and "Review + create".
- The sub-section is "Directory details" with the sub-instruction "Configure your new directory".
- Fields include:
 - "Organization name *": "Power BI Dev Camp Auth" (with a green checkmark)
 - "Initial domain name *": "powerbidevcampaauth" (highlighted with a yellow arrow) followed by "powerbidevcampaauth.onmicrosoft.com"
 - "Country/Region": "United States" (with a dropdown arrow)
 - A note below states: "Datacenter location is based on the country/region selected above. Azure Active Directory B2C service is available worldwide."

Demo 1 – Getting Around in Azure AD B2C Tenant Portal

The screenshot shows the Azure AD B2C Tenant Portal interface. The top navigation bar includes the tenant name "powerbidevcampauth.onmicrosoft.com", a search bar, and links for "Troubleshoot" and "Got feedback?". The left sidebar has a "Manage" section with links for "App registrations", "Applications (Legacy)", "Identity providers", "API connectors", "Company branding", "User attributes", "Users", "Roles and administrators", "Policies", "User flows", and "Identity Experience Framework". The main content area displays the "Essentials" section with the following details:

| | | | | | |
|-----------------|---|--|---------------------|---|--|
| Domain name | : | powerbidevcampauth.onmicrosoft.com | Tenant type | : | Production-scale tenant |
| Billable Units | : | Monthly Active Users | Subscription status | : | Registered |
| Subscription ID | : | 411fb15e-ad5e-442d-8969-332559b43955 | Resource name | : | powerbidevcampauth.onmicrosoft.com |

The center of the page features a "Welcome to Azure Active Directory B2C" message and three numbered steps:

- 1 Register an application**

The application registration is used to secure your directory by allowing only your applications to make requests and to make sure your users are sent to a trusted place after signing in. [Get started](#)
- 2 Add identity provider(s)**

Identity providers are the different types of accounts your users can use to sign into your application. [Get started](#)
- 3 Create a user flow**

User flows define the experience for your users signing up and signing into your application. [Get started](#)

At the bottom, there is a "Provide feedback" link.

If Success Is Important, Embrace the Azure AD B2C Docs

- <https://learn.microsoft.com/en-us/azure/active-directory-b2c>

The screenshot shows a Microsoft Learn page for Azure AD B2C Documentation. The left sidebar contains a navigation tree with categories like Overview, Quickstarts, Tutorials, Samples, Concepts, How-to guides, Authenticate and authorize, Register and configure apps, Create a user flow or custom policy, Sign-up or sign-in policy, Sign-in policy, and Password reset policies. The 'Sign-up or sign-in policy' item is highlighted. The main content area has a breadcrumb trail: Learn / Azure / Active Directory / B2C /. The title is 'Set up a sign-up and sign-in flow in Azure Active Directory B2C'. Below the title, it says 'Article • 10/21/2021 • 3 minutes to read • 7 contributors' and there is a 'Feedback' button. A 'Choose a policy type' section has 'User flow' and 'Custom policy' buttons, with 'Custom policy' being selected. A note below states: 'Before you begin, use the Choose a policy type selector to choose the type of policy you're setting up. Azure Active Directory B2C offers two methods to define how users interact with your applications: through predefined user flows or through fully configurable custom policies. The steps required in this article are different for each method.' The main heading 'Sign-up and sign-in flow' is at the bottom.

Agenda

- ✓ Understanding Azure AD B2C Architecture
- Building and Testing User Flows
 - Acquiring B2C tokens using msal-browser.js
 - Managing B2C User Accounts using Microsoft Graph API
 - Developing Custom Policies using Visual Studio Code
 - Using Azure functions to implement and deploy API Connectors

Choosing Between User Flows and Custom Policies

- There are two ways to create a B2C policy
 - **User flows** – UI-based experience for creation and configuration
 - **Custom policies** – Developers work with XML files and Identity Experience Framework (IEF)
- Reasons to choose user flows over custom policies
 - **Much easier to use** – custom policies have steep learning curve
 - **Does not require deep knowledge** about claims-based security and access token generation
 - **No code required** – everything configured in Azure AD B2C Tenant portal
- Reasons to choose custom policies over user flows
 - **More control of what claims go in token** – for example, ensure all users have **email** claim
 - **Federation with enterprise accounts** – user flows only support federating social media accounts
 - **Merging accounts during signup** - with the same email address

User flow versus Custom Policy Experience for Creators

User Flow Creators Experience

Custom Policy Development Experience

The screenshot shows the Azure AD B2C Policy Explorer interface. On the left, the Explorer sidebar lists several sections: .vscode, PasswordReset.xml, ProfileEdit.xml, SignUpOrSignIn.xml, TrustFrameworkBase.xml, TrustFrameworkExtensions.xml (which is currently selected), and TrustFrameworkLocalization.xml. Below these are sections for OUTLINE, TIMELINE, and AZURE AD B2C POLICY EXPLORER, which contains links to User Journeys, Claims Providers, Technical Profiles, Claim Types, and Claims Transformations. The main area displays the XML content of TrustFrameworkExtensions.xml. The XML code includes sections for UserJourneys, OrchestrationSteps, and ClaimsProviderSelections, with various claims exchanges and orchestration steps defined.

```
<UserJourneys>
  <UserJourney Id="CustomSignUpSignIn">
    <OrchestrationSteps>
      <OrchestrationStep Order="1" Type="ClaimsProviderSelections">
        <ClaimsProviderSelections>
          <ClaimsProviderSelection Id="LocalAccountProvider" Order="1"/>
        </ClaimsProviderSelections>
        <ClaimsExchanges>
          <ClaimsExchange Id="LocalAccountLogin" Order="1" Type="Logon">
            <Preconditions>
              <Precondition Type="ClaimIsPresent" Value="objectId" Action="SkipThisOrchestrationStep" />
            </Preconditions>
            <ClaimsExchanges>
              <ClaimsExchange Id="SignedInUser" Order="1" Type="Logon">
                <Preconditions>
                  <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
                </Preconditions>
                <ClaimsExchanges>
                  <ClaimsExchange Id="LinkedInLogon" Order="1" Type="Logon">
                    <Preconditions>
                      <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
                    </Preconditions>
                    <Actions>
                      <Action>SkinThisOrchestrationStep</Action>
                    </Actions>
                  </ClaimsExchange>
                </ClaimsExchanges>
              </ClaimsExchange>
            </ClaimsExchanges>
          </ClaimsExchange>
        </ClaimsExchanges>
      </OrchestrationStep>
      <!-- Check if the user has selected LinkedIn -->
      <OrchestrationStep Order="2" Type="ClaimsProviderSelections">
        <Preconditions>
          <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="SignedInUser" Order="1" Type="Logon">
            <Preconditions>
              <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
            </Preconditions>
            <ClaimsExchanges>
              <ClaimsExchange Id="LinkedInLogon" Order="1" Type="Logon">
                <Preconditions>
                  <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
                </Preconditions>
                <Actions>
                  <Action>SkinThisOrchestrationStep</Action>
                </Actions>
              </ClaimsExchange>
            </ClaimsExchanges>
          </ClaimsExchange>
        </ClaimsExchanges>
      </OrchestrationStep>
      <!-- Extra step for LinkedIn to skip the local account -->
      <OrchestrationStep Order="3" Type="ClaimsProviderSelections">
        <Preconditions>
          <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="LinkedInLogon" Order="1" Type="Logon">
            <Preconditions>
              <Precondition Type="ClaimIsPresent" Value="identityProvider" Action="SkipThisOrchestrationStep" />
            </Preconditions>
            <Actions>
              <Action>SkinThisOrchestrationStep</Action>
            </Actions>
          </ClaimsExchange>
        </ClaimsExchanges>
      </OrchestrationStep>
    </OrchestrationSteps>
  </UserJourney>
</UserJourneys>
```

Working with User Flows in the Azure AD B2C Portal

The screenshot shows the Microsoft Azure portal interface for managing User Flows in the Azure AD B2C service. A yellow arrow points from the 'User flows' link in the left sidebar to the 'New user flow' button at the top of the main content area. Another yellow arrow points from the 'User flows' link in the left sidebar to the 'User flows' section header in the main content area.

Microsoft Azure Search resources, services, and docs (G+)

Home > Azure AD B2C

Azure AD B2C | User flows

powerbidevcampauth.onmicrosoft.com

Search New user flow Got feedback?

User flow name

Search using user flow name

| Name | Type | Mfa |
|--------------------|-----------------------------------|-------------|
| B2C_1_HelloWorld | Sign up and sign in (Recommended) | Off |
| B2C_1_edit_profile | Profile editing (Recommended) | Off |
| B2C_1_reset | Password reset (Recommended) | Off |
| B2C_1_signin | Sign in (Recommended) | Off |
| B2C_1_signup | Sign up (Recommended) | Conditional |
| B2C_1_susi | Sign up and sign in (Recommended) | Off |

Manage

- App registrations
- Applications (Legacy)
- Identity providers
- API connectors
- Company branding
- User attributes
- Users
- Roles and administrators

Policies

- User flows
- Identity Experience Framework

Creating a New User Flow

Create a user flow ...

User flows are predefined, configured policies that you can use to set up authentication experiences for your end users. Select a user flow type to get started. [Learn more](#).

Select a user flow type

 **Sign up and sign in**
Enables a user to create an account or sign in to their account.

 **Profile editing**
Enables a user to configure their user attributes.

 **Password reset**
Enables a user to choose a new password after verifying their email.

 **Sign up**
Enables a user to create a new account.

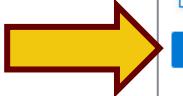
 **Sign in**
Enables a user to sign in to their account.

 **Sign in using resource owner password credentials (ROPC)**
Enables a user with a local account to sign in directly in native applications

Version

 **Recommended**
This is the generally available, next-generation user flow with latest features.

 **Standard (Legacy)**
This is the legacy user flow. Unless you have a specific business need, we don't recommend using this version.

 **Create**

User Flow Configuration Options

B2C_1_HelloWorld ...

