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**

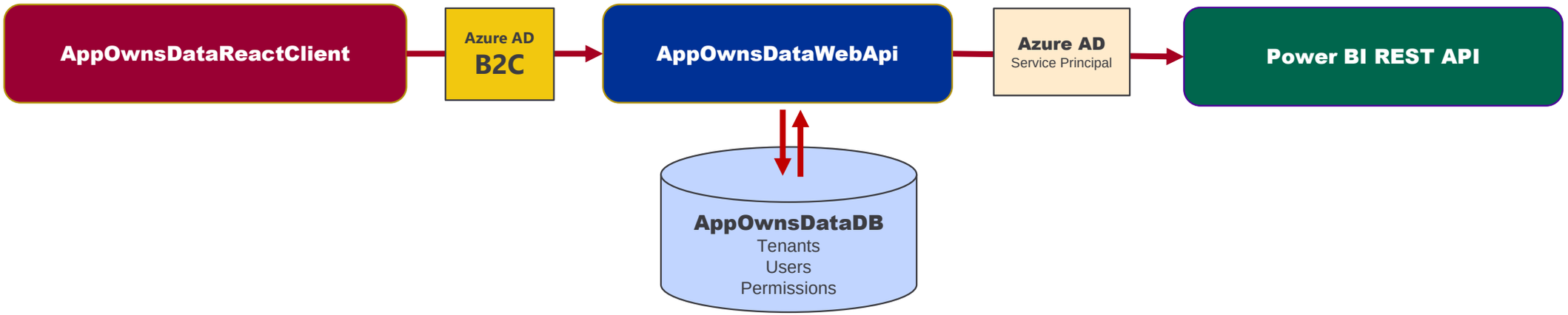
Comparing Azure AD B2B versus Azure AD B2C

- 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

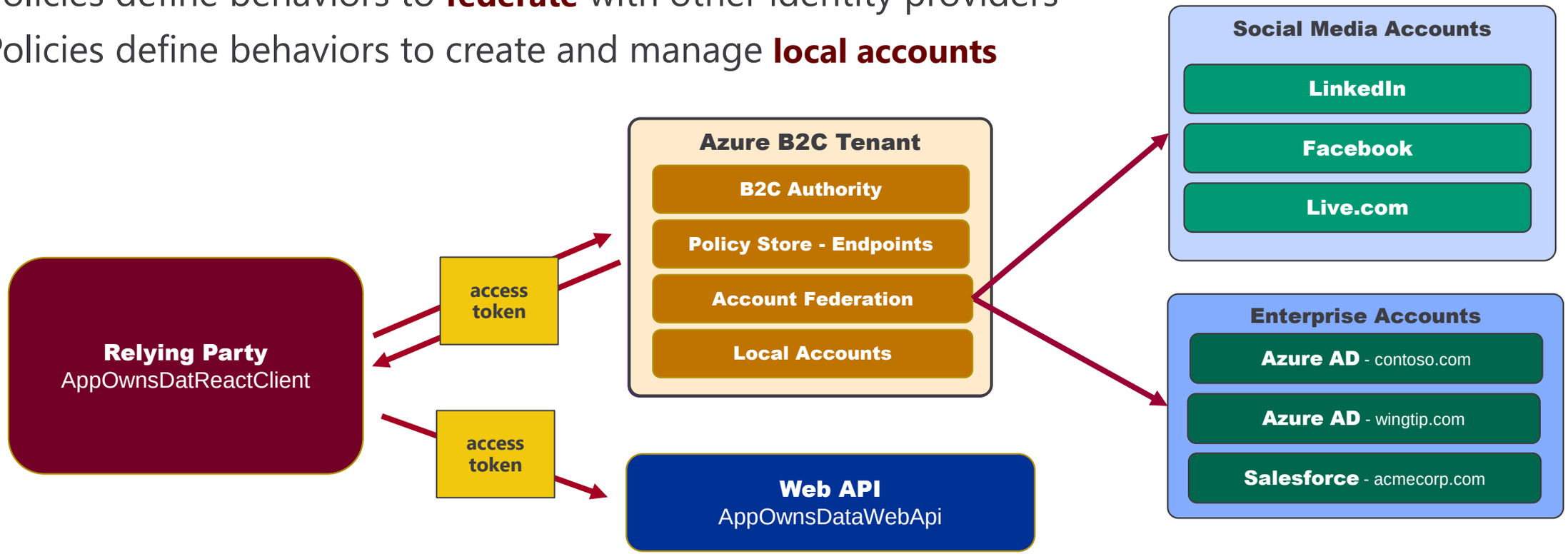
- Let's walk through a typical App-Owns-Data embedding solution



- 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)