Sign up and sign in (Recommended)

Search < Run user flow Download Delete

Overview

Settings

Properties Multifactor authentication Off

Identity providers Age gating Disabled

 Password configuration Strong

Identity providers Email signup

User attributes

Application claims Identity Provider

API connectors

After federating with an identity provider during sign-up Not selected

Before creating the user Not selected

Before including application claims in token (preview) Not selected

Customize

Page layouts Ocean Blue (Default)

Languages English

User Flow Configuration Properties

- Configure properties to control **MFA** and **conditional access**

The screenshot shows two configuration sections side-by-side.

Multifactor authentication:
Enabling multifactor authentication (MFA) requires your users to verify their identity with a second factor before allowing them into your application. [Learn more about multifactor authentication](#).
Type of method: Email
 SMS or phone call
 SMS only
 Phone call only
 Authenticator app - TOTP
MFA enforcement: Off
 Always on
 Conditional

Conditional access:
Conditional access policies evaluate signals in real time and enforce grants like MFA only when required. [Learn more about conditional access](#).
Enforce conditional access policies:

- Configure properties to control **token lifetimes** and **claims compatibility**

The screenshot shows two configuration sections side-by-side.

Token lifetime:
Access & ID token lifetimes (minutes): 60
Refresh token lifetime (days): 14
Refresh token sliding window lifetime: Bounded No expiry
Lifetime length (days): 90

Token compatibility settings:
Issuer (iss) claim: https://<domain>/ddeb8759-2319-4983-...
Subject (sub) claim: ObjectId Not supported
Claim representing user flow: tfp acr

- Configure properties to control **session behavior** and to enable **support for password reset**

The screenshot shows two configuration sections side-by-side.

Session behavior:
Web app session lifetime (minutes): 1440
Web app session timeout: Absolute Rolling
Single sign-on configuration: Tenant Application Policy Disabled
Require ID Token in logout requests: No Yes
Enable keep me signed in session:
Keep me signed in session (days): 30
Keep me signed in and session days only apply to local accounts and will override single sign-on configuration. [Learn more](#).

Password configuration:
Configure the requirements for your users' password. Use custom to define your own set of requirements. [Learn more about password complexity](#).
Self-service password reset:
Forced password reset:
Password complexity: Strong
Minimum 8 characters and maximum 64 characters in length 3 of 4 character classes - uppercase, lowercase, number, symbol. The error message for password validation is updated when a requirement is met.

Testing a User Flow for Local Account Signup

The screenshot shows a browser window for jwt.ms with a long JWT token pasted into the input field. Below the token, a note states it was issued by Azure AD B2C. The token is then decoded, showing its structure and various claims. A yellow callout box contains a note about creating an Azure AD application for jwt.ms to work properly.

jwt.ms: Welcome!

jwt.ms/#id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IlglZVhrNHH5b2pORnVtMWtsMII0djhkbE5QNC1jNTdkTzZRR1RWQndhTmsifQ..eyJleHAiOjE2NjY2Mzg5NTMsIm5iZlI6MTY2NjYzNTM1MywidmVyIjoimS4wIiwiiaXNzIjoiHR0cHM6Ly9wb3d1cmJpZGV2Y2FtcGF1dGguYjjjbG9naW4uY29tL2RkZWl4NzU5LTizMTktNDk4My04ZDUxLTc0Yzc00ThkNzRhYS92M14wLyIsInN1YiI6IjZmYzhNzU5LWM5NDQtNDcxZi11iMtc0LTQ1MmZkNjFhZDEzOSIsImF1ZCI6Ijca4ZjIyY2ZmLWE4ZWytnDBhZS1hMDhkLTi5MjcNTRjMzR1MyIsIm5vbmN1IjoiZGVmYXVsdE5vbmN1IiwiawF0IjoxNjY2NjM1MzUzLCJhdXRoX3RpbwUi0jE2NjY2MzUzNTMsInRmcCI6IkIyQ18xX0h1bGxvV29ybGQifQ.bjznvx c4F_0hcGneG3b7x1rFyekAMcp4HtzjpHdwc7FHN_Kvf2Crpev12QESjRzly6iMuxmmX0e9yTrz32UUKJuczJUCI12I0u4Kd8bIxASwUHVtEJwDist9YQa0PKYRa1SoHEJVWFj jWr7vcbj4sucXvVgA64s5XXbrgW2h4yxSHC67P9jciZ49orcSE6qulG91abKbjBzuCK8QUN43Ij4Ad2H8-d9u2T28aLdiHB700s1tQcW9W0euGi4zDtj3i5pK-vz0-jFxDiAe DPwaP4eUVn5eavspj1qEpju4EFgcAc_4Q9LbrnfVffJcr6E53AwYS9SbJXOsKYAfk9Hw

This token was issued by [Azure AD B2C](#).