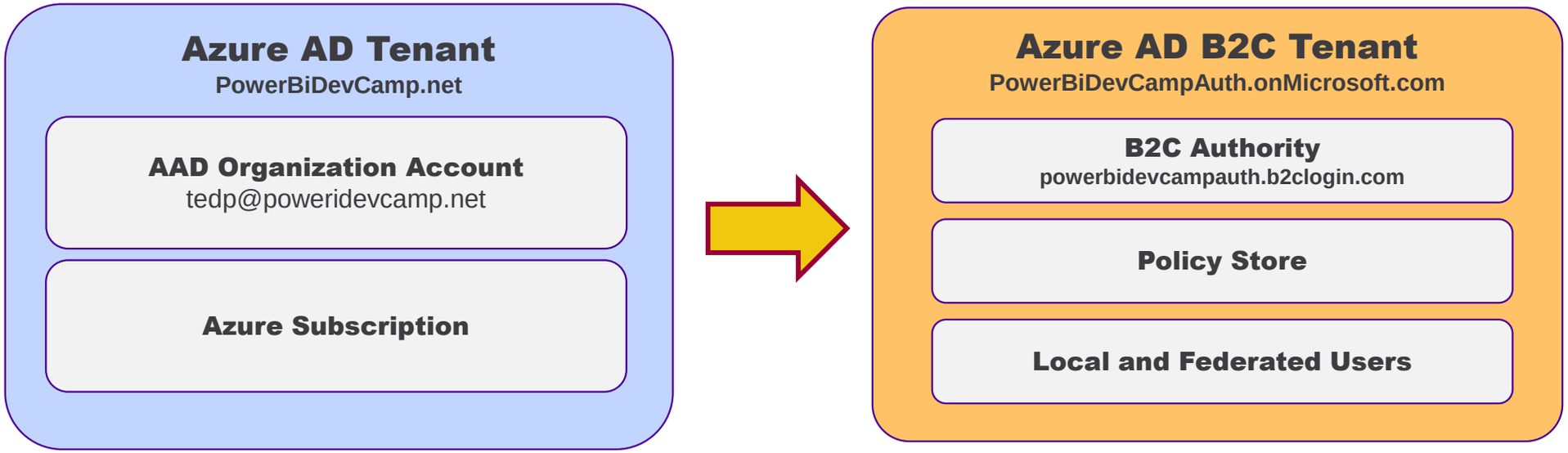
Understanding Azure AD B2C Architecture

- 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**

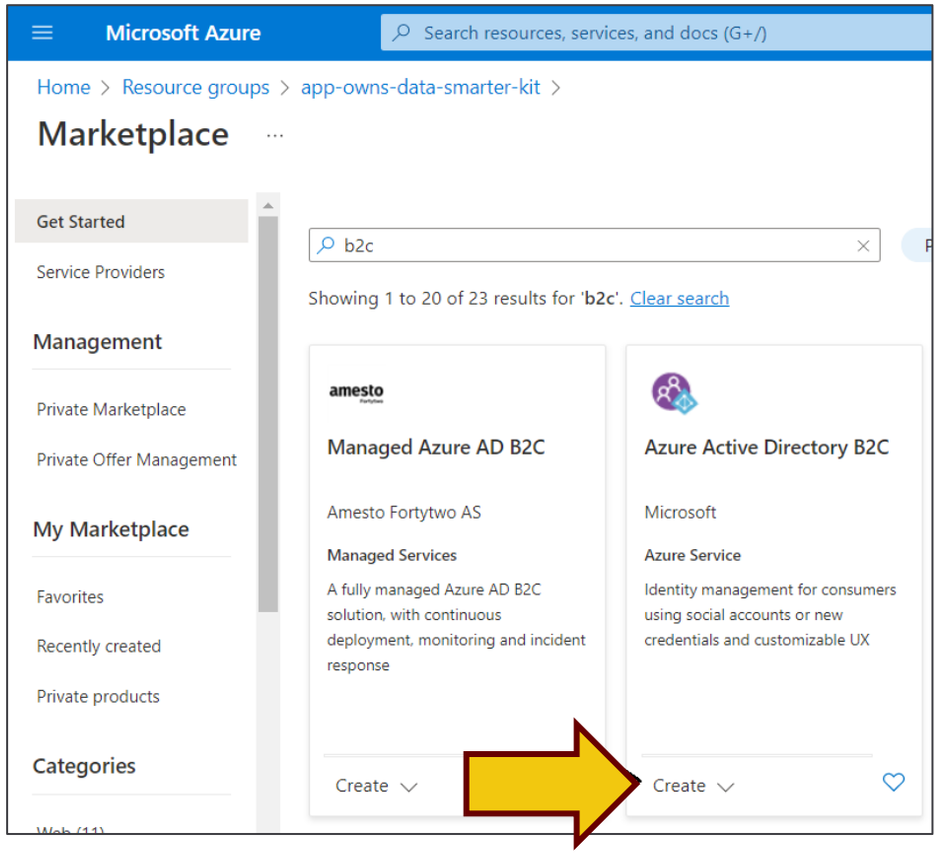


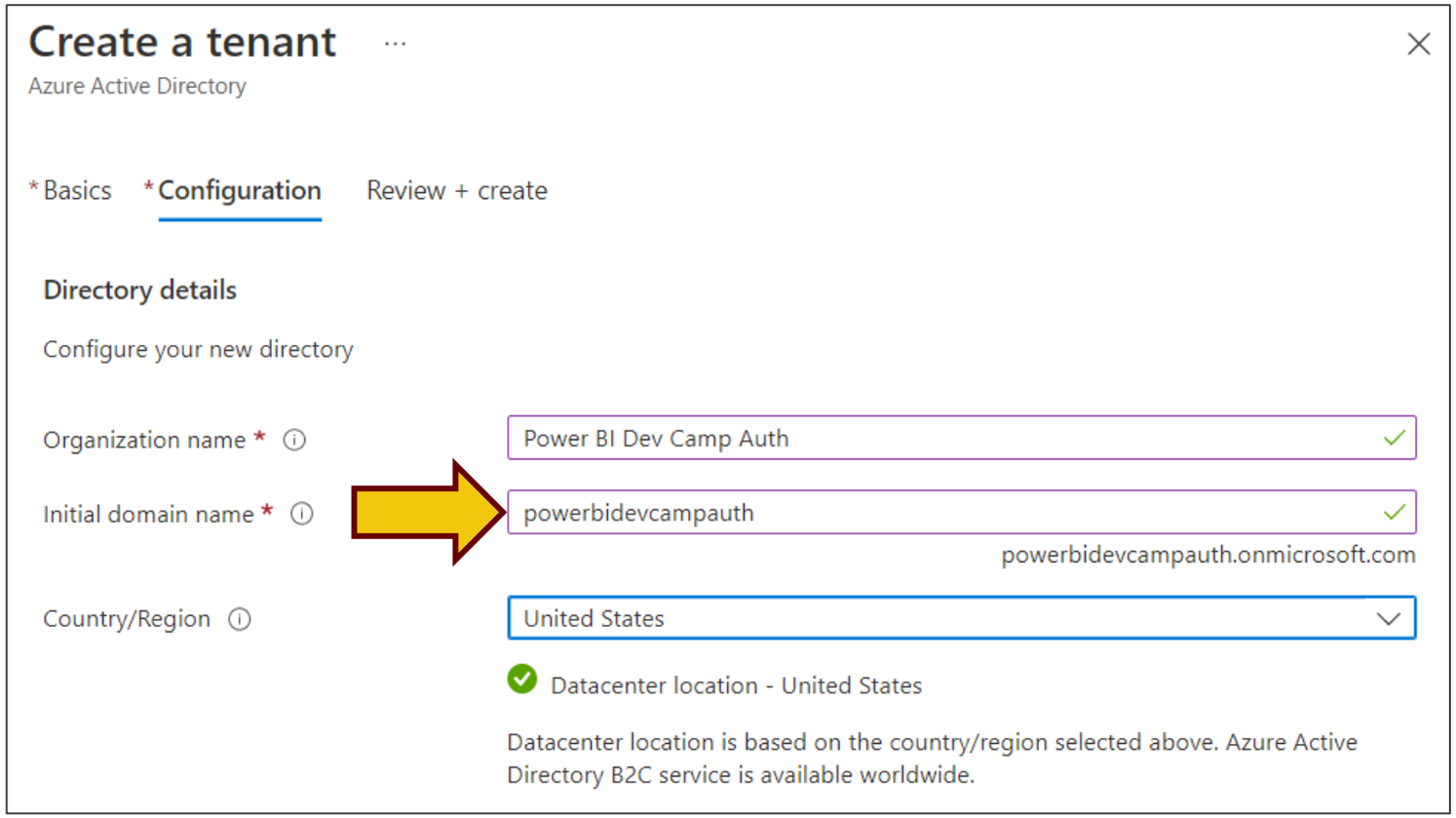
Creating an Azure AD B2C New Tenant

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

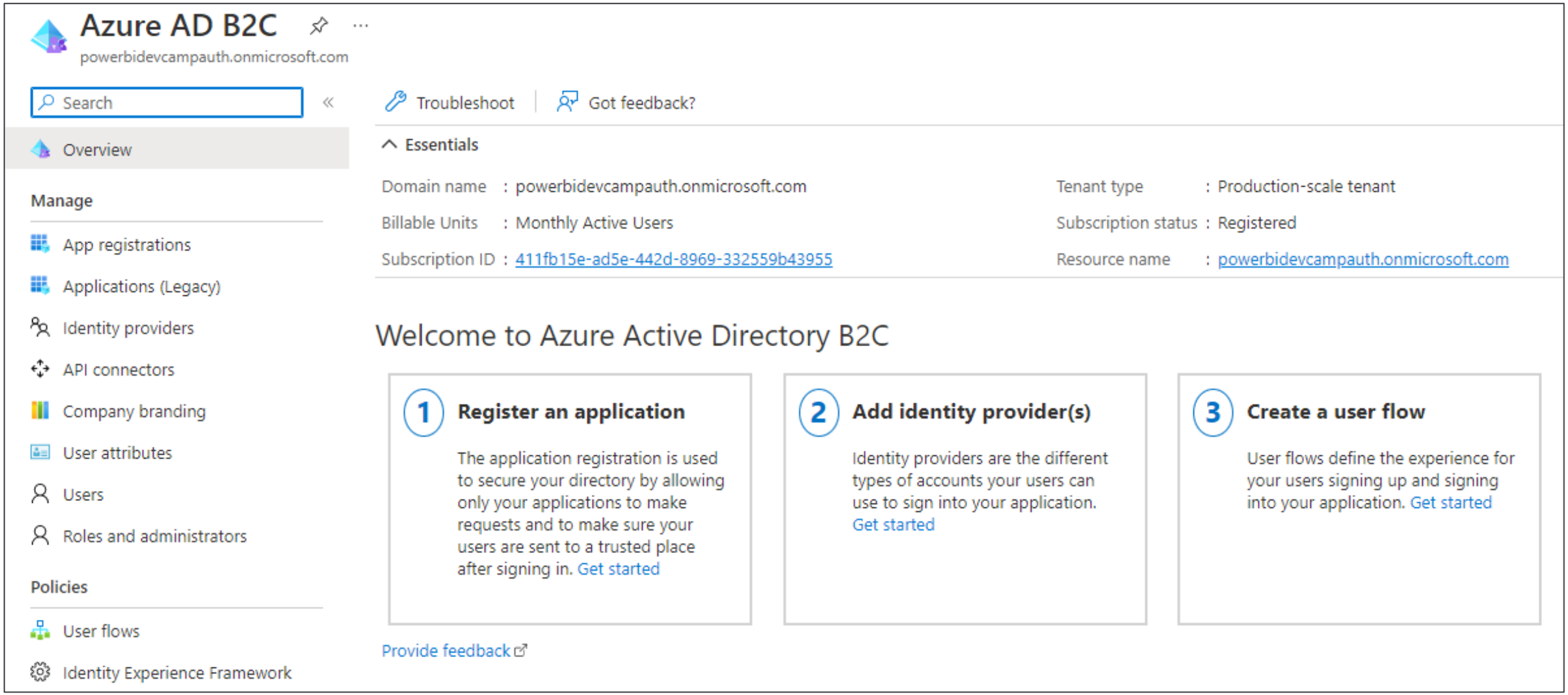


Creating an Azure AD B2C Tenant in the Azure Portal



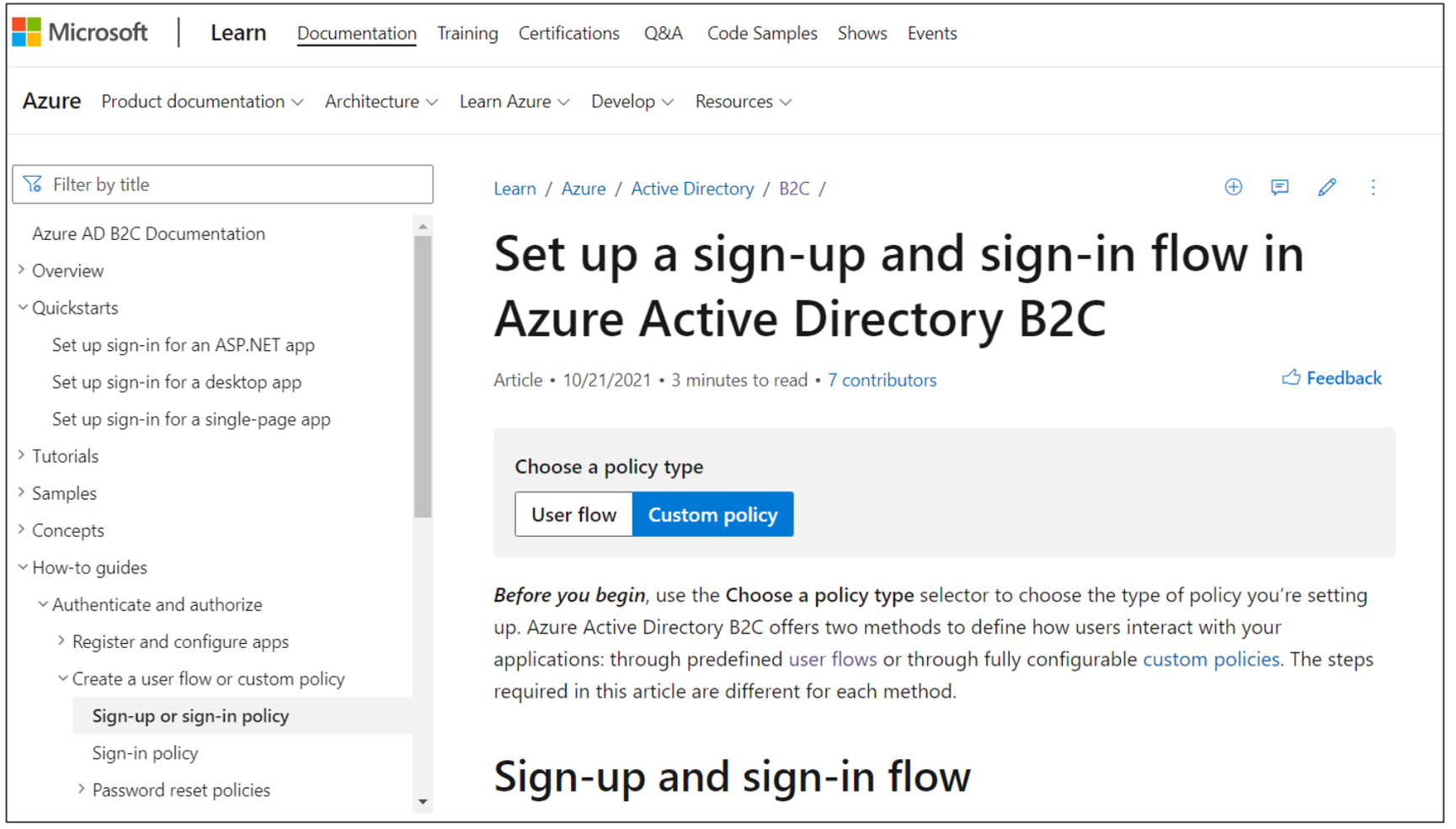


Demo 1 – Getting Around in Azure AD B2C Tenant Portal



If Success Is Important, Embrace the Azure AD B2C Docs

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



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

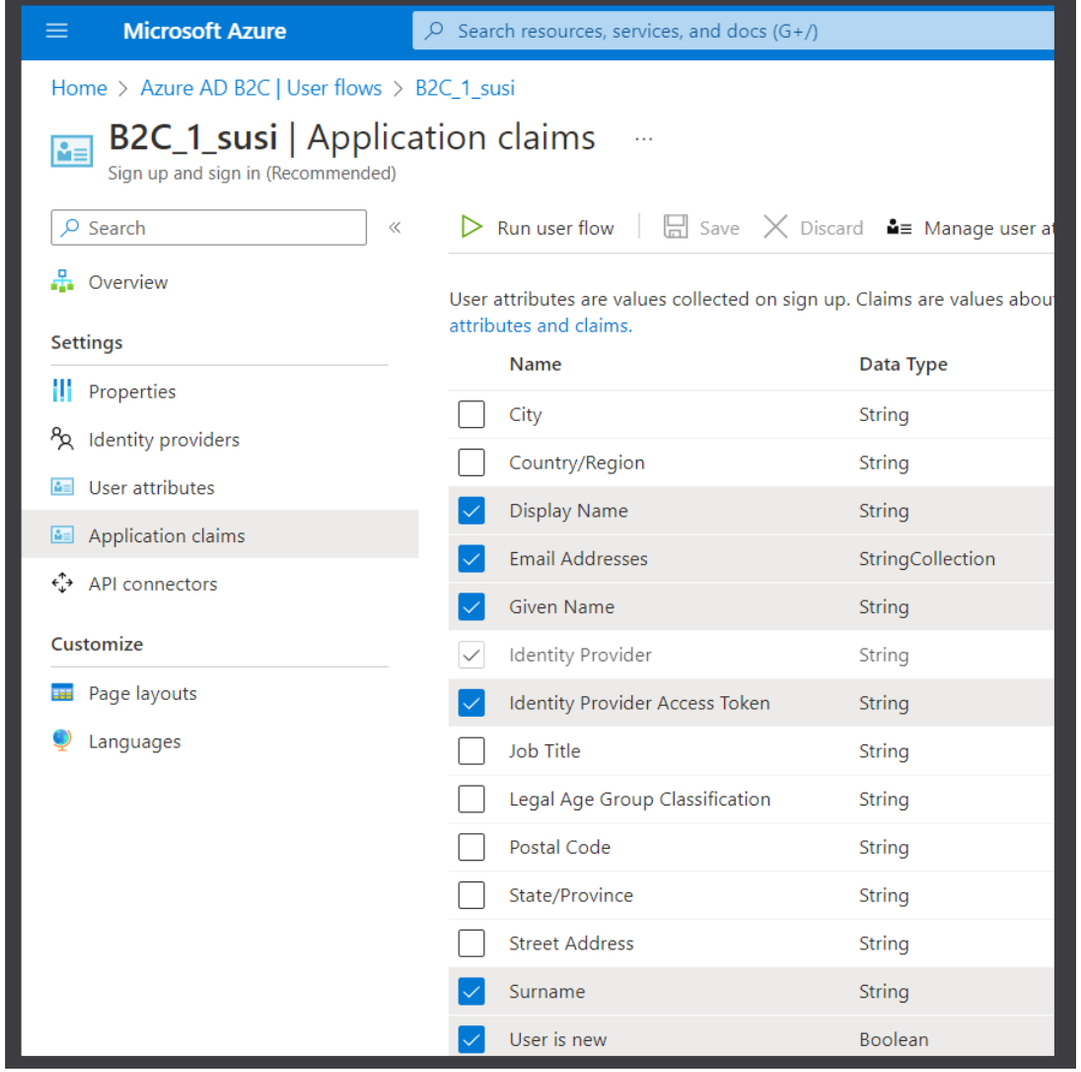
- There are two ways to create a B2C policy
  - **User flows** – UI-based experience for creation and configuration
  - **Custom policies** – Developers work with XML files and Identity Experience Framework (IEF)

- 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

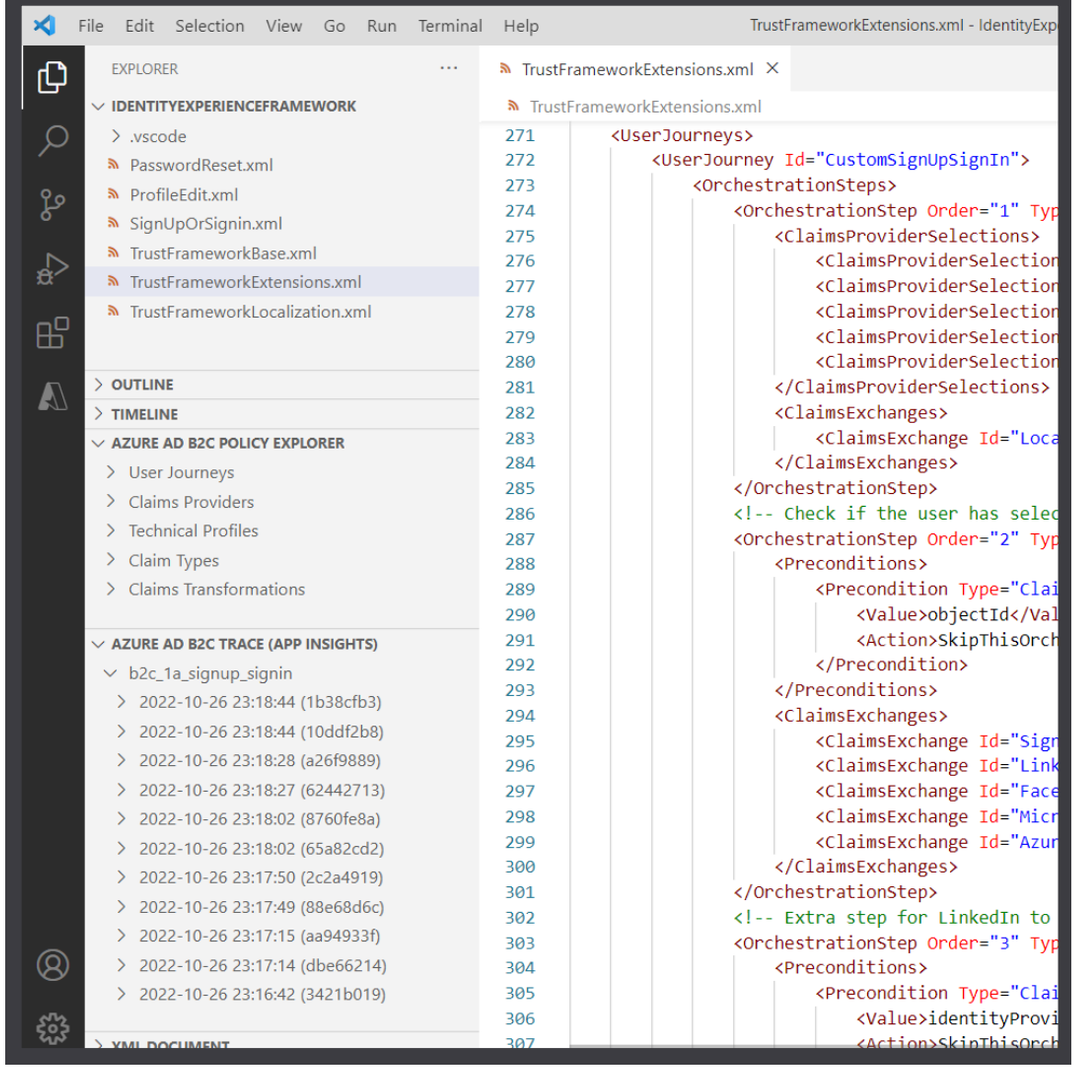
- Reasons to choose custom policies over user flows
  - **More control of what claims go in token** – for example, ensure all users have **email** claim
  - **Federation with enterprise accounts** – user flows only support federating social media accounts
  - **Merging accounts during signup** - with the same email address