Decoded Token Claims

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "kid": "X5eXk4xyojNFum1k12Ytv8d1NP4-c57d06QGTVBwaNk"  
}.{  
  "exp": 1666638953,  
  "nbf": 1666635353,  
  "ver": "1.0",  
  "iss": "https://powerbidevcampauth.b2clogin.com/ddeb8759-2319-4983-8d51-74c7498d74aa/v2.0/",  
  "sub": "6fc8a759-c944-471f-b174-452fd61ad139",  
  "aud": "78f22cff-a8ef-a08d-2927354c34e3",  
  "nonce": "defaultNonce",  
  "iat": 1666635353,  
  "auth_time": 1666635353,  
  "tfp": "B2C_1_HelloWorld"  
}.[Signature]
```

NOTE: You must create an Azure AD application for **jwt.ms** to work properly

Tenant Settings for Company Branding

Home > Azure AD B2C

Azure AD B2C | Company branding

powerbidevcampauth.onmicrosoft.com

Search « + New language Delete Refresh Columns Got feedback?

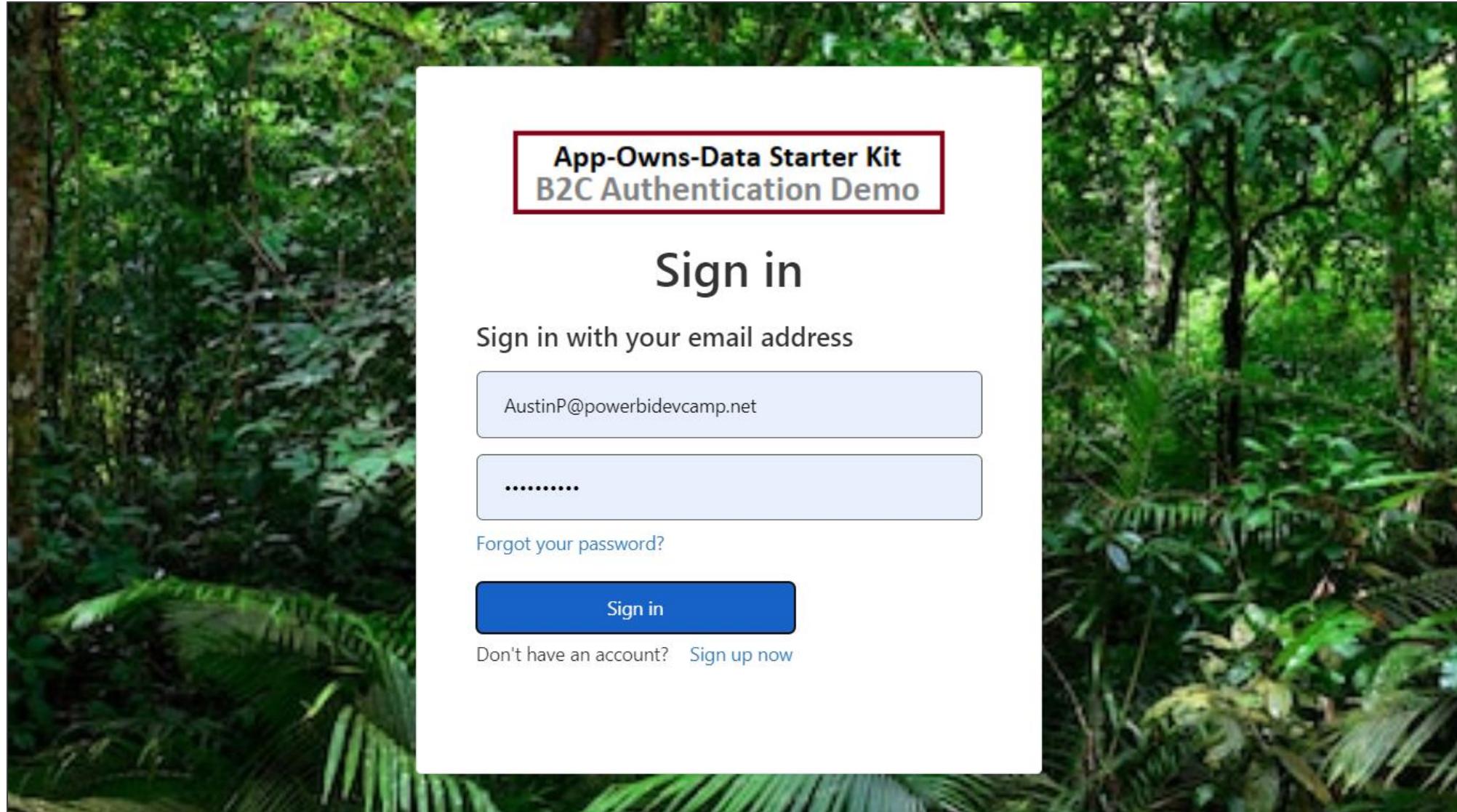
| Locale | Background image | Banner logo | Username hint | Sign-in page text |
|----------------------------------|------------------|-------------|---------------|--------------------------|
| <input type="checkbox"/> Default | ✓ | ✓ | | Hello Kind Human. Please |

Overview

Manage

- App registrations
- Applications (Legacy)
- Identity providers
- API connectors
- Company branding**
- User attributes
- Users

Company Branding Provides Instant Gratification



Identity Providers and Social Account Federation

- Users can sign in with existing accounts from popular social identity providers
 - Built-in support to federate accounts from LinkedIn, Facebook, Google, Twitter, etc.
 - Integration requires creating an “app” in the external identity provider
 - You configure support for external identity provider in Azure AD B2C portal
 - You decide which identity providers to support for each individual user flow

The screenshot shows the Microsoft Azure AD B2C Identity providers page for the B2C_1_HelloWorld user flow. The left sidebar menu includes options like Overview, Settings (Properties, Identity providers selected), User attributes, Application claims, API connectors, Customize, Page layouts, and Languages. The main content area displays information about identity providers. It starts with a note: "Identity providers are the different types of accounts your users can use to sign up or sign in to your application." Under "Local accounts", there are two radio button options: "Email signup" (selected) and "None". Under "Social identity providers", there is a table:

| Identity Provider | Name |
|---|-----------------------------|
| <input checked="" type="checkbox"/> Microsoft Account | Microsoft Personal Accounts |
| <input checked="" type="checkbox"/> Facebook | Facebook |
| <input checked="" type="checkbox"/> LinkedIn | LinkedIn |

Creating Apps To Integrate with Social Identity Providers

The screenshot shows the LinkedIn Developers platform interface for the "App-Owns-Data Starter Kit" app. The "Auth" tab is selected. The "Application credentials" section displays the Client ID (78yj2put3a818c) and Client Secret (redacted). The "OAuth 2.0 settings" section shows the Access token duration as 2 months (5184000 seconds). The "Authorized redirect URLs" section lists the URL <https://powerbidevcampauth.b2clogin.com/powerbidevcampauth.onmicrosoft.com/oauth2/authresp>. Three yellow arrows point to the Client ID, Client Secret, and Authorized redirect URL fields.

in DEVELOPERS Products Docs and tools Resources My apps

App-Owns-Data Starter Kit Client ID: 78yj2put3a818c | Created: Oct 7, 2022

Settings Auth Products Analytics Team members

Application credentials

Authentication keys

Client ID: 78yj2put3a818c

Client Secret:

OAuth 2.0 settings

Token time to live duration

Access token: 2 months (5184000 seconds)

Authorized redirect URLs for your app

<https://powerbidevcampauth.b2clogin.com/powerbidevcampauth.onmicrosoft.com/oauth2/authresp>

Understanding authentication and OAuth 2.0

In order for your applications to access LinkedIn member data and/or act on their behalf, they must be authenticated.

To make this process as easy as possible, LinkedIn relies on the industry standard OAuth 2.0 protocol for granting access.

Using [OAuth 2.0 tools](#) you can create new access tokens and inspect token details such as token validity, scopes.

[Learn more](#)

Configuring Identity Provider in AAD B2C Portal

The screenshot shows the Azure AD B2C Identity providers configuration page. On the left, under the 'Manage' section, the 'Identity providers' option is selected. In the center, a list of identity providers is shown, including Amazon, Apple, Facebook, GitHub (Preview), Google, LinkedIn, Local account, Microsoft Account, QQ (Preview), and Twitter. The LinkedIn entry is highlighted with a yellow arrow pointing to it. On the right, a modal window titled 'Configure social IDP' is open for LinkedIn. The modal contains fields for 'Origin URL' (set to <https://powerbidevcampauth.b2clogin.com>), 'Callback URL' (set to <https://powerbidevcampauth.b2clogin.com/powerbidevcampauth.onmicrosoft.com/oa...>), 'Name *' (set to LinkedIn), 'Client ID *' (set to 78), and 'Client secret *' (set to *****). A red arrow points from the LinkedIn entry in the list to the 'Name' field in the modal. Another red arrow points from the LinkedIn entry in the list to the 'Client ID' field in the modal.

Home > Azure AD B2C

Azure AD B2C | Identity providers

powerbidevcampauth.onmicrosoft.com

Search Overview

App registrations Applications (Legacy) Identity providers API connectors Company branding User attributes Users Roles and administrators Policies

New OpenID Connect provider Got feedback?

Identity provider

- Amazon
- Apple
- Facebook
- GitHub (Preview)
- Google
- LinkedIn
- Local account
- Microsoft Account
- QQ (Preview)
- Twitter

Configure social IDP

LinkedIn

Save Discard Remove

Configure your social provider. [Learn more about configuring LinkedIn as an identity provider.](#)

Origin URL

Callback URL

Name *

Client ID *

Client secret *

Signup and Sign In with Federated User Accounts

jwt.ms

Enter token below (it never leaves your browser):

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiIsImtpZCI6ig1ZvhrNhh5b2p0RnVtMwtsMll0djhkbE5QNC1jNTdkTzZRR1RwQndhTmsifQ.eyJleHAiOiE2NjY2NDExmzcsIm5iZii6MTY2NjYzNzUzNywidmVijoiMS4wiawiXNzIjoiaHR0cHM6Ly9wb3dlcm3pZGV2Y2FtcGF1dGguYjjjbG9naW4uY29tL2RkZWl4NzU5LTizHTktNDk4My04ZDUxLTc0Yzc00ThkzRhYs92Mi4wLy1sInN1YiI6Ijj0WUzYzA3LTkxYzUtNDFiNC05N2E1LTk30WVjZjdjMzzjOCIsImF1ZC16Ijc4ZjIyY2ZmLWE4ZlWytNDBhzS1hMDhkLT15MjczNTRjMzR1My1sIm5vbmn1IjoiZGvmYXVsds5vbmn1IiawiWF0iijoNxNjY2NjM3NTM3LCjhdXRox3RpbwUiOjE2NjYzMzc1MzcsIm1kcF9hY2N1c3Nfdg9rZw4i0iJBUVclenE2NuitTenhwYmIzul0xrNHJ0Q2NFSkRTMkZicVpEaG1vVjMscj1FnjNGT0d1R0p1TgdQMEppeTRlMDJ6dxZxjU0luwlBt18yUxlEeXdNRGllc1wxPeDQ2HNH1Qkp5UkVxu1YwZThTSmd4LXjTVW5mdUZQzJ5TUZMRUZXttk5WV2M3pPrmZnYWI0bktuR2wyUml1WHJUR0J5WnJlay0wazN6eDFD0Gd3bw1QR11rRGZjbXo2QkRKeEdGMUFsNxXA0MjdzUm9ycWgwZDhCdE1LUUgyZ0gxrdxRobWE3N05pbGtpam9E5kh2Q3p6UWFtew9XT3Z2YktUwlZMakMwc2tFVwZalXhTd1RFc19NNlkxMklqaGs1eXZDNuJSX2pkeThncXhzUnUzc1jpoEp2Z1hod3owb11rdmZLL2Z3tzJXTFV2akNLT295b3VLS1tYvWhZdkhSwUvqN2Z0QSiImddpmVuX25hbwUiOjJUZwqiLCJmYv1pbH1fbmFtZS16iLBhdHpc29uiawiawRwIjoiBglua2Vkal4uY29tiIwiw1bmtZ2C0QYXR0aXNvbisIm9pZC16Ijj0WUzYzA3LTkxYzUtNDFiNC05N2E1LTk30WVjZjdjMzzjOCIsIm5ld1VzZXi0OnRydWUsInRmcCI6IkIyQ18xX3N1c2kifQ.WKWiCNCQh_jmdlxX55MFQj2Nsad7xiTmNvUovtt1w8hTveqJatQ04f0DTk9210QpPOL3dSj0Sc510hVaLkI_0NGBH8waxMF0jbUj1-hsRJJ1akQ4sPWyVT_AMhMGr4v2jyp_Z55LkNNISmrZKFQnP0lyB-PaIqmN9qzOpdy_k3vIKD3cF2puqedDvo94FnbsVfXNyw-Rdsk05rIBKgUN8y5DuuSYOBfnCj003joxqwiBsjsjE50ymI4YGOtixhXVCV7XikU151Qir8uE17MT9dc_S2nnerbkQt1hDqAs5wSd2MD8hCQrHmxG_7-N4UM7K1-v8Wfngy3EqVeYqrF_w
```

This token was issued by Azure AD B2C.

Decoded Token Claims

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "X5eXk4xyojNFum1k12Ytv8d1NP4-c57d06QGTVBwaNk"
}.{
  "exp": 1666641137,
  "nbf": 1666637537,
  "ver": "1.0",
  "iss": "https://powerbidevcampauth.b2clogin.com/ddeb8759-2319-4983-8d51-74c7498d74aa/v2.0",
  "sub": "2c9e3c07-91c5-41b4-97a5-979ecf7c36c8",
  "aud": "78f22cff-a8ef-40ae-a08d-2927354c34e3",
  "nonce": "defaultNonce",
  "iat": 1666637537,
  "auth_time": 1666637537,
  "given_name": "Ted",
  "family_name": "Pattison",
  "idp": "linkedin.com",
  "name": "Ted Pattison",
  "oid": "2c9e3c07-91c5-41b4-97a5-979ecf7c36c8",
  "newUser": true,
  "tfp": "B2C_1_susi"
}.[Signature]
```

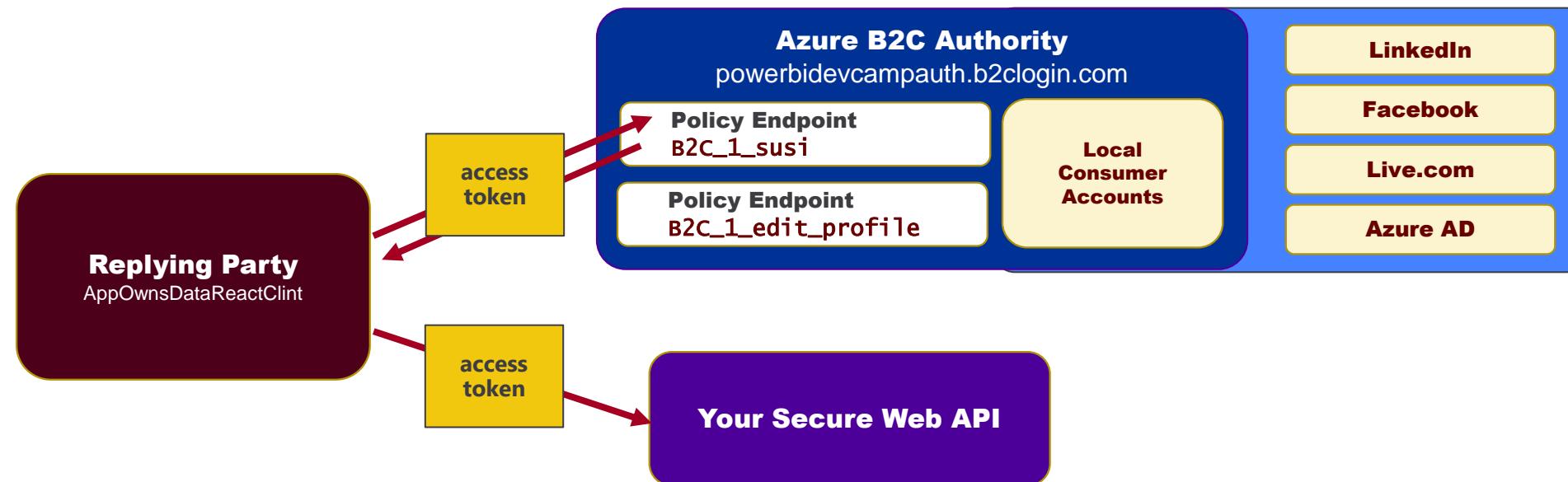


Agenda

- ✓ Understanding Azure AD B2C Architecture
- ✓ Building and Testing User Flows
- Acquiring B2C tokens using msal-browser.js
 - Managing B2C User Accounts using Microsoft Graph API
 - Developing Custom Policies using Visual Studio Code
 - Using Azure functions to implement and deploy API Connectors

Acquiring Tokens from AD B2C Authority

- Each Azure AD B2C authentication endpoint defined by a policy
 - Replying party redirects the user to policy endpoint to begin authentication flow
 - Replying party must include scope(s) in request to policy endpoint to acquire access token
 - B2C authority can optionally provide interactive experience as it executes policy
 - B2C authority returns access token if policy executes successfully
 - Relying party transmits access token in calls to Web API where Web API validates access token
 - Web API inspects access token claims to determine whether user has required access



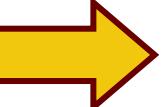
Creating an Application in the Azure AD B2C Tenant

Microsoft Azure Search resources, services, and docs (G+/-)

Home > Azure AD B2C | App registrations >

App-Owns-Data Starter Kit B2C Auth Demo

Search Delete Endpoints Preview features

Overview 

Integration assistant

Manage

Branding & properties

Authentication

Essentials

| | | | |
|-------------------------|---|-----------------------------|---|
| Display name | : App-Owns-Data Starter Kit B2C Auth Demo | Client credentials | : 0 certificate, 1 secret |
| Application (client) ID | : 81d70ab2-2492-4203-9fa0-40f5d6d36c31 | Redirect URIs | : 0 web, 5 spa, 0 public client |
| Object ID | : 5ae1cb05-d137-4eee-99c1-a62367861668 | Application ID URI | : https://powerbidevcampauth.onmicrosoft.com/app-owns-data-starter-kit-b2c-auth-demo |
| Directory (tenant) ID | : ddeb8759-2319-4983-8d51-74c7498d74aa | Managed application in I... | : App-Owns-Data Starter Kit B2C |
| Supported account types | : All users | | |

Configuring Application with SPA Authentication

The screenshot shows the Microsoft Azure portal interface for managing an app registration. The left sidebar lists several management options: Overview, Integration assistant, Branding & properties, Authentication (which is highlighted with a yellow arrow), Certificates & secrets, API permissions, Expose an API, and Owners. The main content area displays the configuration for a "Single-page application". It includes a section for "Redirect URIs" with a list of URLs:

- https://appownsdataclient.azurewebsites.net/
- https://appownsdatareactclient.azurewebsites.net/
- https://localhost
- https://jwt.ms
- http://localhost

Defining Custom Scope to Expose a Web API

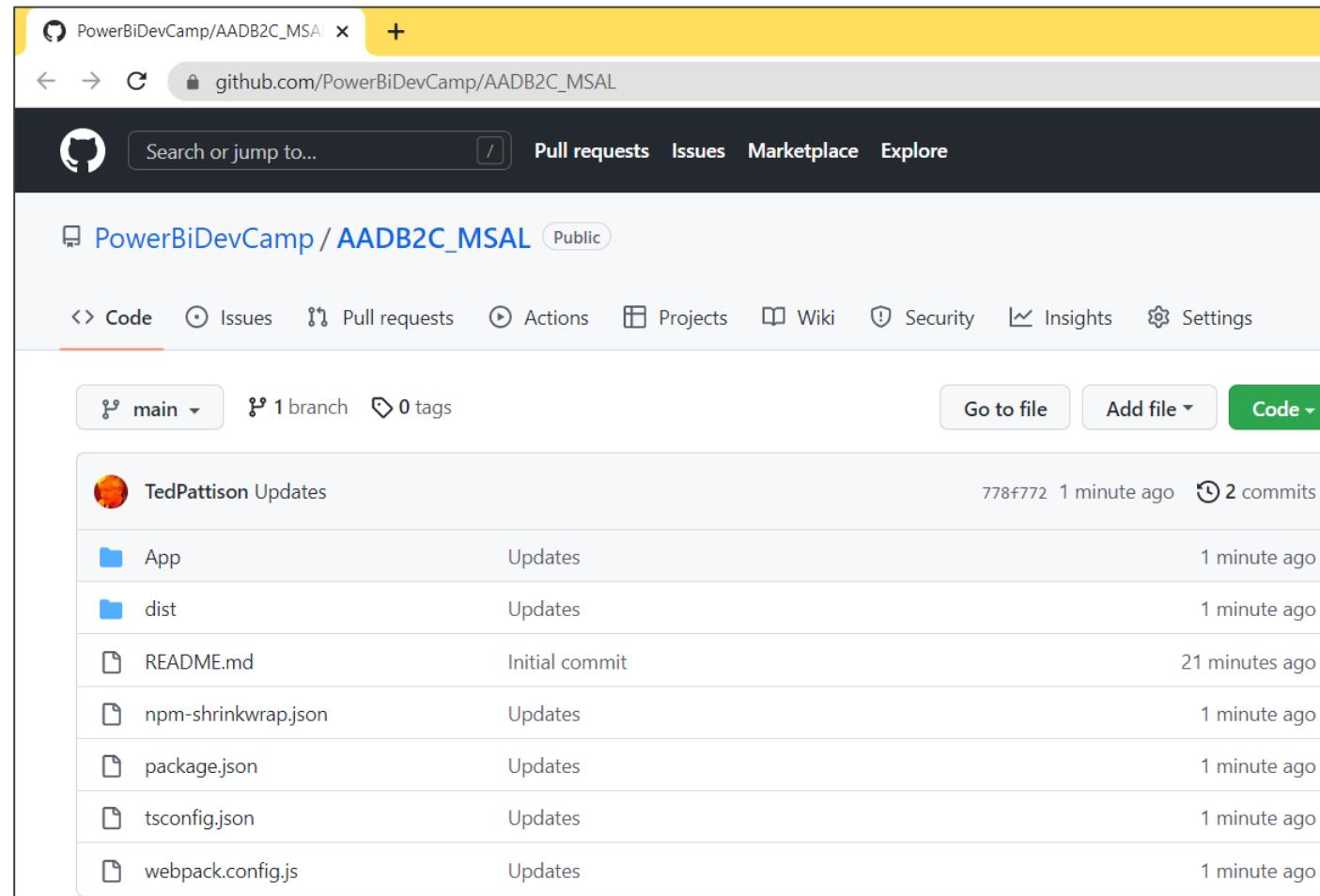
The screenshot shows the Microsoft Azure portal interface for managing an app registration. The top navigation bar includes the Microsoft Azure logo, a search bar, and various navigation icons. The current page path is "Home > Azure AD B2C | App registrations > App-Owns-Data Starter Kit B2C Auth Demo". The main title is "App-Owns-Data Starter Kit B2C Auth Demo | Expose an API". On the left, a sidebar titled "Manage" lists several options: Overview, Integration assistant, Branding & properties, Authentication, Certificates & secrets, API permissions, Expose an API (which is highlighted with a yellow arrow), and Owners. The "Expose an API" section contains the following details:

- Application ID URI:** https://powerbidevcampauth.onmicrosoft.com/81d70ab2-2492-4203-9fa0-40f5d6d36c31
- Scopes defined by this API:** A description stating "Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these."
- Add a scope:** A button to add new scopes.
- Scopes table:** A table listing a single scope entry:

| | Who can consent | Admin consent disp... | State |
|---|-----------------|-------------------------|---------|
| https://powerbidevcampauth.onmicrosoft.com/81d70ab2-2492-4203-9fa0-40f5d6d36c31/Reports.Embed | Admins only | Embed reports in App... | Enabled |

AADB2C_MSAL Sample Application in GitHub

- AADB2C_MSAL demonstrates best practice of authenticating with **msal-browser.js**
 - Download sample code from https://github.com/PowerBiDevCamp/AADB2C_MSAL



Updating AuthConfig.ts for Your Environment

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** AuthConfig.ts - AADB2C_MSAL - Visual Studio Code.
- Explorer Bar:** Shows the project structure under AADB2C_MSAL / App, including components, models, services, App.tsx, and AuthConfig.ts (the current file being edited).
- Code Editor:** The content of AuthConfig.ts is displayed, containing TypeScript code for MSAL configuration. Three specific lines are highlighted with yellow arrows:
 - Line 2: `// Replace this value with the name of your Azure AD B2C tenant`
 - Line 7: `// Replace value for client ID for SPA application you created in Azure AD B2C tenant`
 - Line 8: `// make sure delegated permission scope is correct`
- Status Bar:** Shows the line number 4.

Testing Out the AADB2_MSAL Application

A screenshot of the Azure AD B2C Auth Demo application interface. The top navigation bar includes the title "Azure AD B2C Auth Demo" and a "IDENTITIES" tab, which is highlighted with a yellow arrow. On the right side of the top bar is a user profile for "AUSTIN POWERS". The main content area is titled "Identities" and displays a table of account information. The table has columns: Local Account ID, User Name, Policy Name, and Active. There are two rows of data:

| Local Account ID | User Name | Policy Name | Active |
|--------------------------------------|--|--------------------|--------|
| b023cbcc-c524-4d5d-97e4-ea9914d38fb9 | Austin Powers | B2C_1_susi | true |
| b023cbcc-c524-4d5d-97e4-ea9914d38fb9 | Austin Powers (International Man of Mystery) | B2C_1_edit_profile | false |

In the bottom right corner of the table, there is a small "Fiddler Classic" watermark.

Selecting Signup/Signin Account As the Active Account