User flow versus Custom Policy Experience for Creators

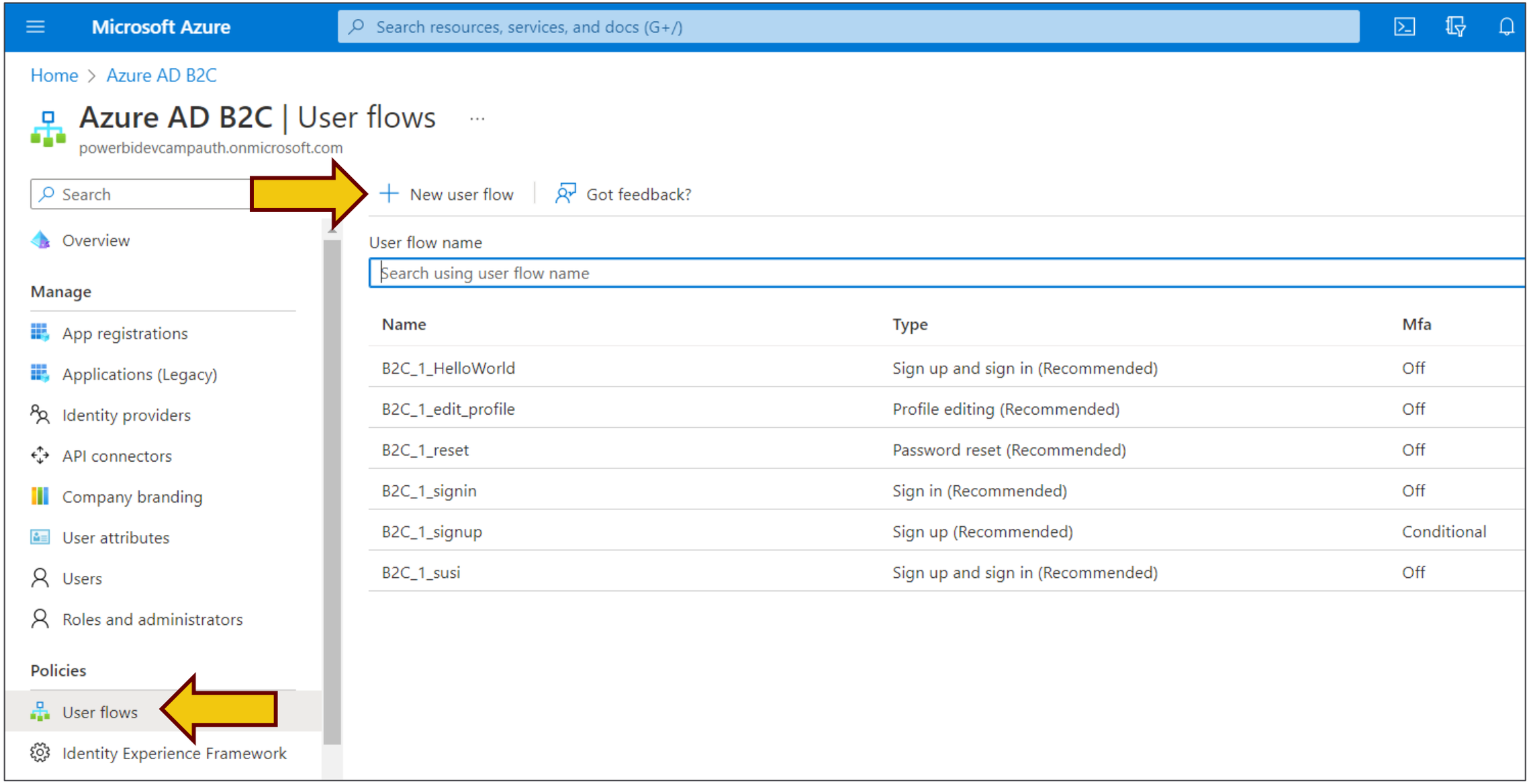
User Flow Creators Experience



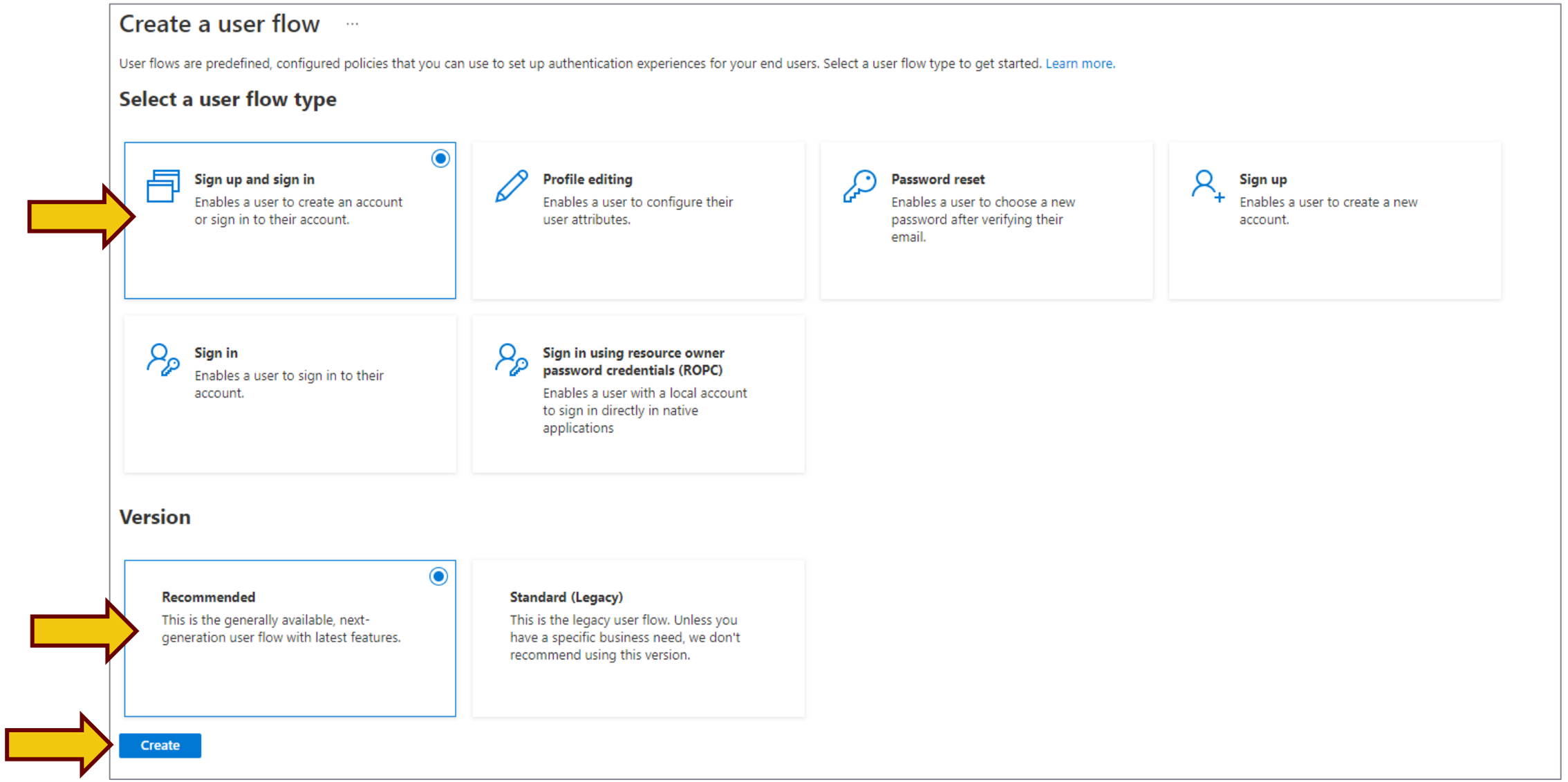
Custom Policy Development Experience



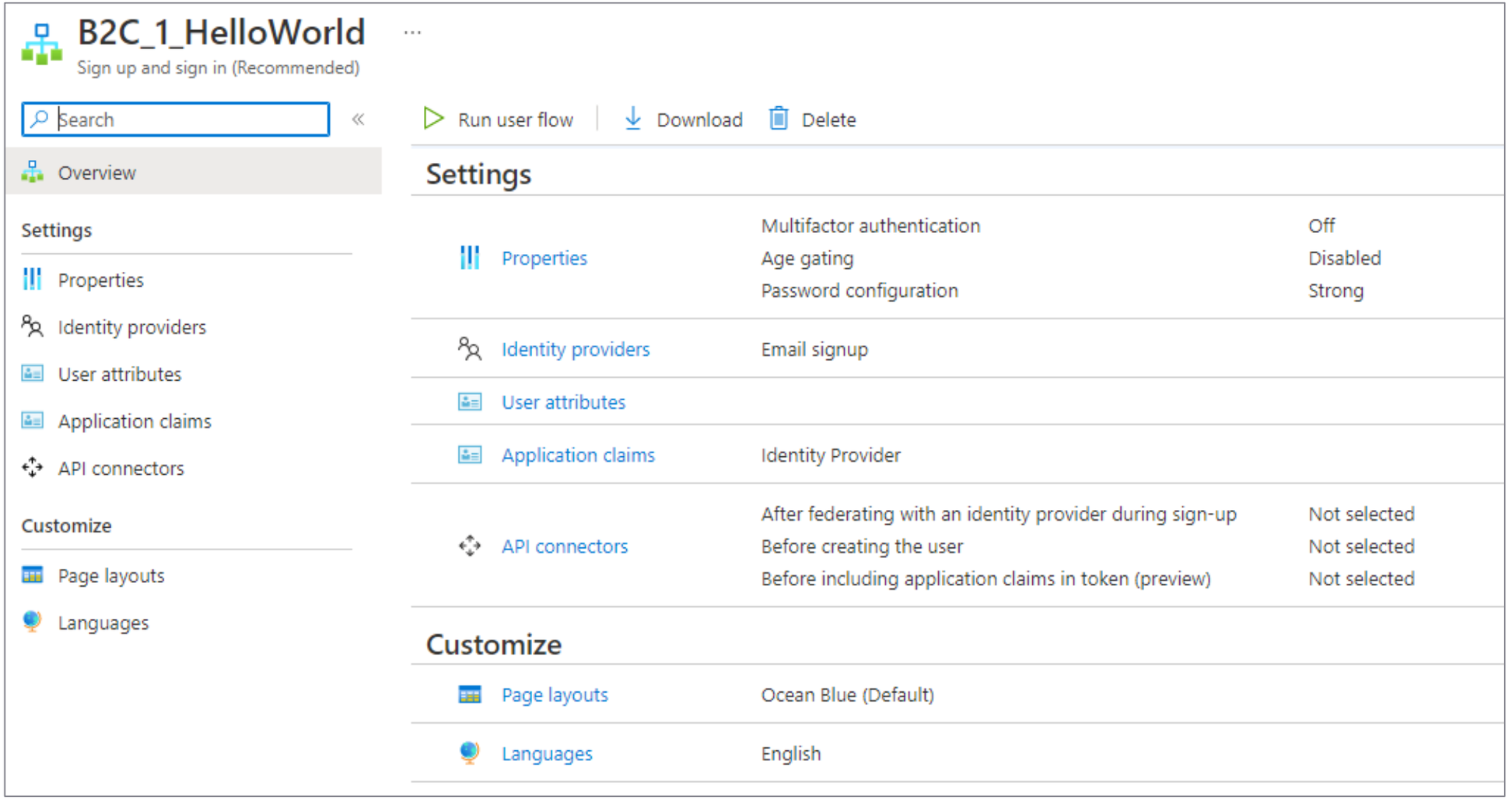
Working with User Flows in the Azure AD B2C Portal



Creating a New User Flow

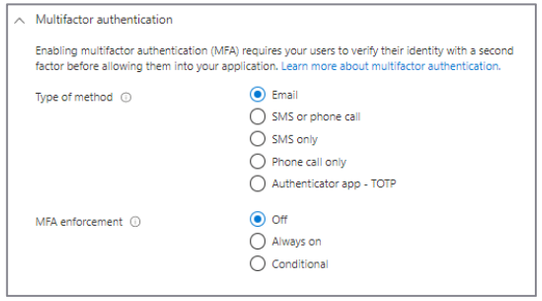


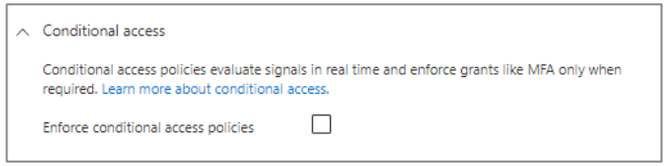
User Flow Configuration Options



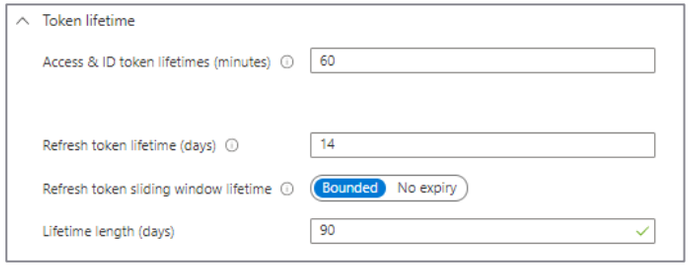
User Flow Configuration Properties

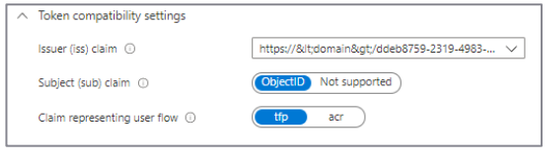
- Configure properties to control **MFA** and **conditional access**



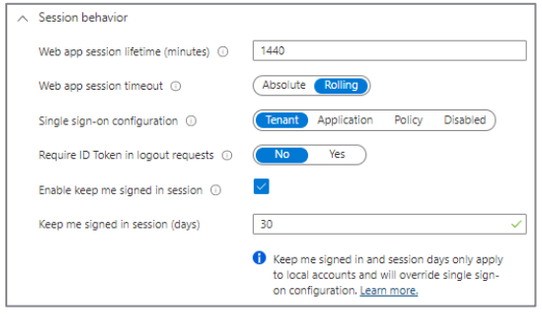


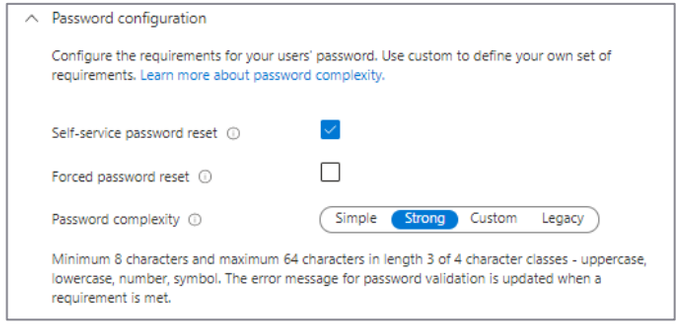
- Configure properties to control **token lifetimes** and **claims compatibility**



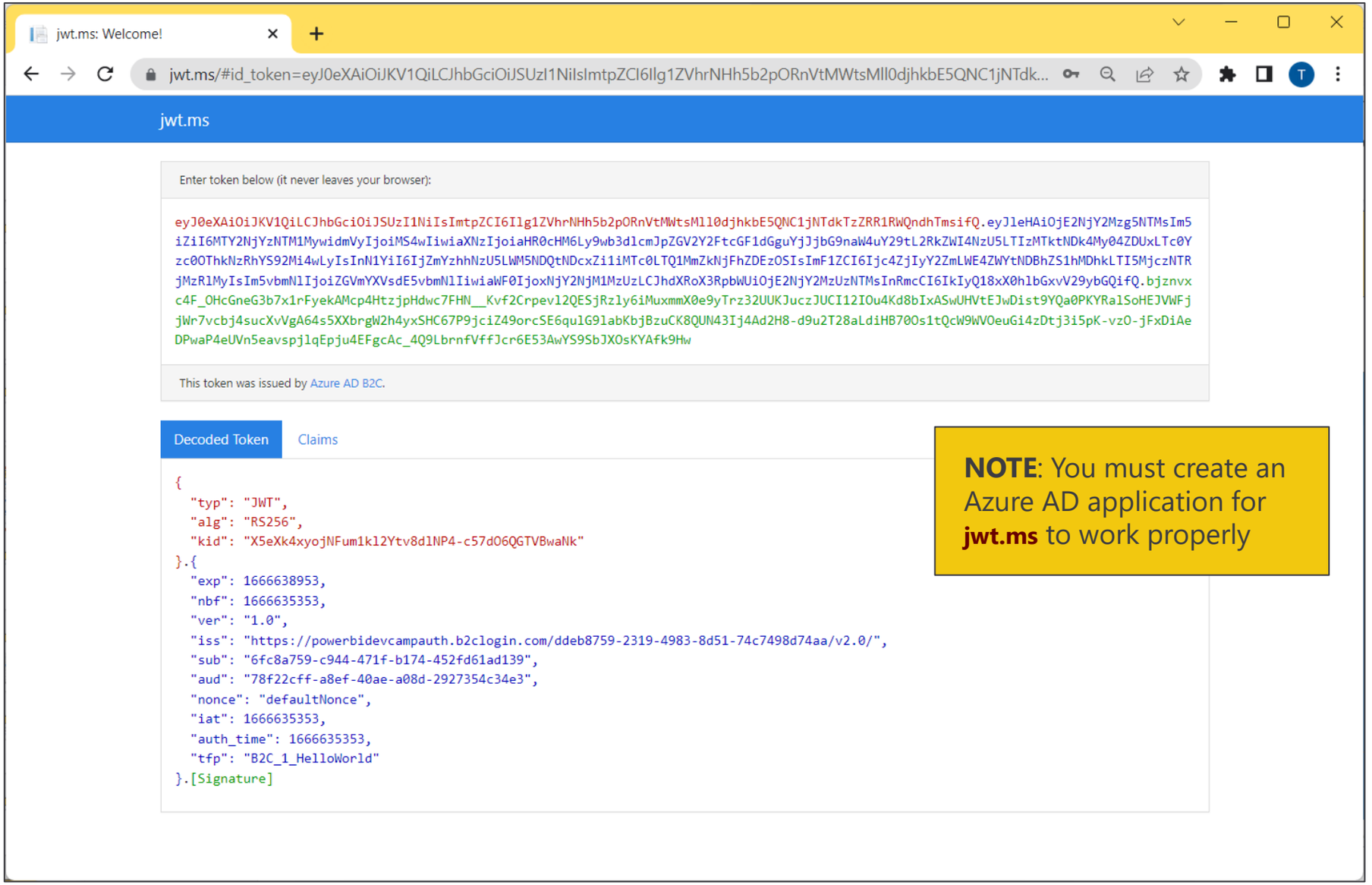


- Configure properties to control **session behavior** and to enable **support for password reset**