```
const selectAccount = () => {
  console.log("Executing selectAccount");
  const currentAccounts = instance.getAllAccounts();

  // if there are no accounts in cache, do nothing
  if (currentAccounts.length < 1) {
    return;
  }

  // if there is 1 account, set that account as the active account
  if (currentAccounts.length === 1) {
    instance.setActiveAccount(currentAccounts[0]);
    return;
  }

  // if there are multiple accounts, ensure the account with the signup/signin policy is active
  if (currentAccounts.length > 1) {
    let currentAccountPolicy = (instance.getActiveAccount()?.idTokenClaims as any)?.tfp.toUpperCase();
    let requiredAccountPolicy = b2cPolicies.names.signIn.toUpperCase();
    if (currentAccountPolicy !== requiredAccountPolicy) {
      console.log("WRONG ACCOUNT ACTIVE! - setting active account back to " + b2cPolicies.names.signIn);
      const filteredAccounts = currentAccounts.filter(account =>
        account.homeAccountId.toUpperCase().includes(b2cPolicies.names.signIn.toUpperCase()) &&
        account.idTokenClaims.iss.toUpperCase().includes(b2cPolicies.authorityDomain.toUpperCase()) &&
        account.idTokenClaims.aud === msalConfig.auth.clientId
      );

      // determine if search found signup/signin policy account
      if (filteredAccounts.length > 0) {
        // if so, set signup/signin policy account as active account
        console.log("Setting SignUp/Signin policy as active account", filteredAccounts[0])
        instance.setActiveAccount(filteredAccounts[0]);
      }
    }
  }
}
```

| Local Account ID | User Name | Policy Name | Active |
|--------------------------------------|--|--------------------|--------|
| b023cbcc-c524-4d5d-97e4-ea9914d38fb9 | Austin Powers | B2C_1_susi | true |
| b023cbcc-c524-4d5d-97e4-ea9914d38fb9 | Austin Powers (International Man of Mystery) | B2C_1_edit_profile | false |

Inspecting the B2C Security Tokens using Fiddler

The screenshot shows the Fiddler Classic interface with the following details:

- Session List:** Shows multiple requests from "powerbidevcampauth.b2login.com". Request #62 is selected, showing the full URL: /powerbidevcampauth.onmicrosoft.com/b2c_1_susi/oauth2/v2.0/token.
- Log Tab:** Displays various headers and parameters used in the request, such as x-client-current-telemetry, code_verifier, grant_type, client_info, client-request-id, and X-AnchorMailbox.
- JSON Tab:** Shows the decoded JSON structure of the access token. Key fields include:
 - access_token: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilg1ZVhrNhh5b2pORnVtMWtsMII0djhkbE5QNC1jNTdkTzZRR1RWQndhTmsifQ,eyJpc3...
 - client_info: eyJ1aWQiOiI2ZmM4YTc1OS1jOTQ0LTQ3MWYtYjE3NC00NTJmZDYxYWQzMzktyjJXzFfc3VzaSIsInV0aWQiOjkZGVlODc1OS0yMzE5LTQ5...
 - expires_in: 3600
 - expires_on: 1666644377
 - id_token: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilg1ZVhrNhh5b2pORnVtMWtsMII0djhkbE5QNC1jNTdkTzZRR1RWQndhTmsifQ,eyJleHaiOjE...
 - not_before: 1666640777
 - refresh_token: eyJraWQiJjcGltY29yZV8wOTI1MjAxNSIsInZlciI6IjEuMCIsInppcCI6IkRlZmxhdGUiLCJzZXIiOlxLjAifQ,..Dnnqz31t82j_Q-u.ZLoV3VsLZ...
 - refresh_token_expires_in: 1209600
 - resource: 81d70ab2-2492-4203-9fa0-40f5d6d36c31
 - scope: https://powerbidevcampauth.onmicrosoft.com/81d70ab2-2492-4203-9fa0-40f5d6d36c31/Reports.Embed
 - token_type: Bearer

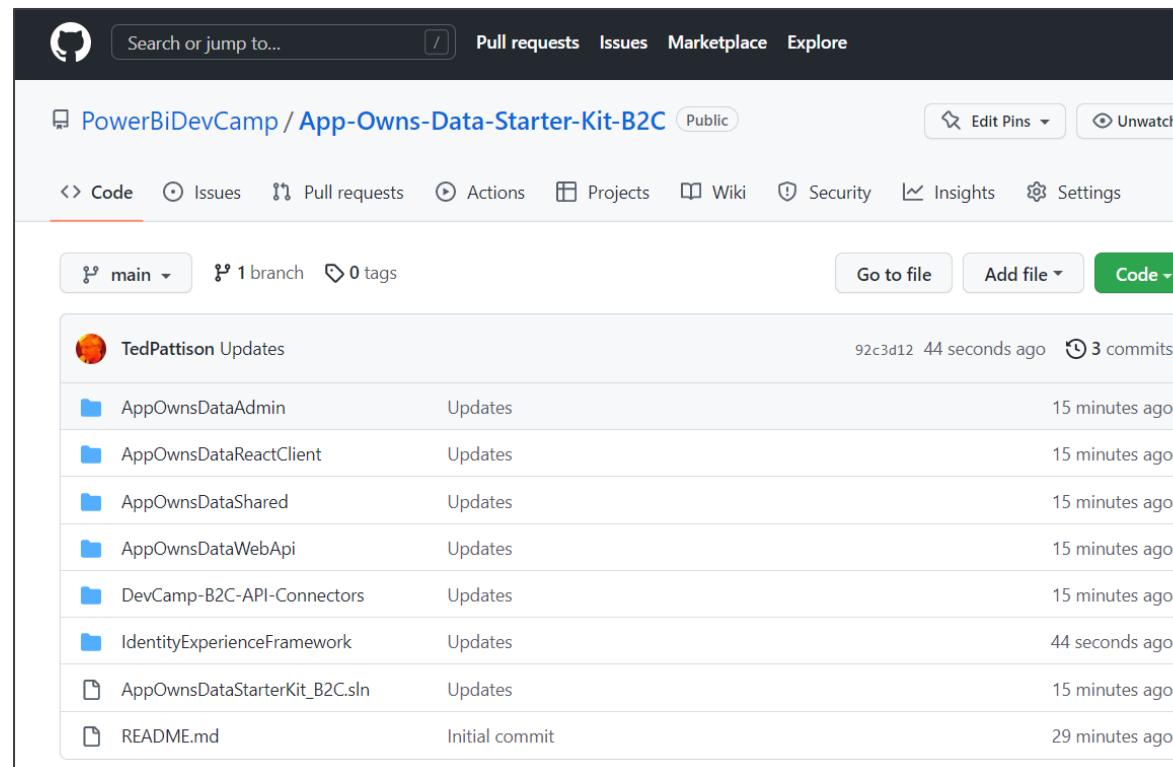
Agenda

- ✓ Understanding Azure AD B2C Architecture
- ✓ Building and Testing User Flows
- ✓ Acquiring B2C tokens using msal-browser.js
- Managing B2C User Accounts using Microsoft Graph API
 - Developing Custom Policies using Visual Studio Code
 - Using Azure functions to implement and deploy API Connectors

AppOwnsDataStarterKit_B2C

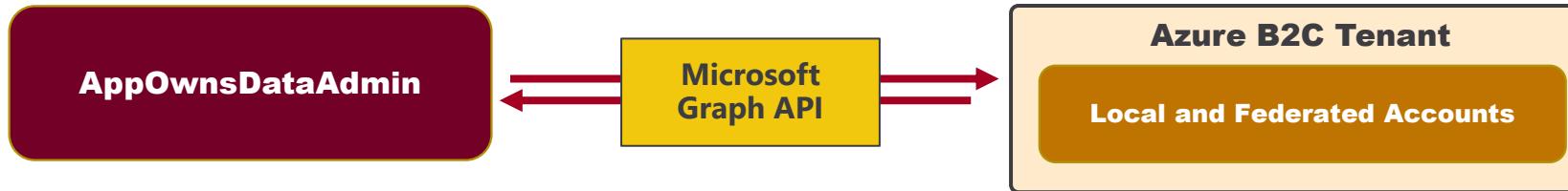
a customized Version of the App-Owns-Data Starter Kit

- Available for inspection or download in GitHub repository
 - https://github.com/PowerBiDevCamp/AppOwnsDataStarterKit_B2C
- Try out the **live demo** to experience Azure AD B2C signup and sign in process
 - <https://appownsdatareactclient.azurewebsites.net/>



Managing Consumers Accounts with Microsoft Graph API

- Custom application can create/update/delete consumer accounts



- Microsoft Graph API makes it possible to manage consumer accounts in B2C tenant

The screenshot shows a web-based application interface for managing B2C User Accounts. The top navigation bar includes links for 'Tenants', 'Users', 'B2C User Accounts', and 'Activity Log'. On the right, there are 'Welcome Ted Pattison' and 'Sign out' options. The main content area is titled 'B2C User Accounts' and displays a table of user data:

| Email | Display Name | Identity Provider | Created | Enabled | View | Delete |
|----------------------------|------------------|---------------------------------|-----------------------|---------|------|--------|
| ted@tedpattison.net | Tedrick Pattison | Facebook | 10/27/2022 3:14:02 AM | False | | |
| austinp@powerbidevcamp.net | Austin Powers | Local B2C Account | 10/27/2022 3:17:11 AM | True | | |
| edpattis@microsoft.com | Edward Pattison | LinkedIn | 10/27/2022 3:15:23 AM | False | | |
| JasonB@powerbidevcamp.net | Jason Bourne | AAD Tenant (powerbidevcamp.net) | 10/27/2022 3:18:41 AM | False | | |
| maxwelbsmart@outlook.com | Maxwell Smart | Microsoft Personal (live.com) | 10/27/2022 3:17:59 AM | False | | |

Creating Administrative Application in B2C Tenant

- Application requires application permission **User.ReadWrite.All**

The screenshot shows the Microsoft Azure portal interface for managing API permissions. The left sidebar has a 'Manage' section with several options: Overview, Integration assistant, Branding & properties, Authentication, Certificates & secrets, API permissions (which is selected and highlighted in grey), Expose an API, Owners, and Manifest. The main content area is titled 'Custom B2C Tenant Manager App | API permissions'. It shows a 'Configured permissions' section with a note about granting permissions for APIs. Below this is a table with columns: API / Permissions name, Type, Description, Admin consent required, and Status. The table lists four permissions under 'Microsoft Graph (4)': 'Application.ReadWrite.All' (Application type, Yes status, Granted for Power BI Dev Camp Auth), 'offline_access' (Delegated type, No status, Granted for Power BI Dev Camp Auth), 'openid' (Delegated type, No status, Granted for Power BI Dev Camp Auth), and 'User.ReadWrite.All' (Application type, Yes status, Granted for Power BI Dev Camp Auth). A large yellow arrow points to the 'User.ReadWrite.All' row.

| API / Permissions name | Type | Description | Admin consent required | Status |
|---------------------------|-------------|--|------------------------|------------------------------------|
| Application.ReadWrite.All | Application | Read and write all applications | Yes | Granted for Power BI Dev Camp Auth |
| offline_access | Delegated | Maintain access to data you have given ... | No | Granted for Power BI Dev Camp Auth |
| openid | Delegated | Sign users in | No | Granted for Power BI Dev Camp Auth |
| User.ReadWrite.All | Application | Read and write all users' full profiles | Yes | Granted for Power BI Dev Camp Auth |

B2CAccountsManager.cs in AppOwnsDataAdmin