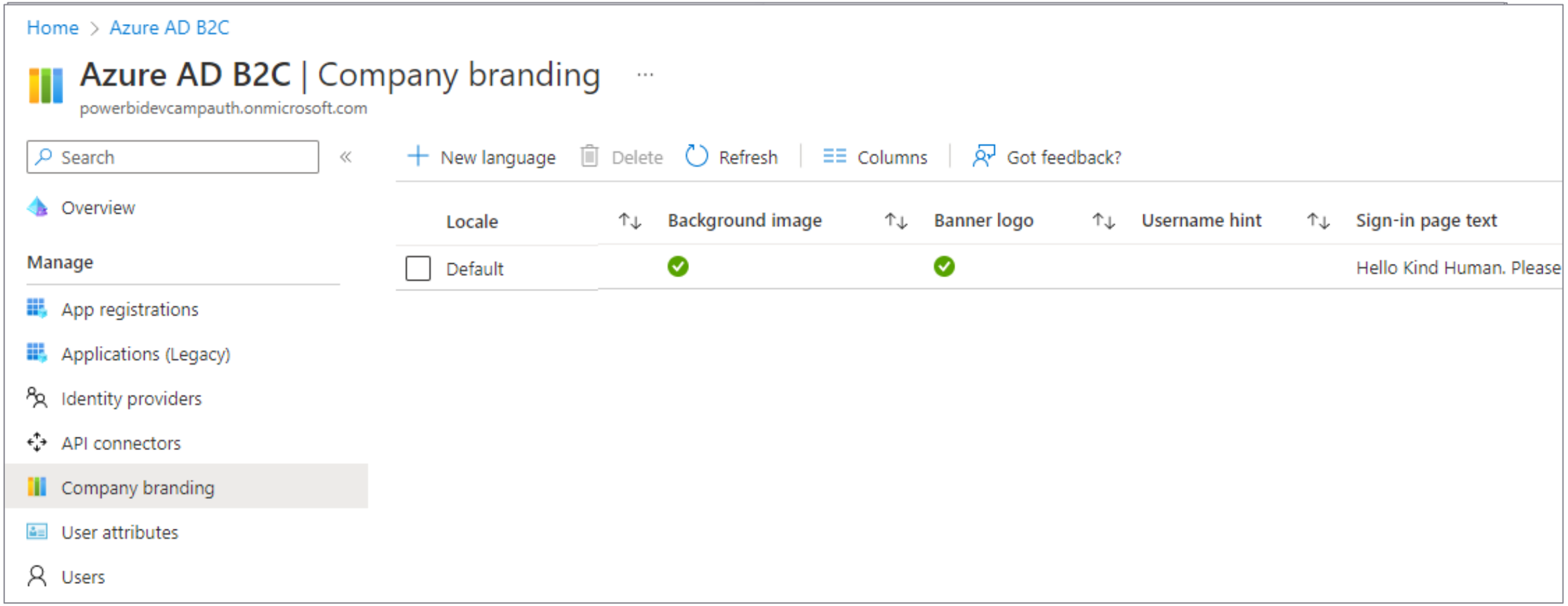




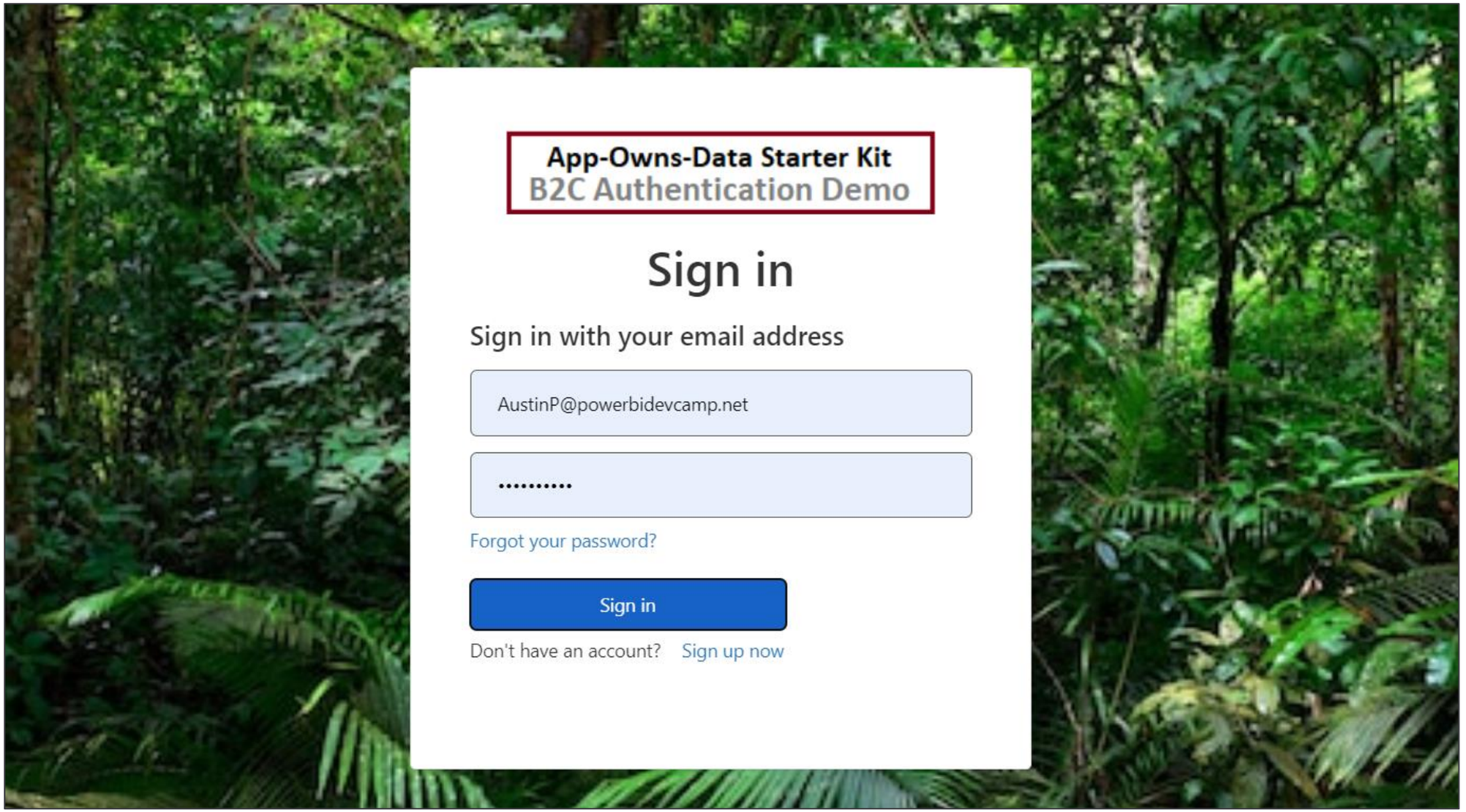
Testing a User Flow for Local Account Signup



Tenant Settings for Company Branding

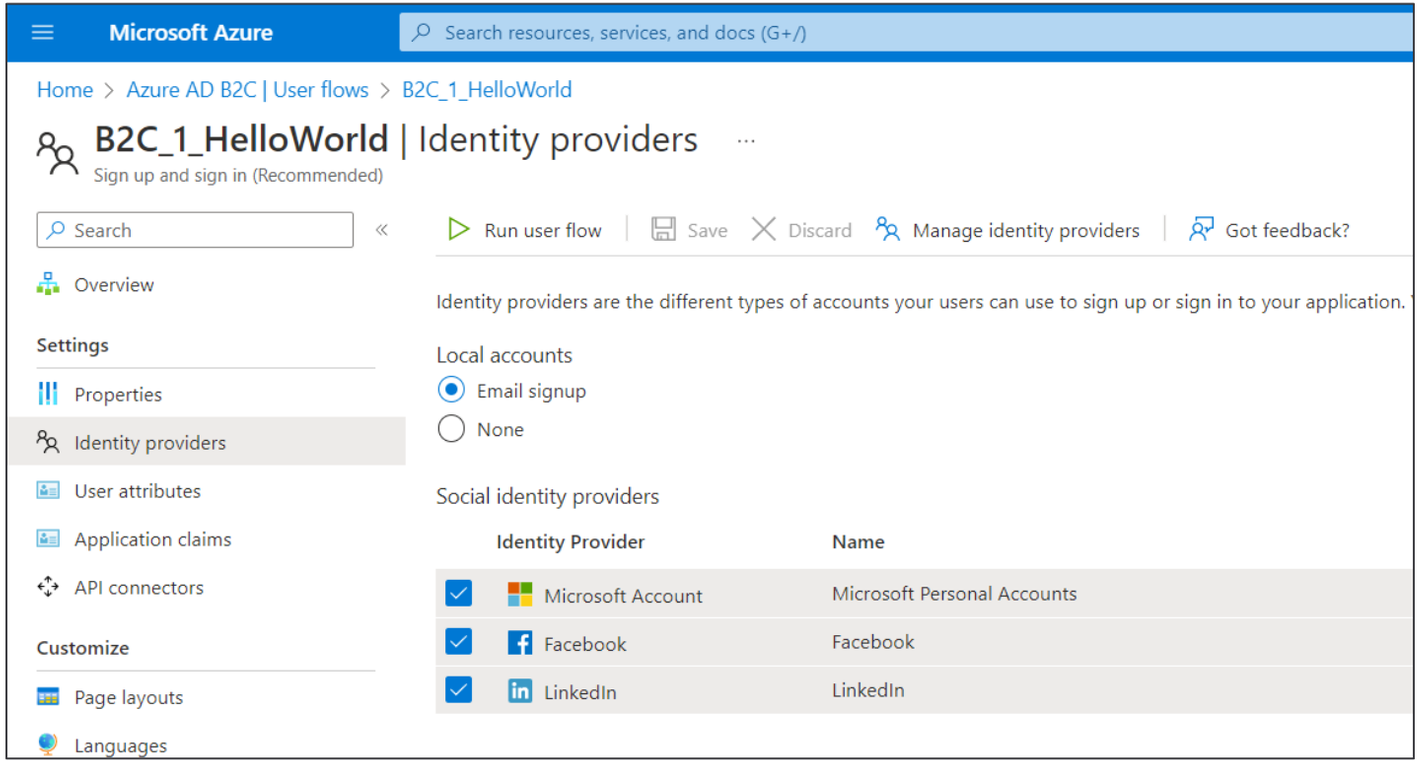


Company Branding Provides Instant Gratification

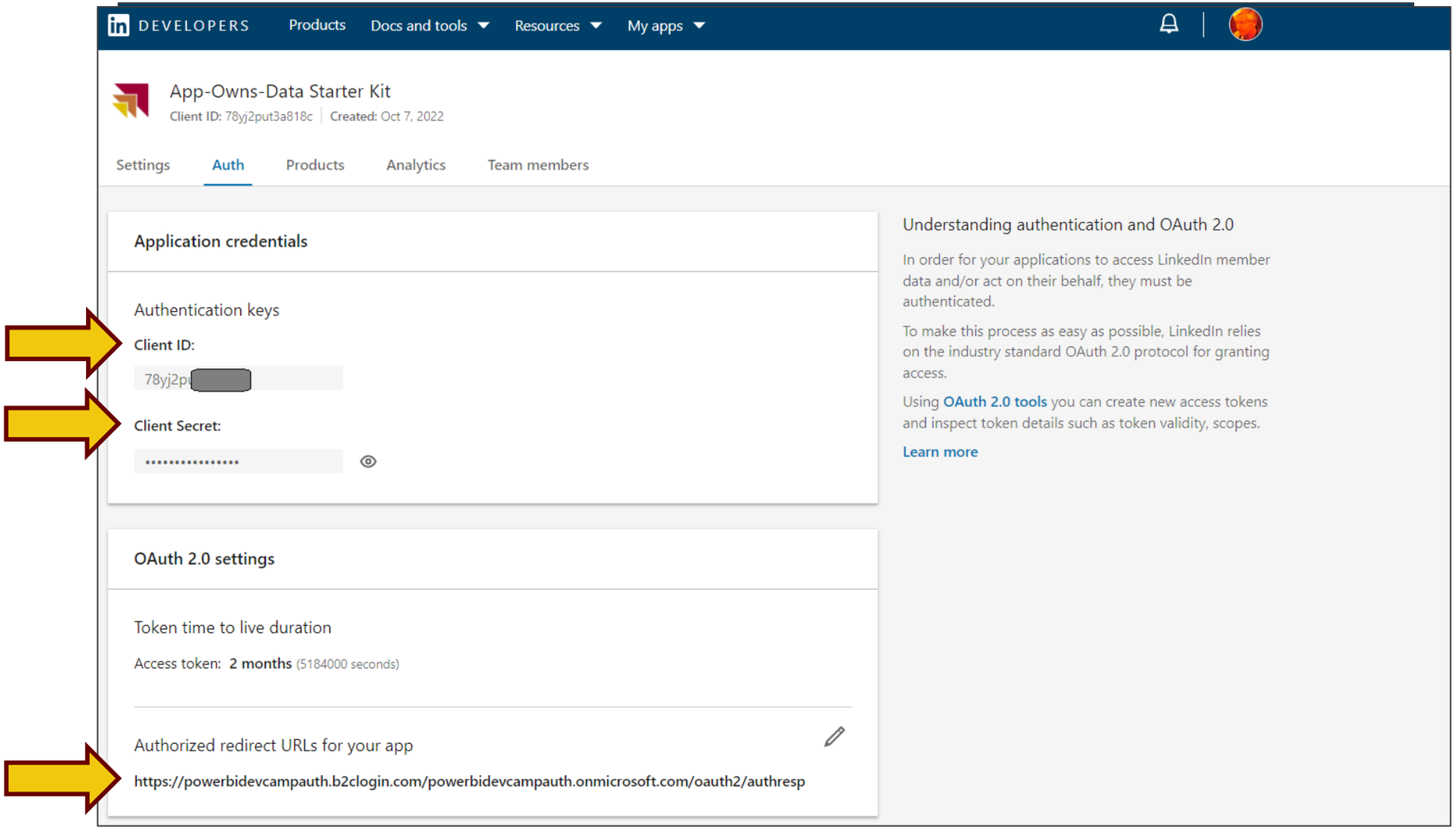


Identity Providers and Social Account Federation

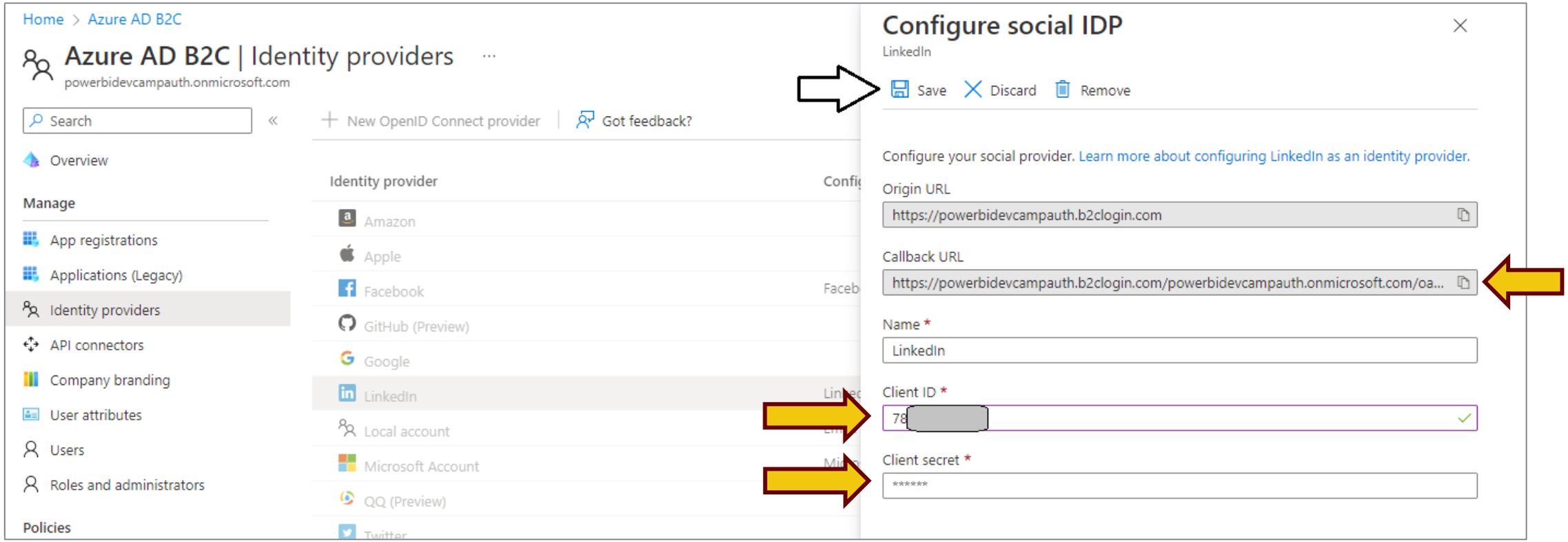
- Users can sign in with existing accounts from popular social identity providers
  - Built-in support to federate accounts from LinkedIn, Facebook, Google, Twitter, etc.
  - Integration requires creating an “app” in the external identity provider
  - You configure support for external identity provider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



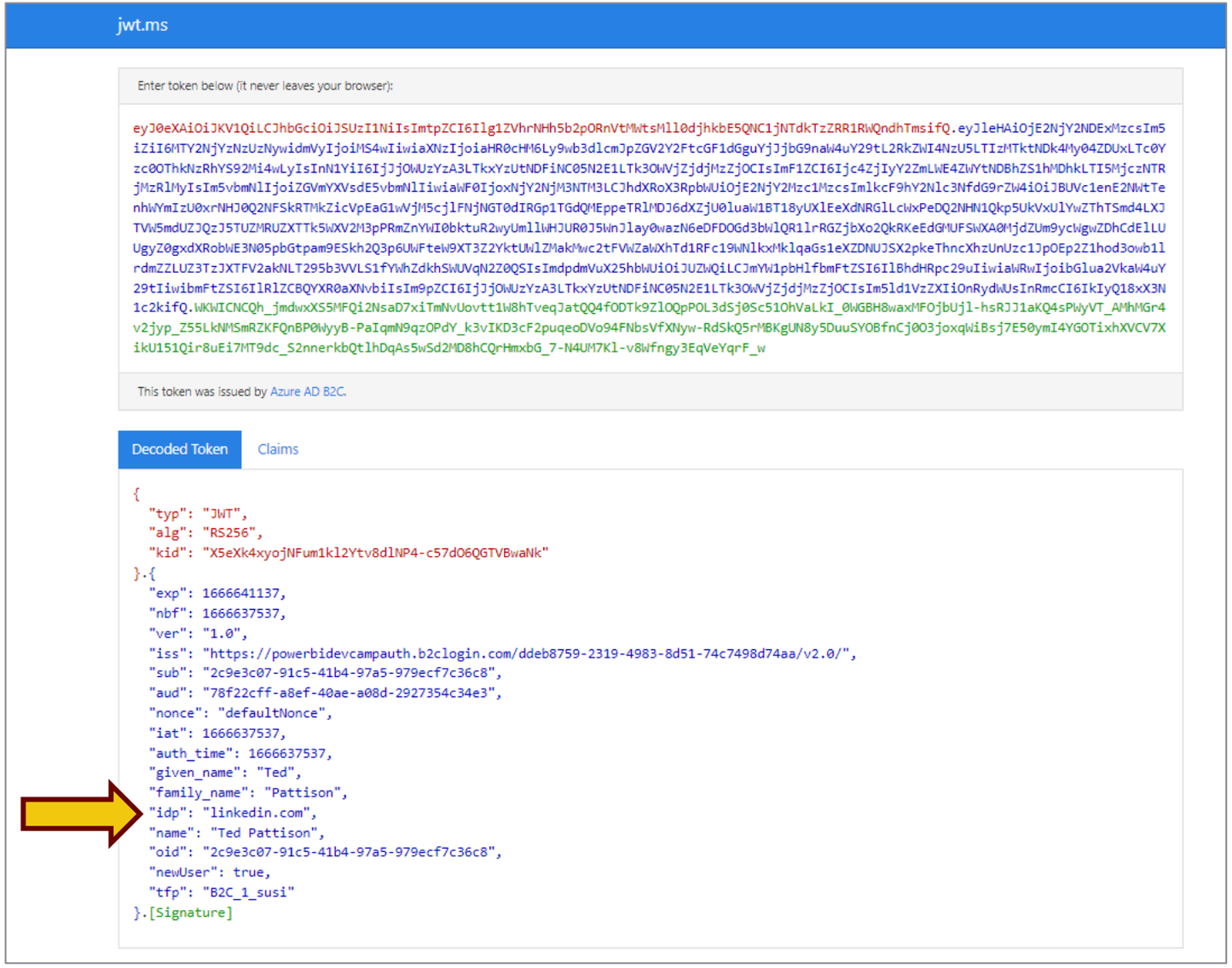
Creating Apps To Integrate with Social Identity Providers