```
namespace AppOwnsDataAdmin.Services {
    public class B2CAccountsManager {
        "Auth support"

        public List<User> GetUsers() {
            List<User> users = new List<User>();
            string userProperties = "id, identities, displayName, givenName, surname, userPrincipalName, creationType, otherMails, createdDateTime, accountEnabled";
            var usersResponse = graphClient.Users.Request().Select(userProperties).GetAsync().Result;
            foreach (User user in usersResponse) {
                if (user.Id != TenantAdminObjectId) {
                    user.CreationType = GetIdentityProviderName(user);
                    users.Add(user);
                }
            }
            return users;
        }

        public void DeleteUser(string userId) {
            graphClient.Users[userId].Request().DeleteAsync().Wait();
            return;
        }

        public User GetUser(string userId) {
            string userIdFilter = "id eq '" + userId + "'";
            string userProperties = "id, displayName, givenName, surname, otherMails, identities, userPrincipalName, creationType, createdDateTime, accountEnabled";
            var user = graphClient.Users.Request().Select(userProperties).Filter(userIdFilter).GetAsync().Result.Single();
            user.CreationType = GetIdentityProviderName(user);
            return user;
        }

        public User ToggleUserEnabled(string userId) {
            string userIdFilter = "id eq '" + userId + "'";
            string userProperties = "id, accountEnabled";
            var user = graphClient.Users.Request().Select(userProperties).Filter(userIdFilter).GetAsync().Result.Single();
            bool userStatus = user.AccountEnabled.Value;

            var updatedValues = new User {
                AccountEnabled = !userStatus
            };

            return this.graphClient.Users[user.Id].Request().UpdateAsync(updatedValues).Result;
        }

        public string GetIdentityProviderName(User user) ...
    }
}
```

Agenda

- ✓ Understanding Azure AD B2C Architecture
- ✓ Building and Testing User Flows
- ✓ Acquiring B2C tokens using msal-browser.js
- ✓ Managing B2C User Accounts using Microsoft Graph API
- Developing Custom Policies using Visual Studio Code
 - Using Azure functions to implement and deploy API Connectors

Developing Custom Policies

- Custom policy are developed in accordance with **Identity Experience Framework (IEF)**
 - Policies for authentication and token acquisition defined using customized XML files
 - Developer must continually edit and upload XM files to test and debug custom policies
- Why would you decide to develop custom policies?
 - They add extra control over what claims get added to token (e.g. verified email address)
 - They provide ability to integrate Azure AD tenant(s)
 - They provide ability to integrate other enterprise identity providers such as Salesforce
 - They provide a wealth of functionality not covered here!
- Custom policy development has a steep learning curve
 - Getting up to speed could take 2-3 months

Getting Started with Custom Policies

- Start by going through the steps in this article

<https://learn.microsoft.com/en-us/azure/active-directory-b2c/tutorial-create-user-flows?pivots=b2c-custom-policy>

The screenshot shows a Microsoft Azure documentation page for 'Azure AD B2C Documentation'. The left sidebar contains a navigation menu with several items under 'Tutorials': 'Overview', 'Quickstarts' (with sub-items for ASP.NET app, desktop app, and single-page app), 'Tutorials' (with sub-items for creating a tenant, registering a web application, and the current item, '3 - Create user flows and custom policies'), and 'Samples'. A red arrow points to the '3 - Create user flows and custom policies' link. The main content area has a breadcrumb trail: Learn / Azure / Active Directory / B2C. The title is 'Tutorial: Create user flows and custom policies in Azure Active Directory B2C'. Below the title, it says 'Article • 07/20/2022 • 13 minutes to read • 16 contributors' and a 'Feedback' button. A callout box titled 'Choose a policy type' contains two tabs: 'User flow' (which is highlighted with a blue background) and 'Custom policy' (which is highlighted with a yellow background). A red arrow points to the 'Custom policy' tab. Below the callout box, a note reads: 'Before you begin, use the Choose a policy type selector to choose the type of policy you're setting up. Azure Active Directory B2C offers two methods to define how users interact with your applications: through predefined user flows or through fully configurable custom policies. The steps required in this article are different for each method.'

B2C Custom Policy Starter Pack

- Developers should use XML files from starter pack as project starting point
 - <https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack>

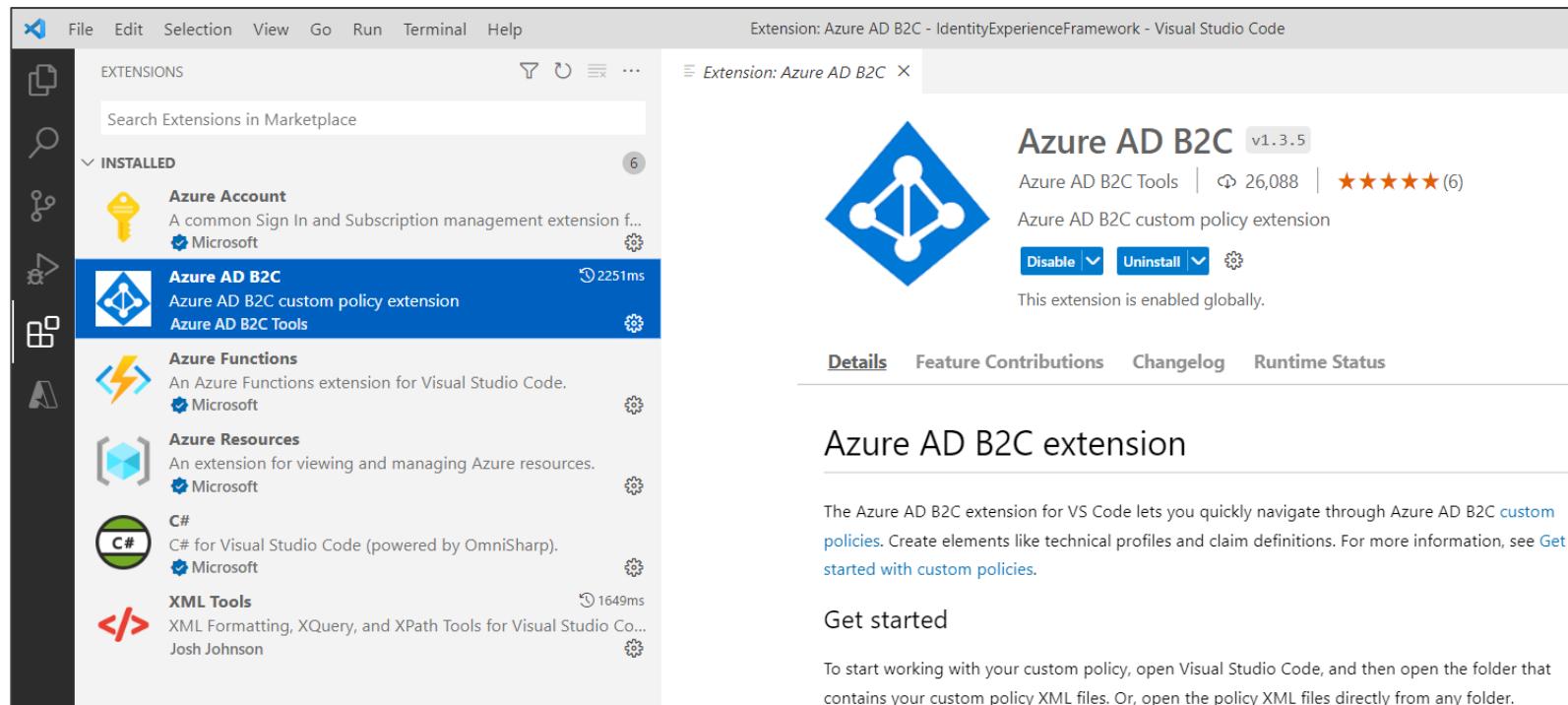
The screenshot shows a GitHub repository page for 'active-directory-b2c-custom-policy-starterpack'. The main repository page has a navigation bar with 'Code' selected, followed by 'Issues 66', 'Pull requests 3', 'Actions', 'Projects', 'Security', and 'Insights'. Below this, there are buttons for 'main' (branch), '2 branches', and '0 tags'. A commit by 'yoelhor' titled 'Email regex restriction improvement' is listed. The repository structure on the left includes 'Display Controls Starterpack', 'LocalAccounts', 'SocialAccounts', a redboxed 'SocialAndLocalAccounts' folder, and 'SocialAndLocalAccountsWithMfa'. A yellow arrow points from the 'SocialAndLocalAccounts' folder to a detailed view of its contents on the right.

SocialAndLocalAccounts /

| File | Description |
|--------------------------------|--|
| PasswordReset.xml | Update Bug Fixes, new feautures |
| ProfileEdit.xml | Update Bug Fixes, new feautures |
| SignUpOrSignin.xml | RefreshToken User Journey Added |
| TrustFrameworkBase.xml | Email regex restriction improvement |
| TrustFrameworkExtensions.xml | RefreshToken User Journey Added |
| TrustFrameworkLocalization.xml | Adding B2C_1A_TrustFrameworkLocalization policy |
| readme.md | Fix the login hint for edit profile and pwd change |

Developing Custom Policies with Visual Studio Code

- Use the Azure AD B2C extension for Visual Studio Code
 - Azure AD B2C Policy Explorer
 - IntelliSense with IEF schema
 - Automated XML file upload in VS Code for testing
 - Easy viewing of Azure AD B2C Traces from App Insights



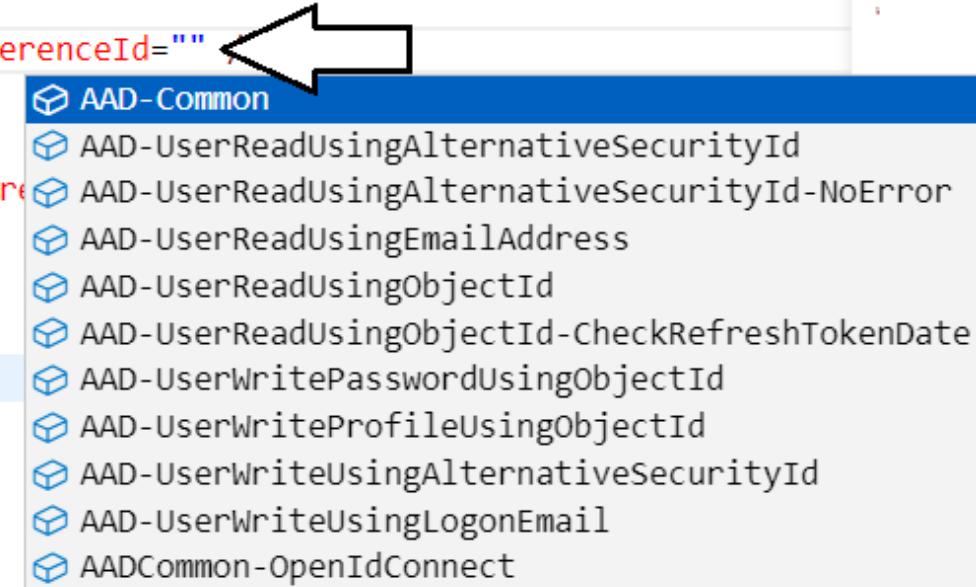
Azure AD B2C Policy Explorer

The screenshot shows a Visual Studio Code window with the title "TrustFrameworkBase.xml - IdentityExperienceFramework - Visual Studio Code". The left sidebar has icons for Explorer, Outline, Timeline, and Azure AD B2C Policy Explorer. The main area shows the XML code for a technical profile named "AAD-UserWriteProfileUsingObjectId". The code includes metadata, input claims, and persisted claims, along with comments explaining optional and required claims. The line numbers 639 through 661 are visible on the left.

```
<TechnicalProfile Id="AAD-UserWriteProfileUsingObjectId">
  <Metadata>
    <Item Key="Operation">Write</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalAlreadyExists">false</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
  </Metadata>
  <IncludeInSso>true</IncludeInSso>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
  </InputClaims>
  <PersistedClaims>
    <!-- Required claims -->
    <PersistedClaim ClaimTypeReferenceId="objectId" />
    <PersistedClaim ClaimTypeReferenceId="email" PartnerClaimType="signInName" />
    <!-- Optional claims -->
    <PersistedClaim ClaimTypeReferenceId="displayName" />
    <PersistedClaim ClaimTypeReferenceId="givenName" />
    <PersistedClaim ClaimTypeReferenceId="surname" />
  </PersistedClaims>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
<!-- The following technical profile is used to read data after user authenticates. -->
<TechnicalProfile Id="AAD-UserReadUsingObjectId">
```

IntelliSense When Configuring Profile References

```
<OrchestrationStep Order="5" Type="ClaimsExchange">
    <ClaimsExchanges>
        <ClaimsExchange Id="B2CUserProfileUpdateExchange" TechnicalProfileReferenceId="" />
    </ClaimsExchanges>
</OrchestrationStep>
<OrchestrationStep Order="6" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="CustomPasswordReset" ...>
    <EntitlementDefinition ReferenceId="DefaultWeb" />
    <Journey>
        <Journey Step="1" Order="1" Type="Start" ...>
            <ProfileReference Id="CustomPasswordReset" ...>
                <Policy>
                    ...
                </Policy>
            </ProfileReference>
        </Journey>
    </Journey>
</OrchestrationStep>
<OrchestrationStep Order="7" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="CustomPasswordReset" ...>
    <EntitlementDefinition ReferenceId="DefaultWeb" />
    <Journey>
        <Journey Step="2" Order="1" Type="Start" ...>
            <ProfileReference Id="CustomPasswordReset" ...>
                <Policy>
                    ...
                </Policy>
            </ProfileReference>
        </Journey>
    </Journey>
</OrchestrationStep>
```



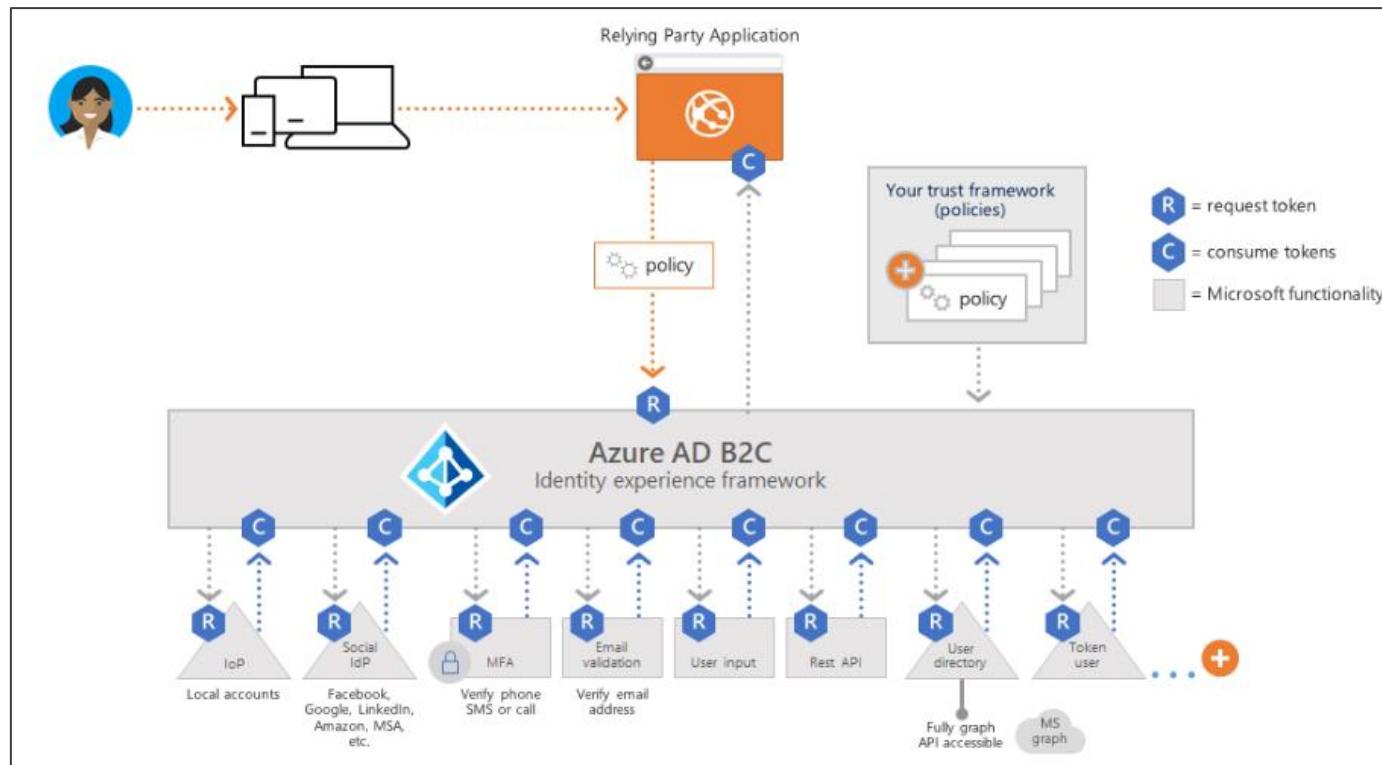
Inspecting Azure AD B2C Traces from App Insights

The screenshot shows a Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** TrustFrameworkBase.xml - IdentityExperienceFramework - Visual Studio Code.
- Sidebar:** EXPLORER, IDENTITYEXPERIENCEFRAMEWORK, OUTLINE, TIMELINE, AZURE AD B2C POLICY EXPLORER, AZURE AD B2C TRACE (APP INSIGHTS). A large white arrow points from the sidebar to the list of exceptions in the main content area.
- Main Content Area:**
 - Application Insights:**
 - Policy: b2c_1a_profileedit
 - Correlation Id: 4d428c42-c6c7-4949-a0bd-e32fb25ce4bd
 - App insights Id: 1761acf4-540c-11ed-8c2e-0022484f730e
 - App insights timestamp: 2022-10-24 22:23:12
 - User journey is completed: No
 - Orchestration steps:
 - Exceptions:**
 - The claim type "alternativeSecurityId", designated as the identifier claim type, could not be found in the claims collection for the claims principal in tenant id "B2C_1A_ProfileEdit".
 - The claim type "alternativeSecurityId", designated as the identifier claim type, could not be found in the claims collection for the claims principal in tenant id "B2C_1A_ProfileEdit".
 - The claim type "alternativeSecurityId", designated as the identifier claim type, could not be found in the claims collection for the claims principal in tenant id "B2C_1A_ProfileEdit".
 - Validation technical profiles:**
 - AAD-UserWriteUsingAlternativeSecurityId
 - Claims:**
 - authenticationSource: localAccountAuthentication
 - displayName: Austin Powers
 - email: austinp@powerbidevcamp.net

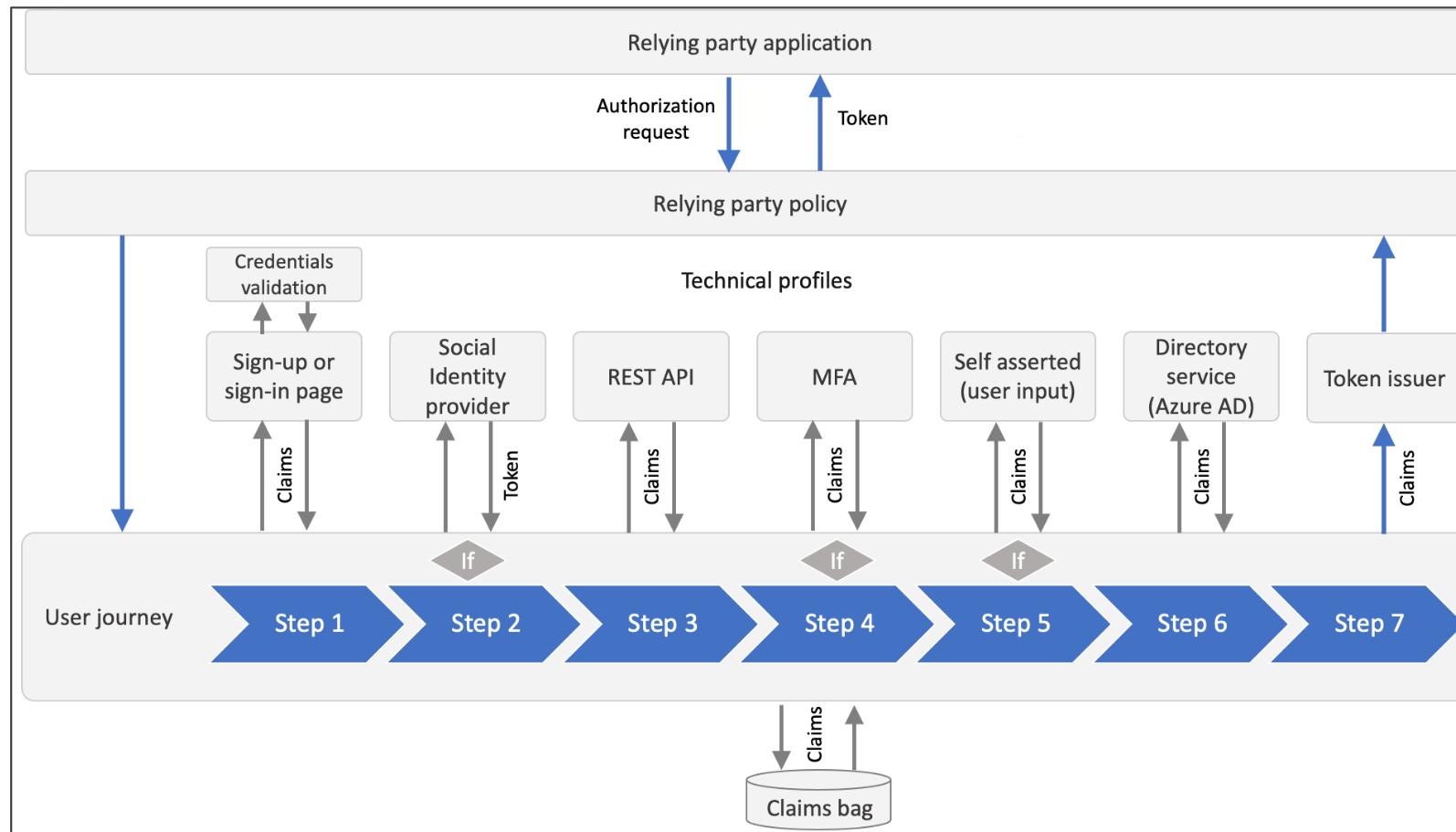
User Journeys and Orchestration Steps