Configuring Identity Provider in AAD B2C Portal



Signup and Sign In with Federated User Accounts

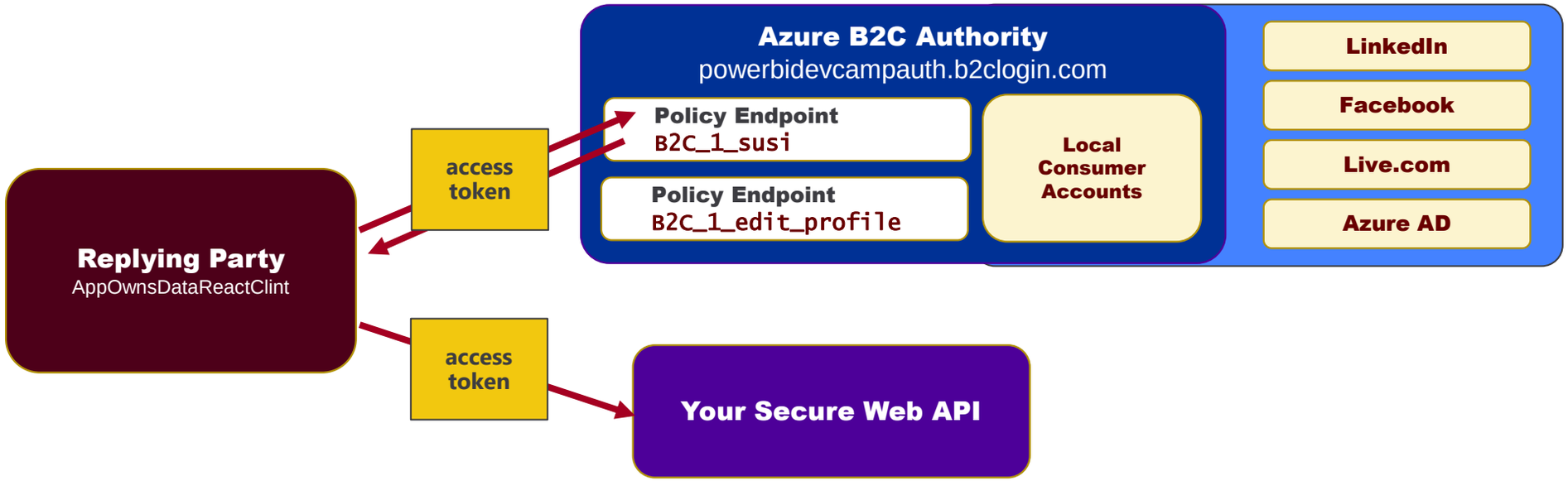


Agenda

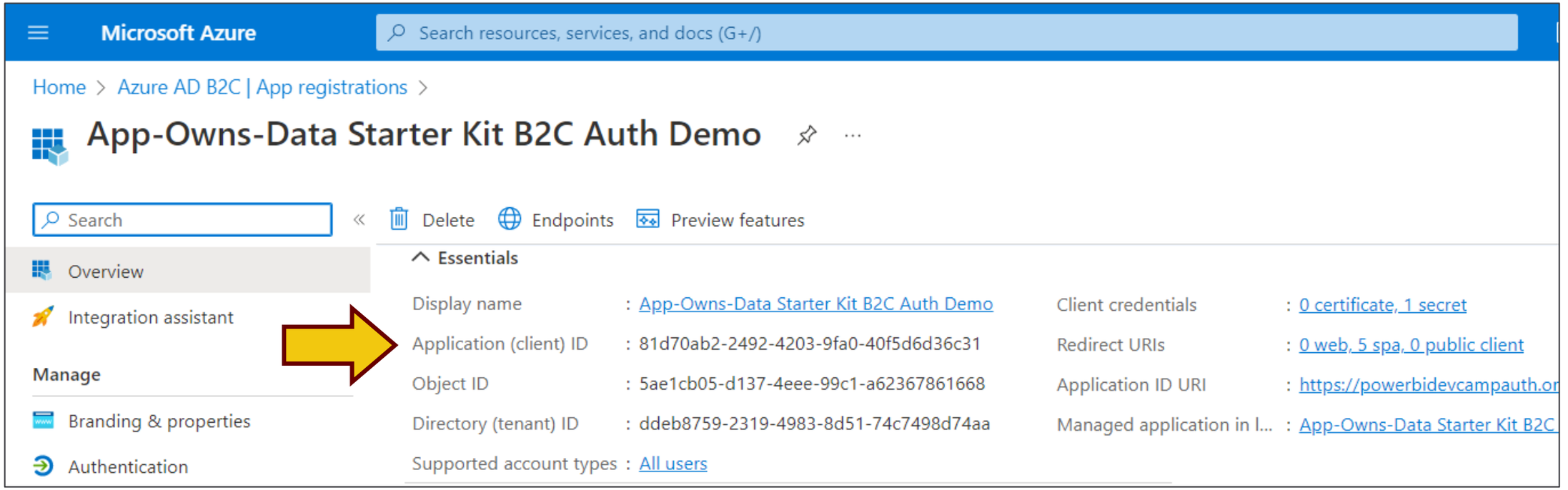
- ✓ Understanding Azure AD B2C Architecture
- ✓ Building and Testing User Flows
- ➢ Acquiring B2C tokens using msal-browser.js
- Managing B2C User Accounts using Microsoft Graph API
- Developing Custom Policies using Visual Studio Code
- Using Azure functions to implement and deploy API Connectors

Acquiring Tokens from AD B2C Authority

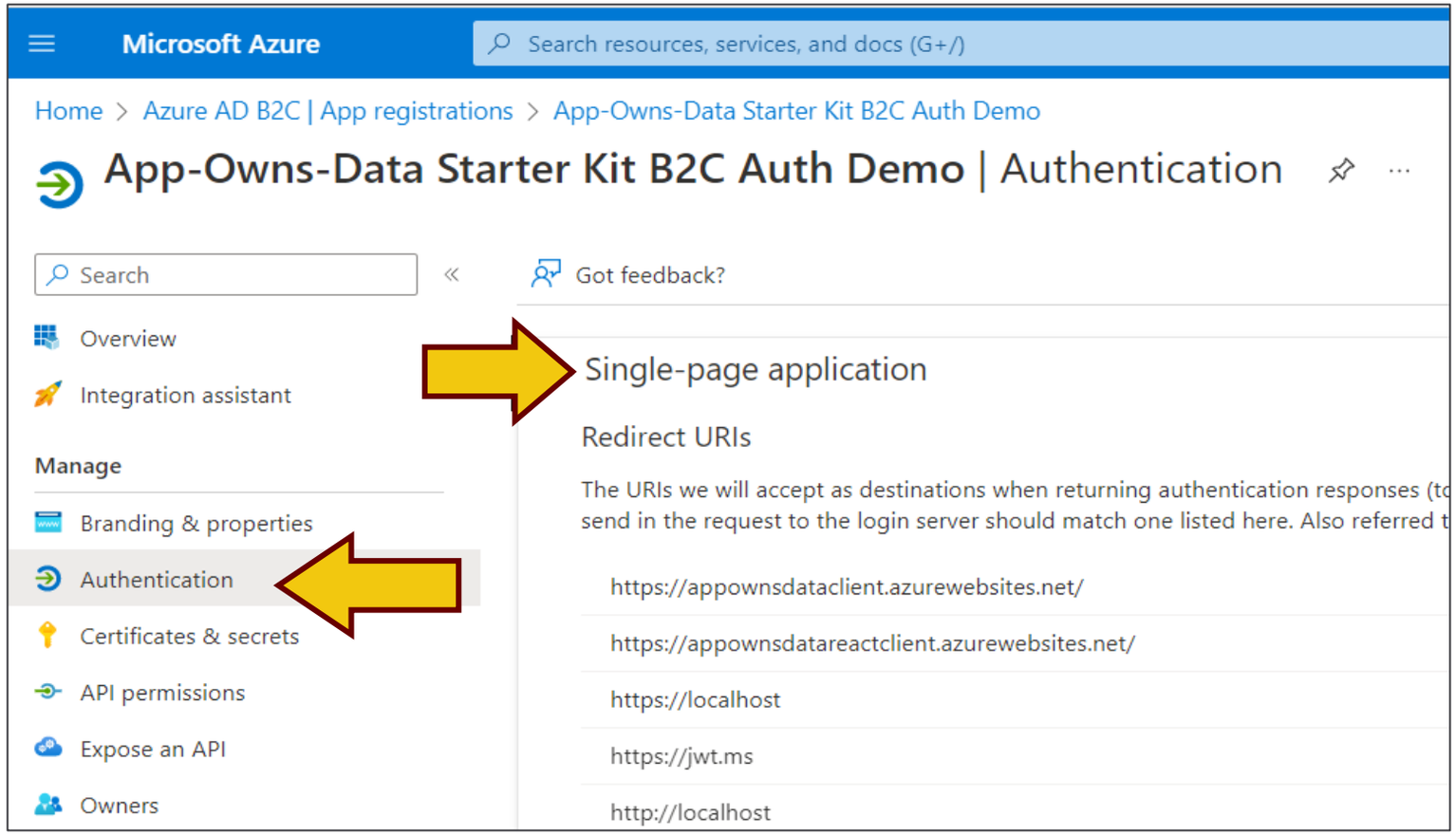
- Each Azure AD B2C authentication endpoint defined by a policy
  - Replying party redirects the user to policy endpoint to begin authentication flow
  - Replying party must include scope(s) in request to policy endpoint to acquire access token
  - B2C authority can optionally provide interactive experience 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 use has required access



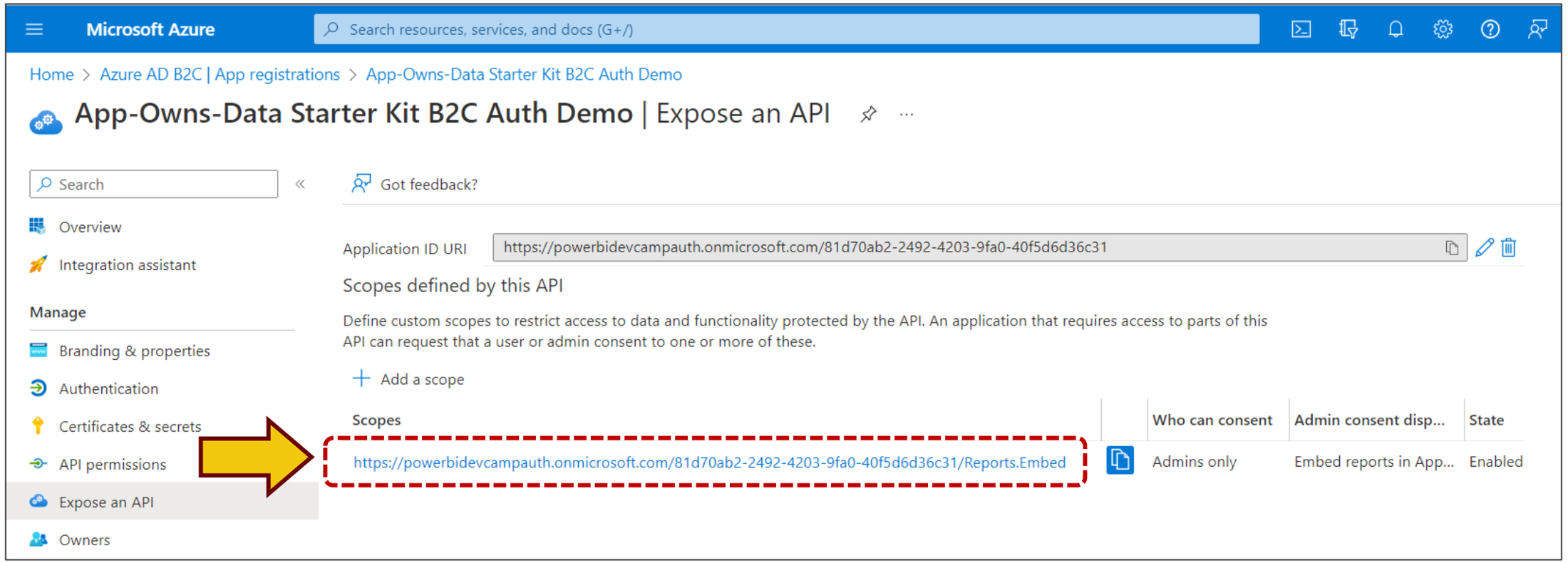
Creating an Application in the Azure AD B2C Tenant



Configuring Application with SPA Authentication

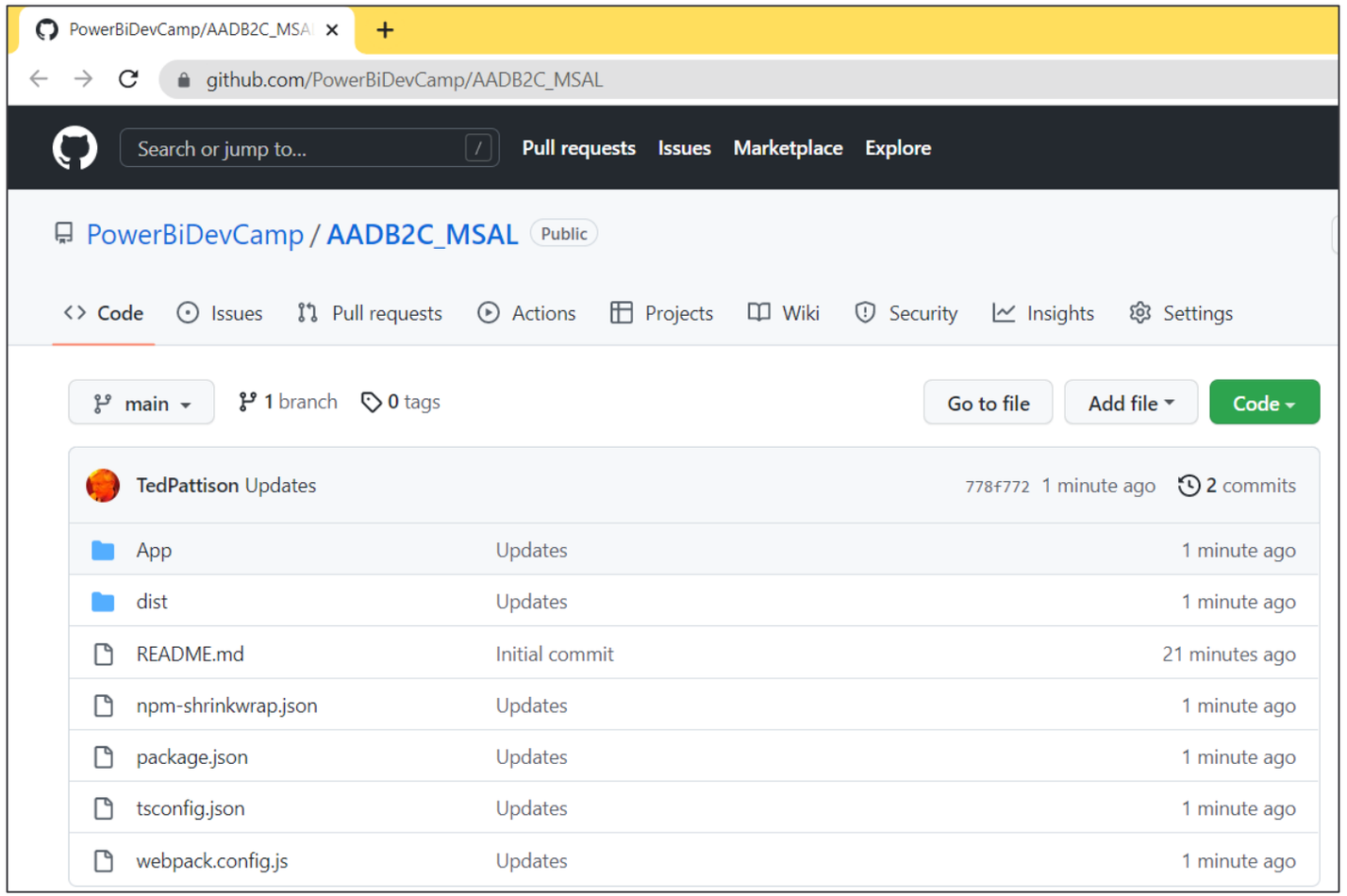


Defining Custom Scope to Expose a Web API

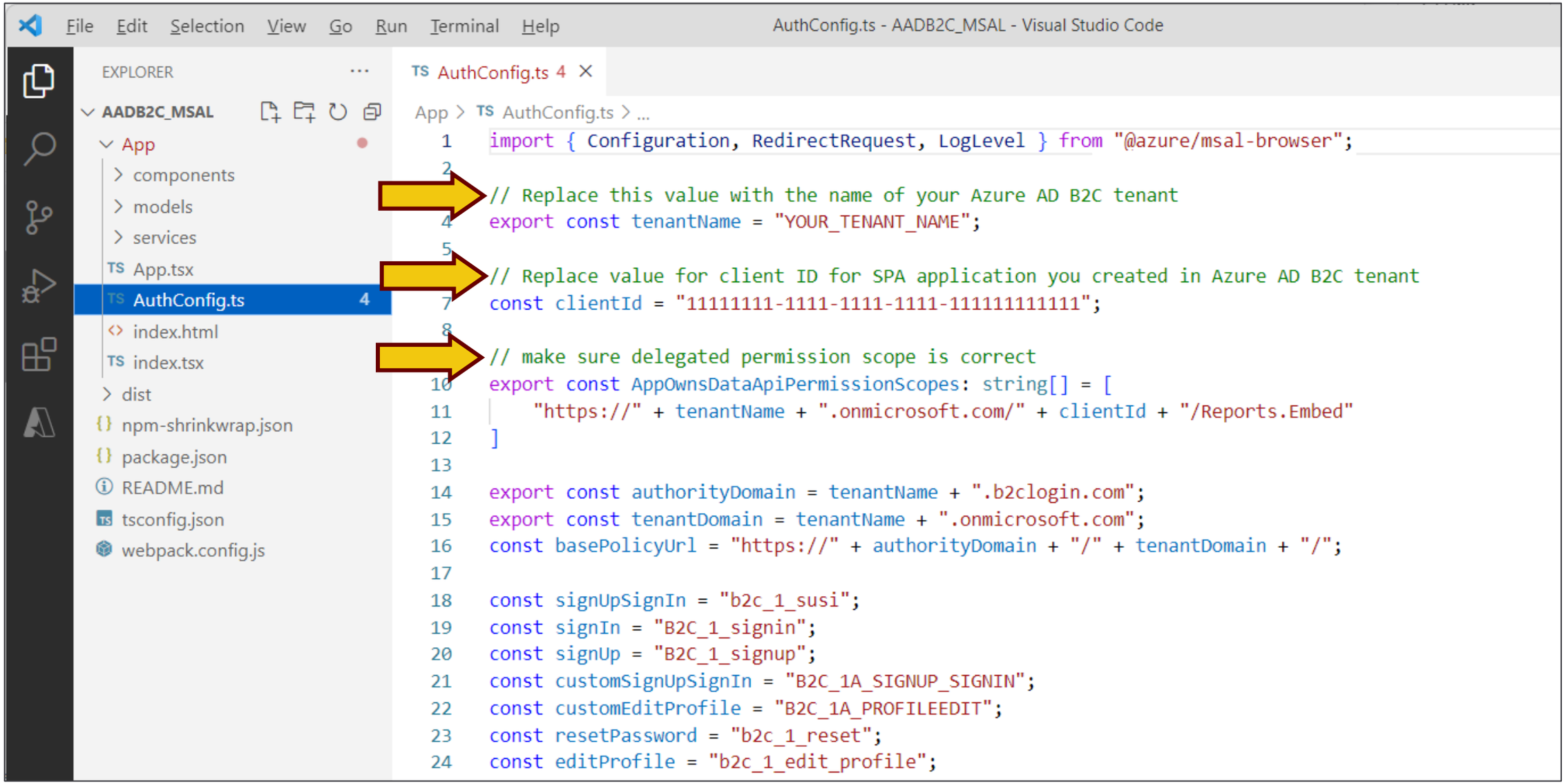


AADB2C_MSAL Sample Application in GitHub

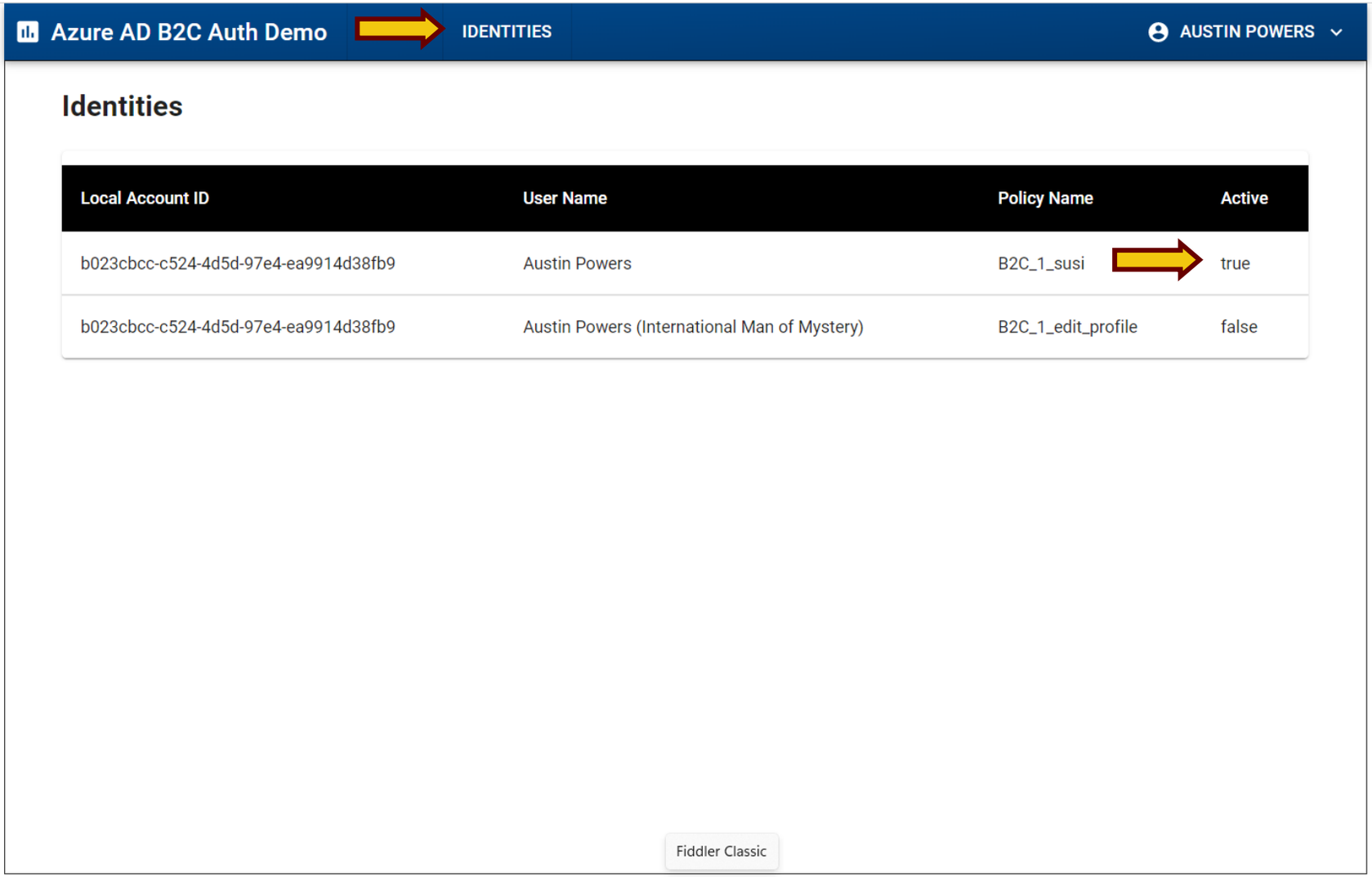
- AADB2C_MSAL demonstrates best practice of authenticating with **msal-browser.js**
  - Download sample code from https://github.com/PowerBiDevCamp/AADB2C_MSAL



Updating AuthConfig.ts for Your Environment



Testing Out the AADB2_MSAL Application

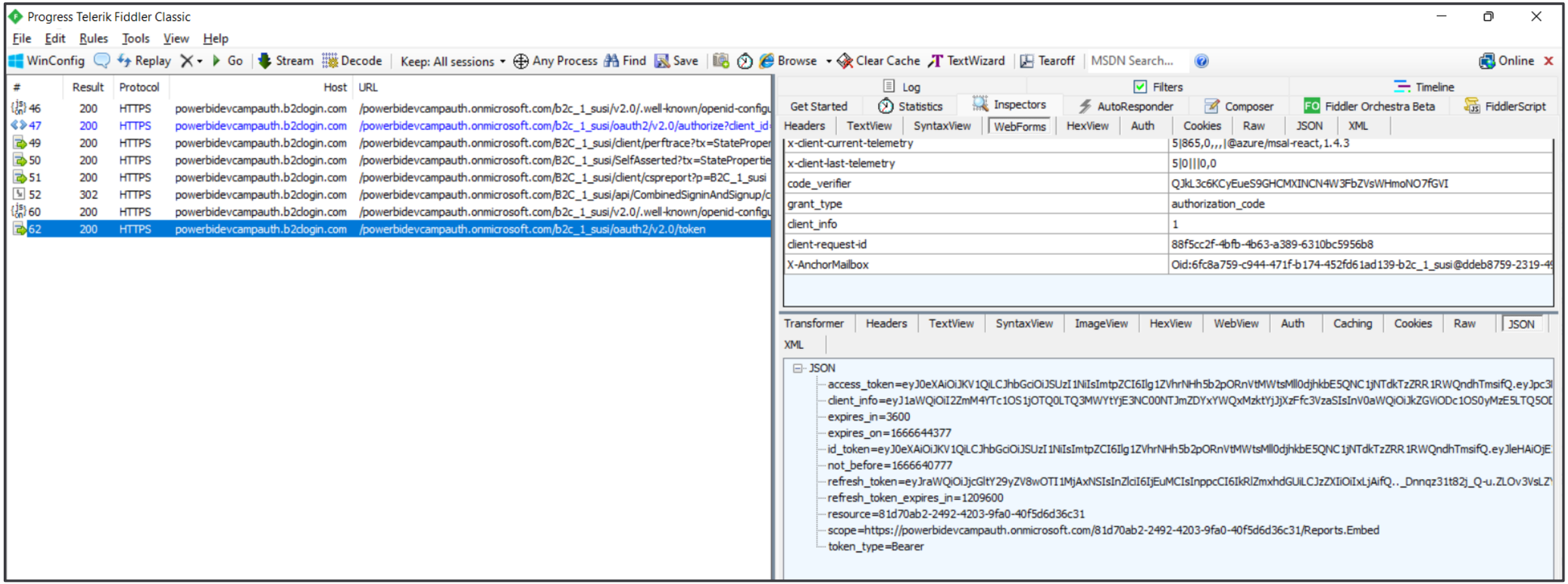


Selecting Signup/Signin Account As the Active Account

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

  // if the 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.signUpSignIn.toUpperCase();
    if (currentAccountPolicy !== requiredAccountPolicy) {
      console.log("WRONG ACCOUNT ACTIVE! - setting active account back to " + b2cPolicies.names.signUpSignIn);
      const filteredAccounts = currentAccounts.filter(account =>
        account.homeAccountId.toUpperCase().includes(b2cPolicies.names.signUpSignIn.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 accountas 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



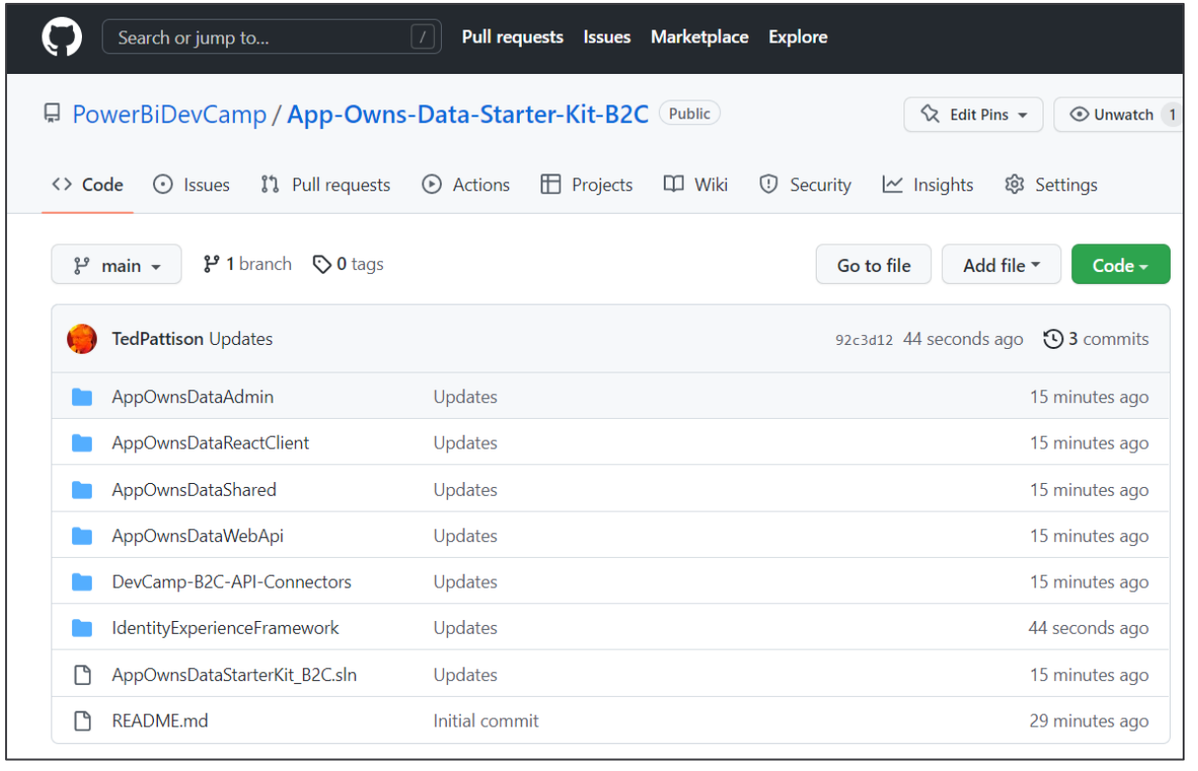
Agenda

- ✓ 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

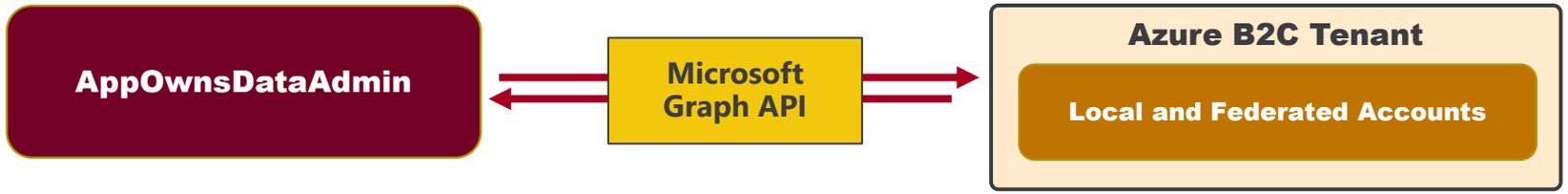
a customized Version of the App-Owns-Data Starter Kit