- User Journey defines entry point for custom policy (e.g. **B2C_1A_customSignupSignIn**)
 - User journey constructed as a sequence of orchestration steps
 - Orchestration steps reference and execute technical profiles
 - Orchestration steps can be conditionally skipped based on preconditions



Building Up Claims in the Claims Bag

- After orchestration step completes, AAD B2C stores outputted claims in claims bag
- Claims added to claims bag can be utilized by later orchestration steps in user journey.

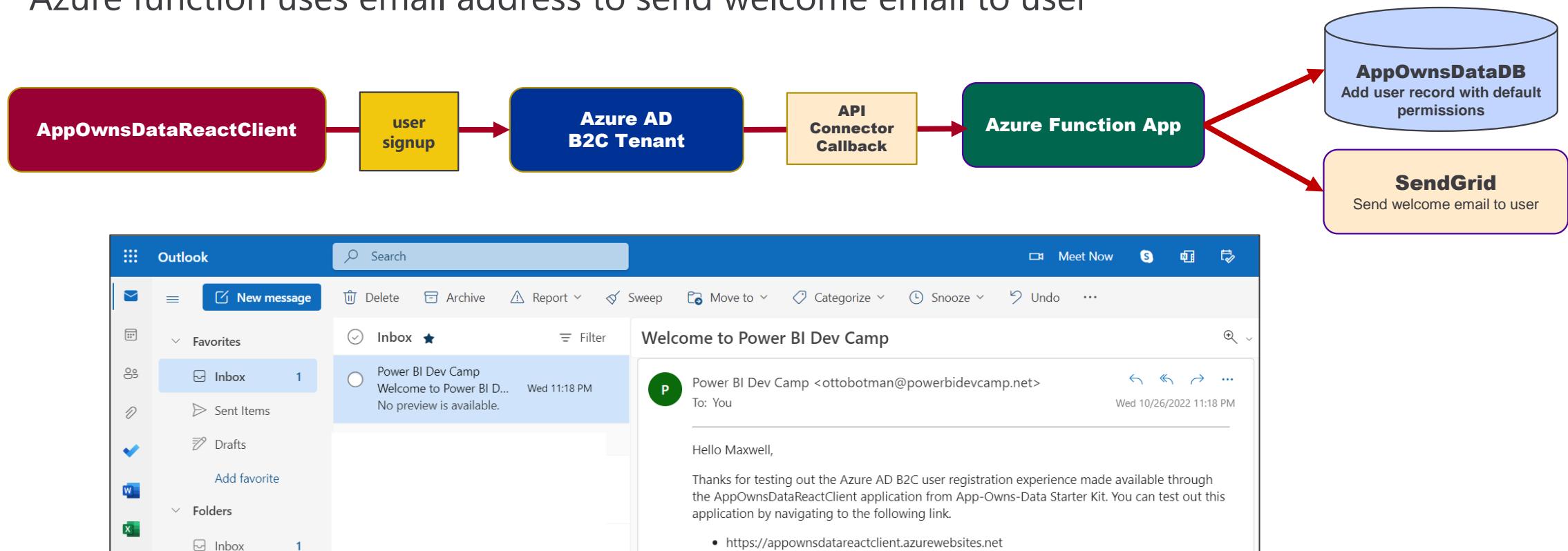


Agenda

- ✓ Understanding Azure AD B2C Architecture
- ✓ Building and Testing User Flows
- ✓ Acquiring B2C tokens using msal-browser.js
- ✓ Managing B2C User Accounts using Microsoft Graph API
- ✓ Developing Custom Policies using Visual Studio Code
- Using Azure functions to implement and deploy API Connectors

Integrating API Connector with a User Signup Policy

- API connector configured to execute after each user signup
 - User goes through signup process using local account or any federated identity provider
 - Completion of signup process triggers execution of Azure function
 - Azure function adds User record to **AppOwnsDataDb** with default permissions (if any)
 - Azure function uses email address to send welcome email to user



Azure Function Triggered in Response to User Signup

```
[FunctionName("OnUserAccountCreated")]
public async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Function, "get", "post")] HttpRequest req, ILogger log) {
    log.LogInformation("API Connection for OnUserAccountCreated executing...");
    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    log.LogInformation("Request Body: " + requestBody);
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    if (data != null && data.email != null && data.givenName != null && data.displayName != null) {
        string email = data.email;
        string firstName = data.givenName;
        string displayName = data.displayName;
        string defaultTenant = "Demo Tenant";
        if (this.appOwnsDataService.GetUser(email) == null) {
            var user = this.appOwnsDataService.CreateUser(new User {
                LoginId = email,
                UserName = displayName,
                TenantName = defaultTenant,
                CanEdit = false,
                CanCreate = false,
                TenantAdmin = false
            });
            this.appOwnsDataService.PostActivityLogEntry(new ActivityLogEntry {
                Activity = "CreateAccount",
                LoginId = user.LoginId,
                User = user.UserName,
                Tenant = defaultTenant
            });
            log.LogInformation("Adding new user: " + email);
            bool mailSentSuccessfully = MailManager.SendWelcomeEmail(email, firstName);
            if (mailSentSuccessfully) {
                log.LogInformation("Welcome message sent to " + displayName + " at " + email);
            }
            else {
                log.LogInformation("Welcome message could not be sent to " + displayName + " at " + email);
            }
            return (ActionResult)new OkObjectResult(new ResponseContent("Continue", "New user has been created."));
        }
        else {
            log.LogInformation("User already existed: " + email);
            return (ActionResult)new OkObjectResult(new ResponseContent("Continue", "This user was previously created."));
        }
    }
}
```

Summary

- ✓ Understanding Azure AD B2C Architecture
- ✓ Building and Testing User Flows
- ✓ Acquiring B2C tokens using msal-browser.js
- ✓ Managing B2C User Accounts using Microsoft Graph API
- ✓ Developing Custom Policies using Visual Studio Code
- ✓ Using Azure functions to implement and deploy API Connectors

Questions?

Microsoft Power BI