- 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

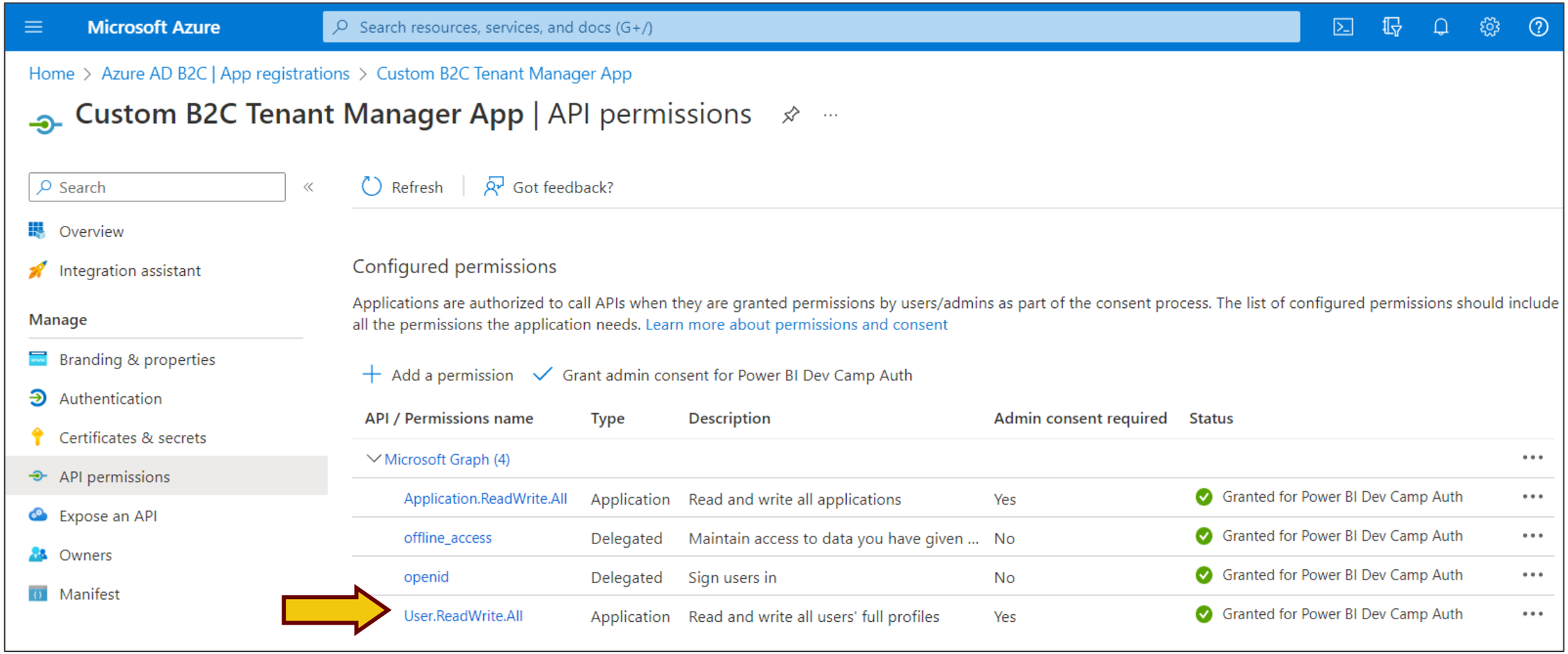
App-Owns-Data Admin | Tenants | Users | B2C User Accounts | Activity Log | Welcome Ted Pattison | Sign out

B2C User Accounts

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**



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

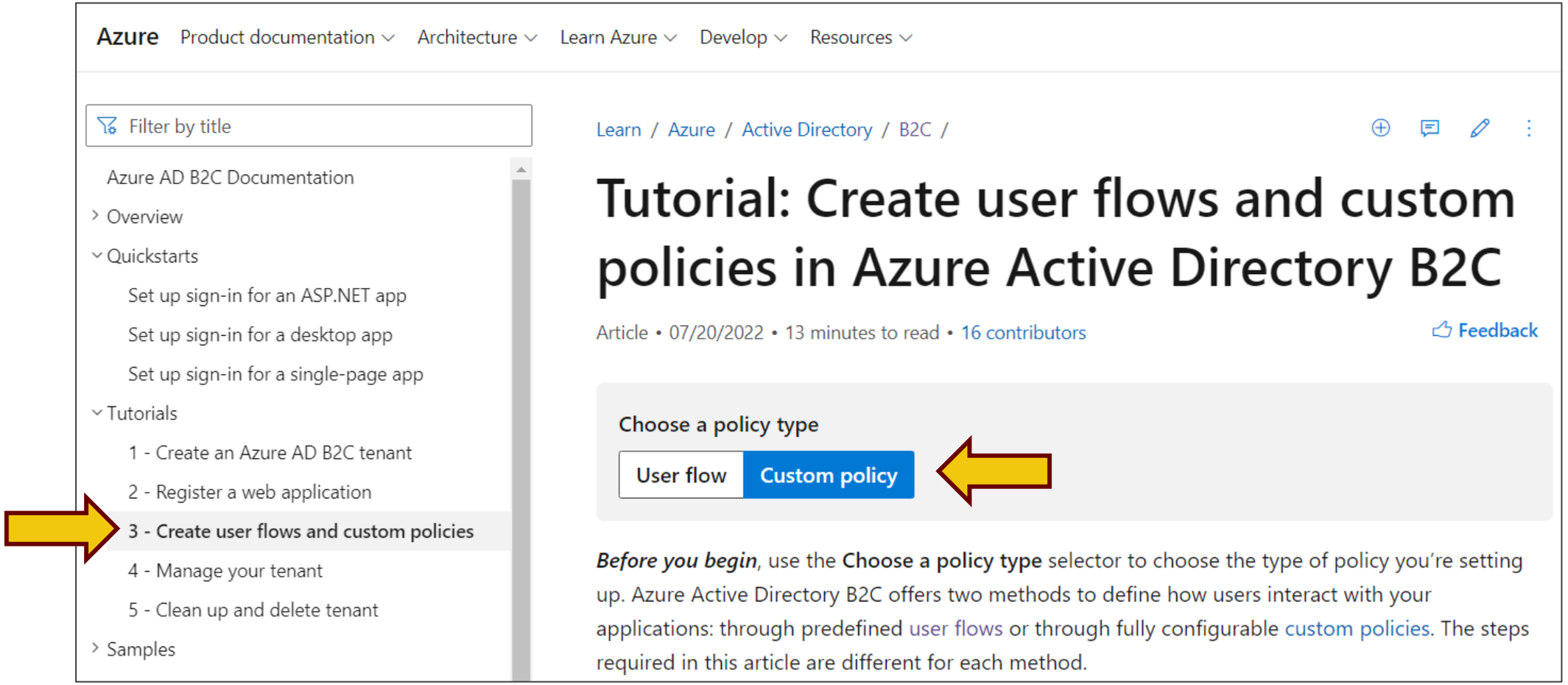
- Custom policy are developed in accordance with **Identity Experience Framework (IEF)**
  - Policies for authentication and token acquisition defined using customized XML files
  - Developer must continually edit and upload XM files to test and debug custom policies

- 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

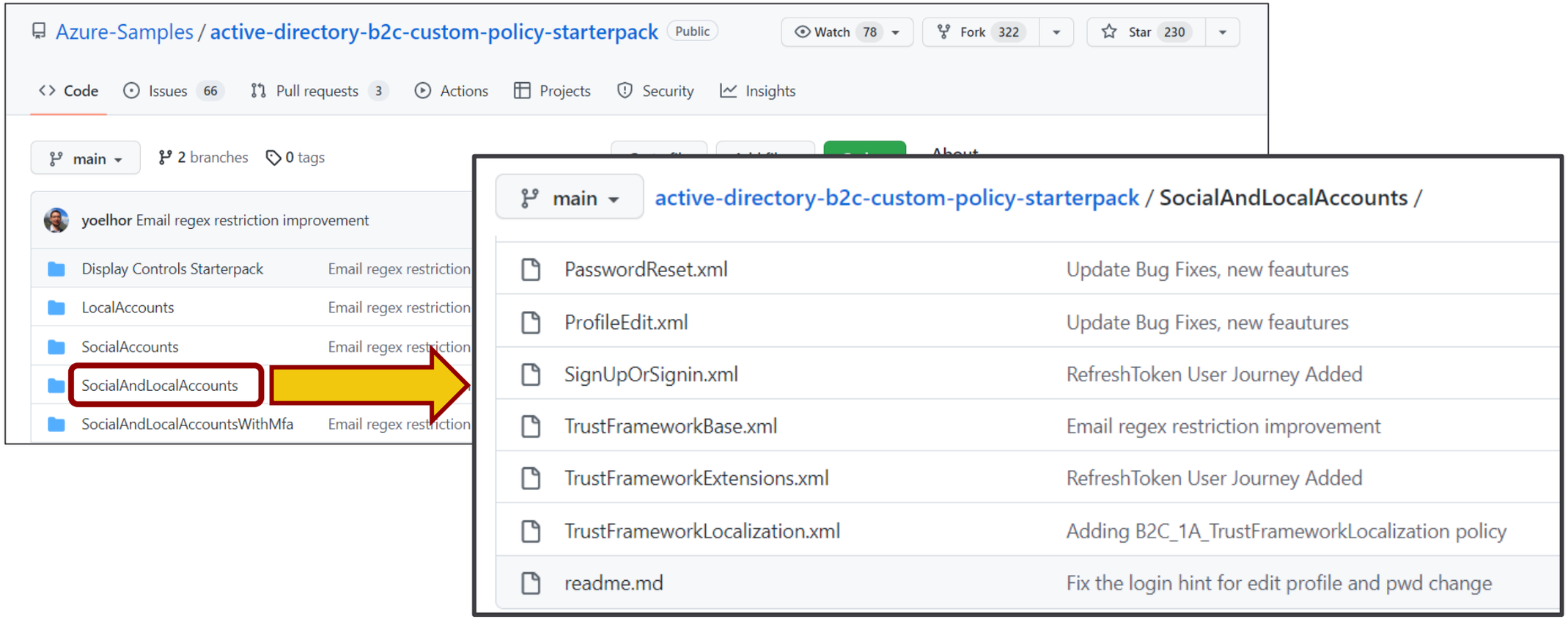
Getting Started with Custom Policies

- 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



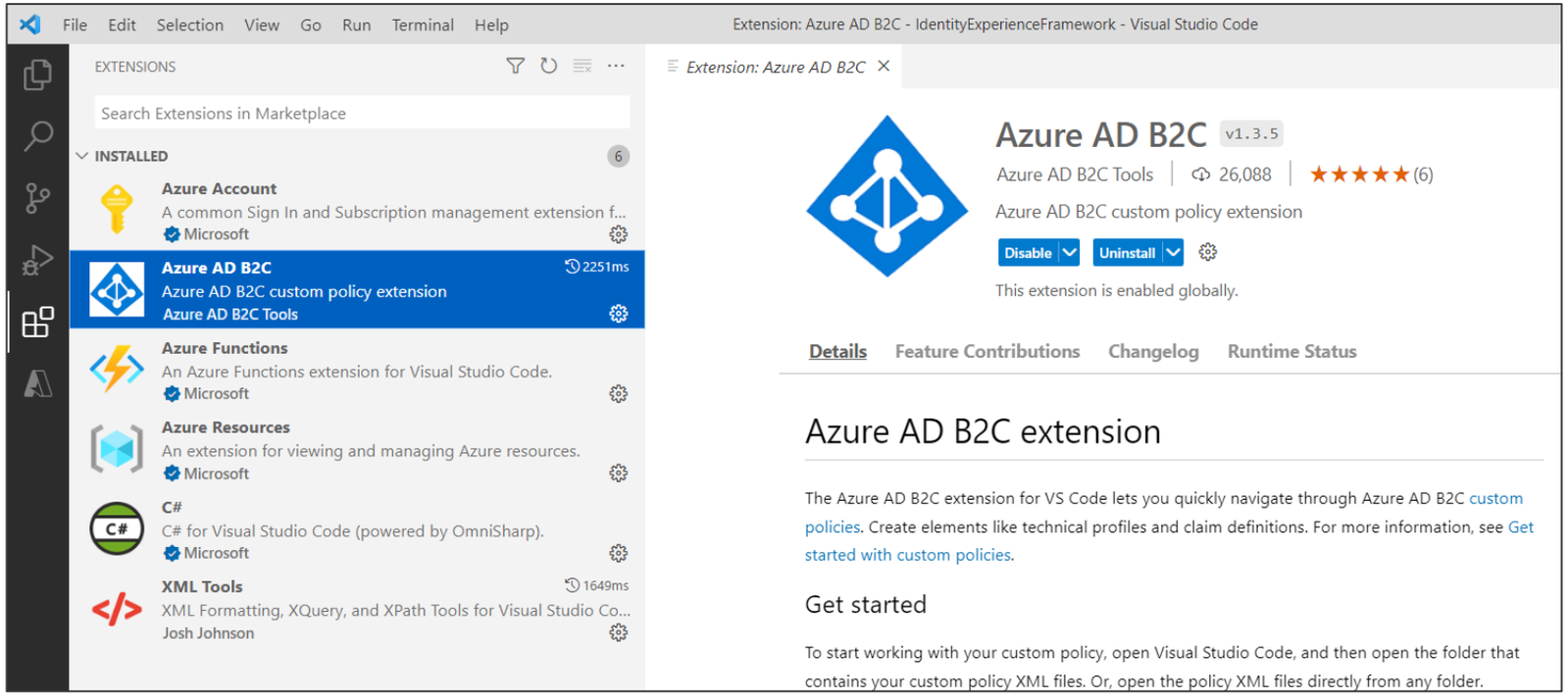
B2C Custom Policy Starter Pack

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

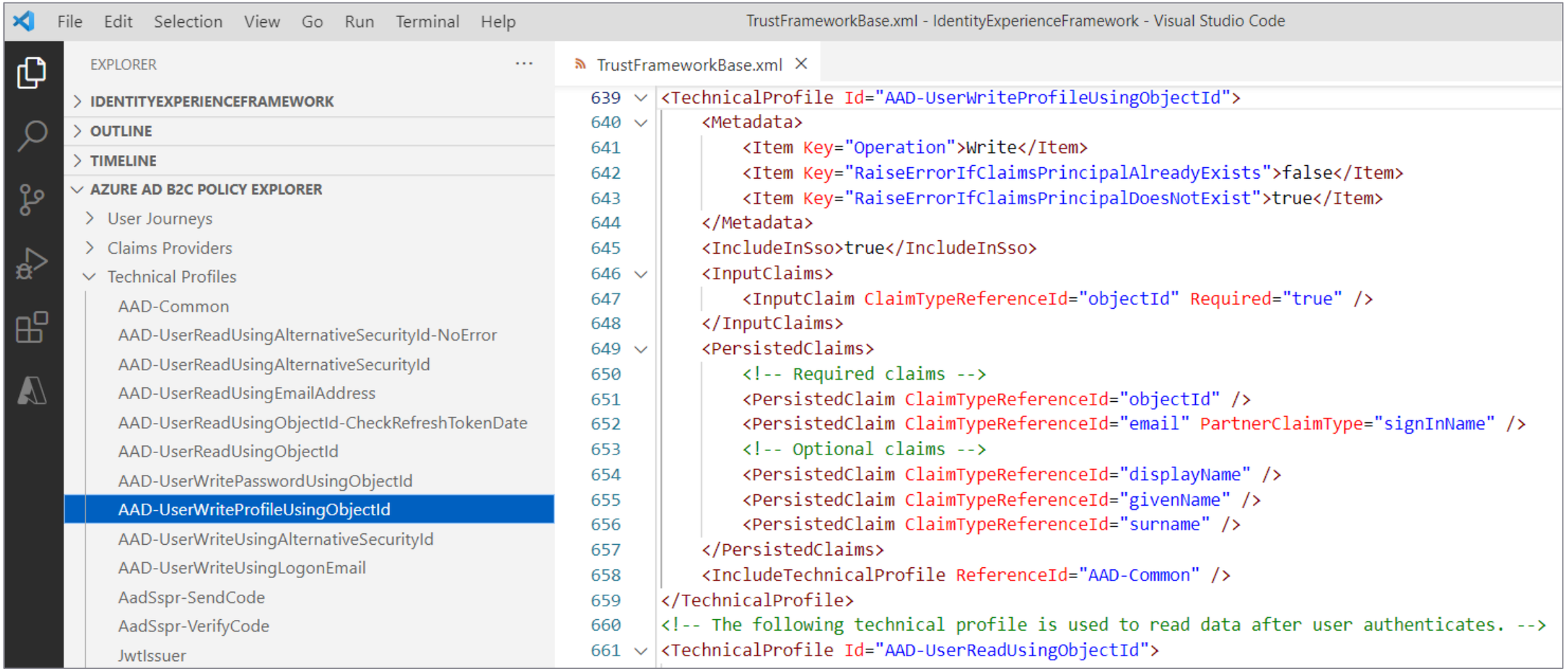


Developing Custom Policies with Visual Studio Code

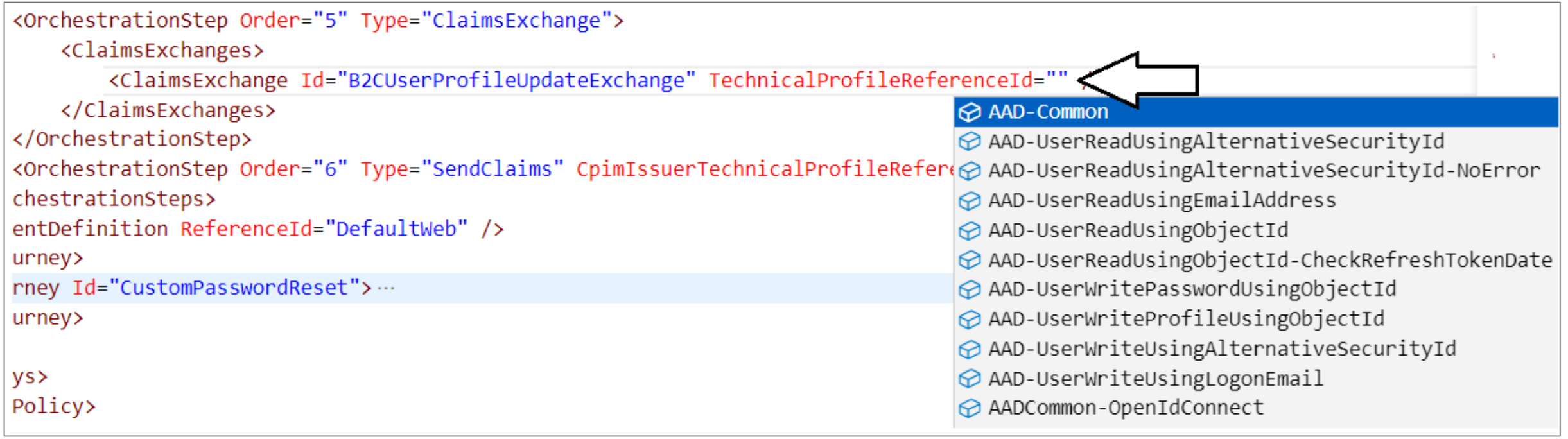
- Use the Azure AD B2C extension for Visual Studio Code
  - Azure AD B2C Policy Explorer
  - IntelliSense with IEF schema
  - Automated XML file upload in VS Code for testing
  - Easy viewing of Azure AD B2C Traces from App Insights



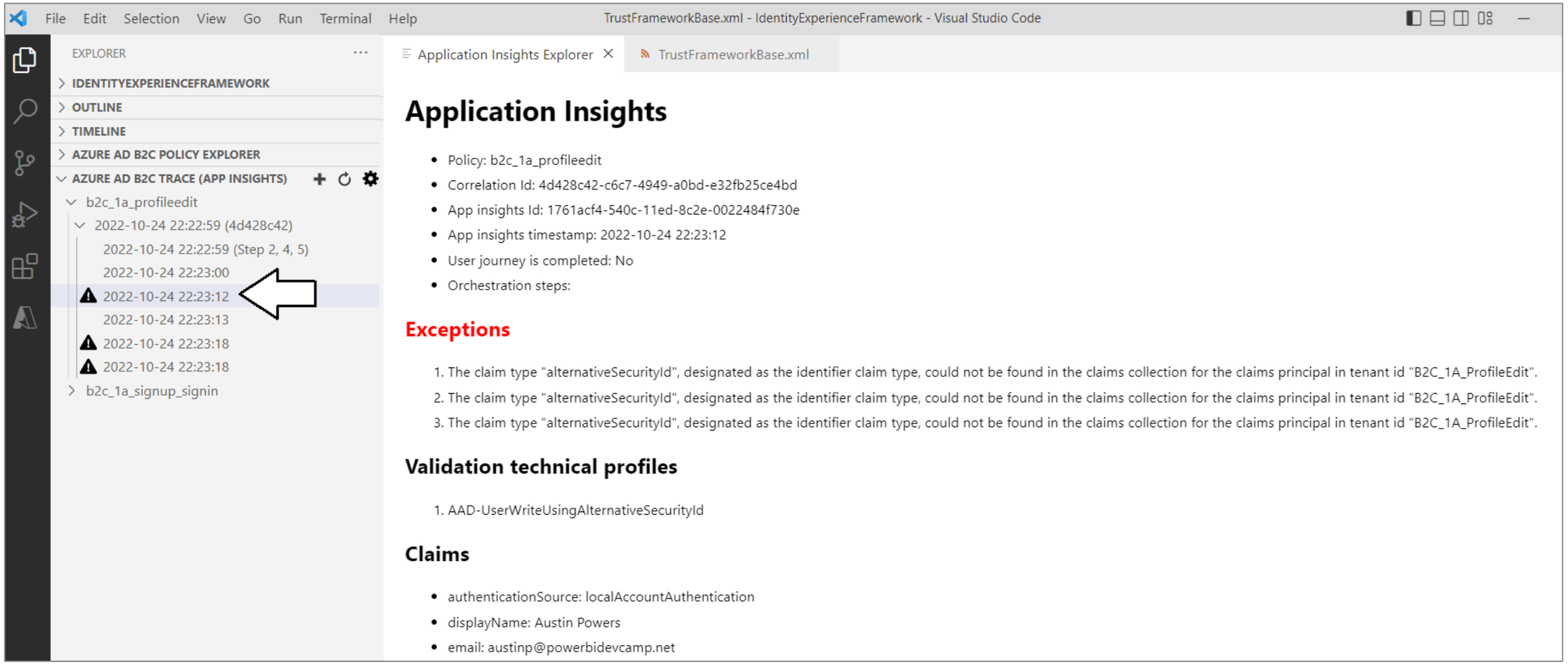
Azure AD B2C Policy Explorer



IntelliSense When Configuring Profile References

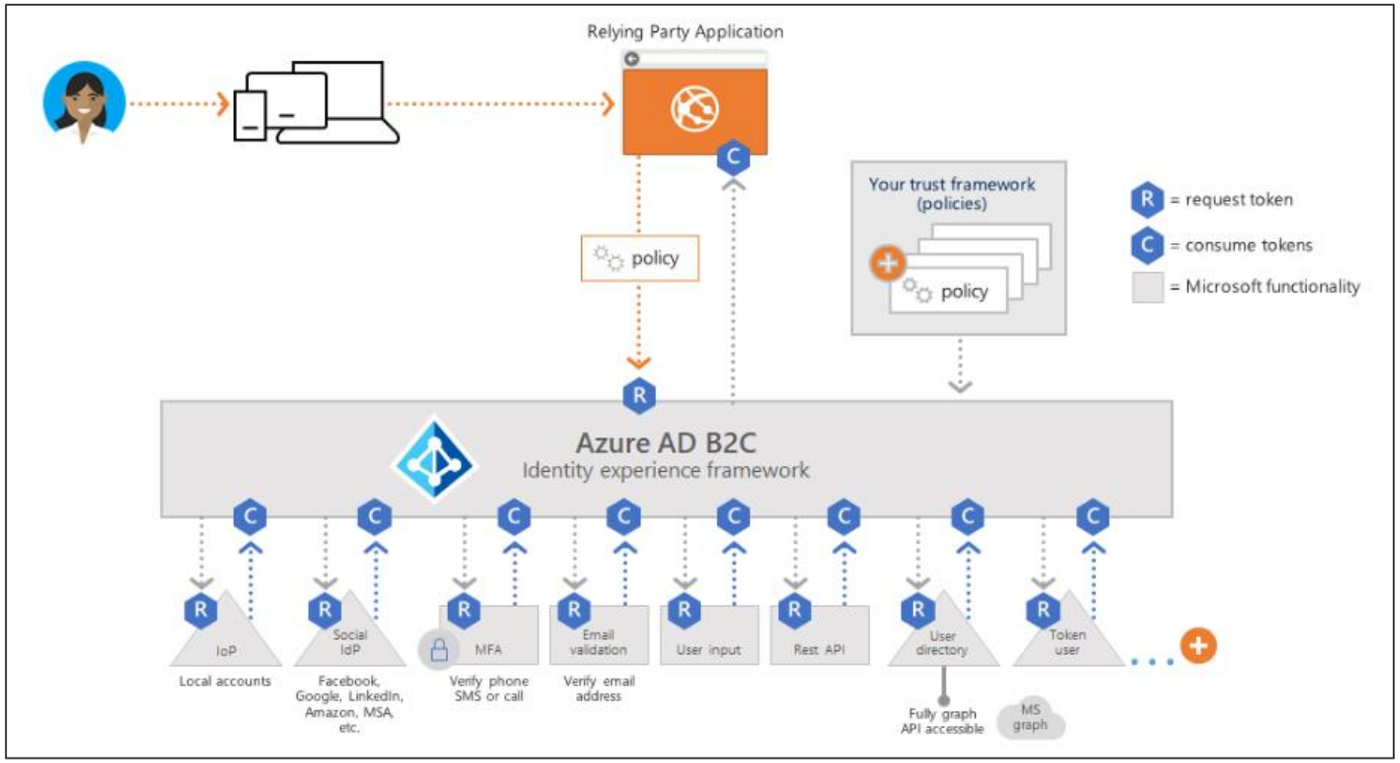


Inspecting Azure AD B2C Traces from App Insights



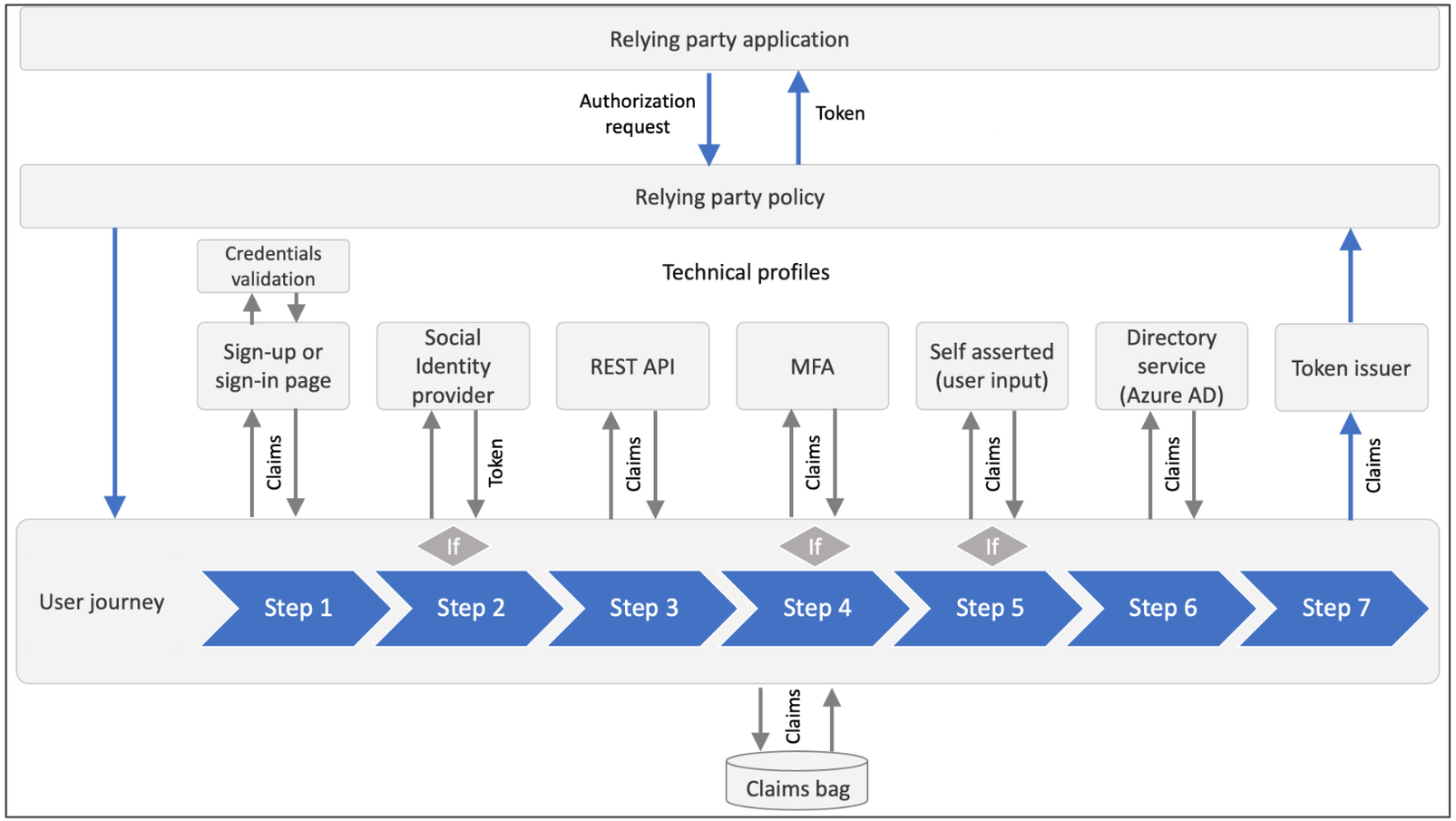
User Journeys and Orchestration Steps

- 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.

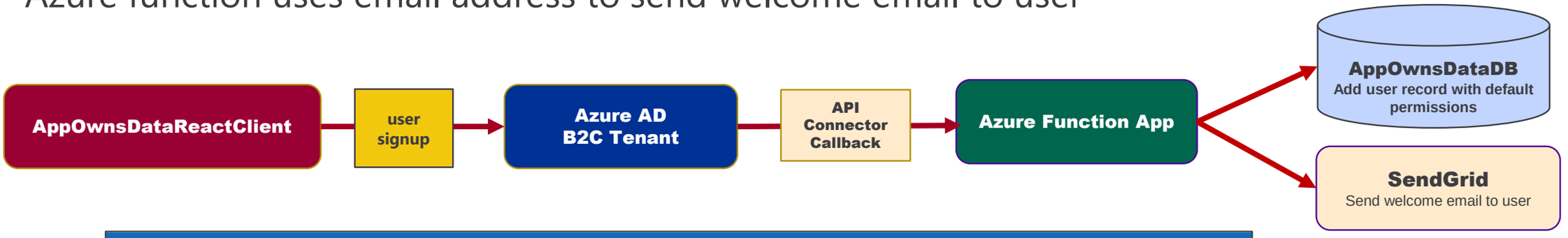


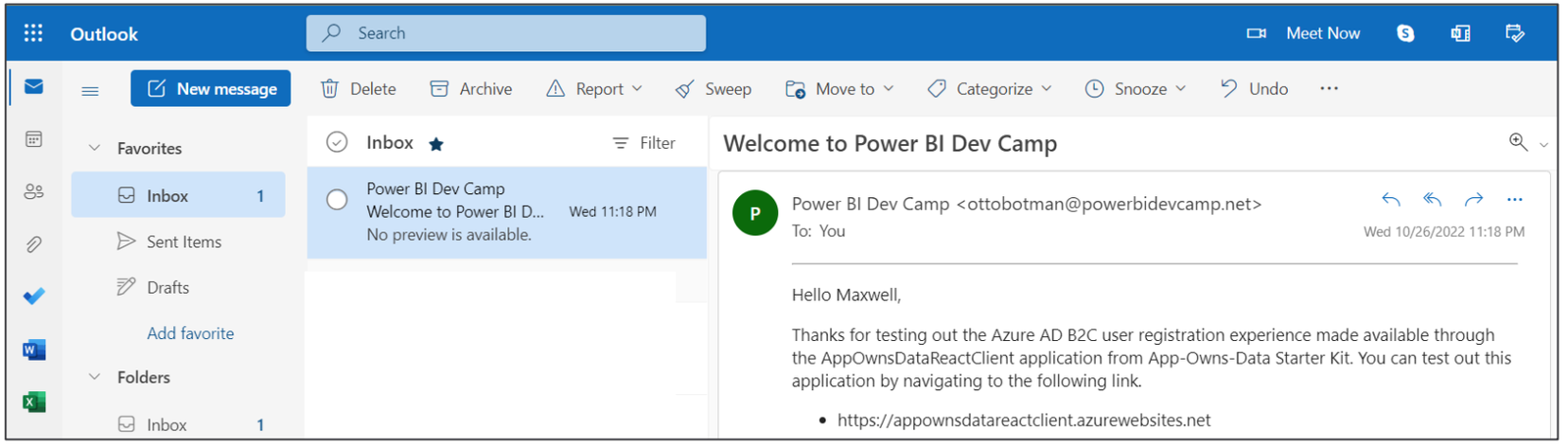
Agenda

- ✓ 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.appOwnsDataDBService.GetUser(email) == null) {

      var user = this.appOwnsDataDBService.CreateUser(new User {
        LoginId = email,
        UserName = displayName,
        TenantName = defaultTenant,
        CanEdit = false,
        CanCreate = false,
        TenantAdmin = false
      });

      this.appOwnsDataDBService.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 creted."));
    }
  }
```

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