IAM Community Call: Azure B2C

Troy Bloesch / Vivek Saxena / Nitin Dev/ Isha Solke

—

February 2024

KPMG

Agenda

1. **What is Azure B2C?**
2. **How Can I Use Azure B2C?**
3. **Lessons Learned**
4. **Possible Extensions**
5. **Client Spotlight**
   a. Description of Service
   b. High-Level View of Output
6. **Deep Dive into the Code**

Microsoft's CIAM Offering

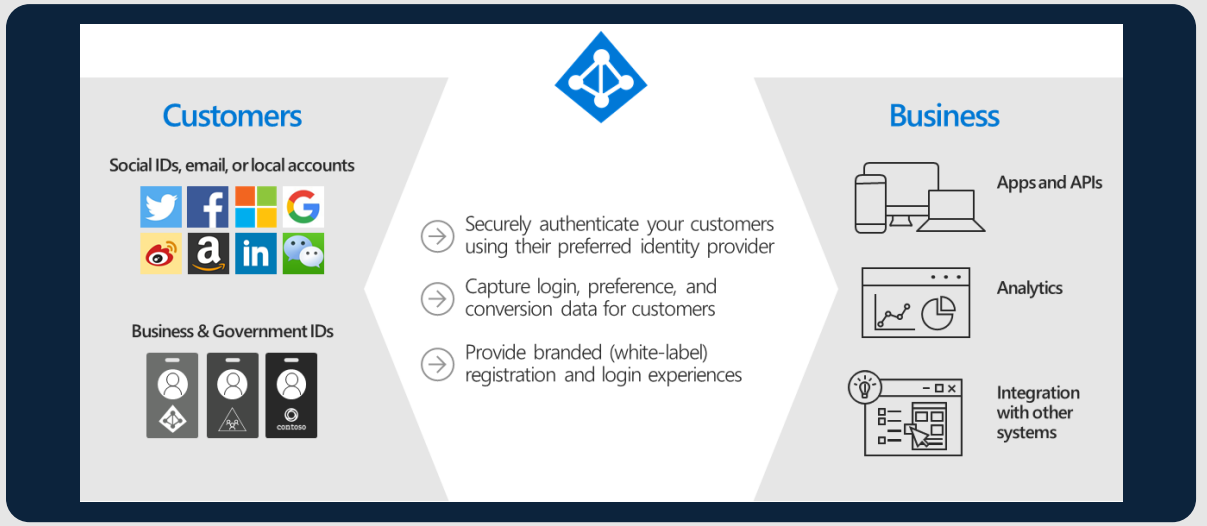
Azure B2C at a Glance

Capable of **supporting millions of users** and billions of authentications per day

Has **out of the box features** to support strong authentication and outsider threats, such as DoS, password spraying, and brute force attacks

Provides **simple instructions for configuration** and integration with several different applications (Appian, Facebook, etc)

Separate service from Azure AD, but built with the same technology (familiar environment)



How to Use Azure B2C

Azure B2C has many capabilities to customize your customers wants

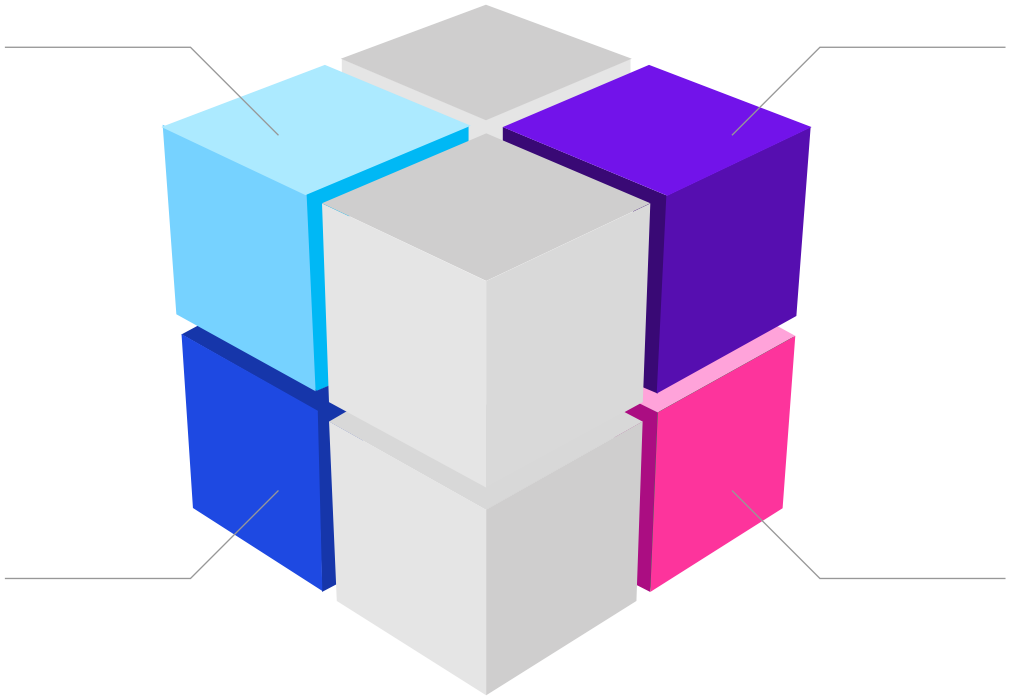
Out of the Box

Users can set up their environment with User Flows for simple websites/experiences for

- Sign In/Sign Up
- Profile Edit
- Password Reset

Identity Experience Framework

For more complex situations, such as setting up a SAML connection with your application, users will be able to use custom policies to create more abilities for their customers



Customized Pages

Not only does B2C allow you to create your own pages, but also allows you to connect to your blob storage to create your own CS/HTML pages to customize the look & feel. These pages can also have their own JavaScript to elevate your customers experience

Helpful Resources

Users are not alone in setting these applications and flows up. There are several custom policies written and tested through GitHub and questions answered through Stack Overflow

Lessons Learned

Lessons Learned

Azure B2C does not have version control history, so we recommend creating copies of working files after testing to serve as a backup in case the code breaks.

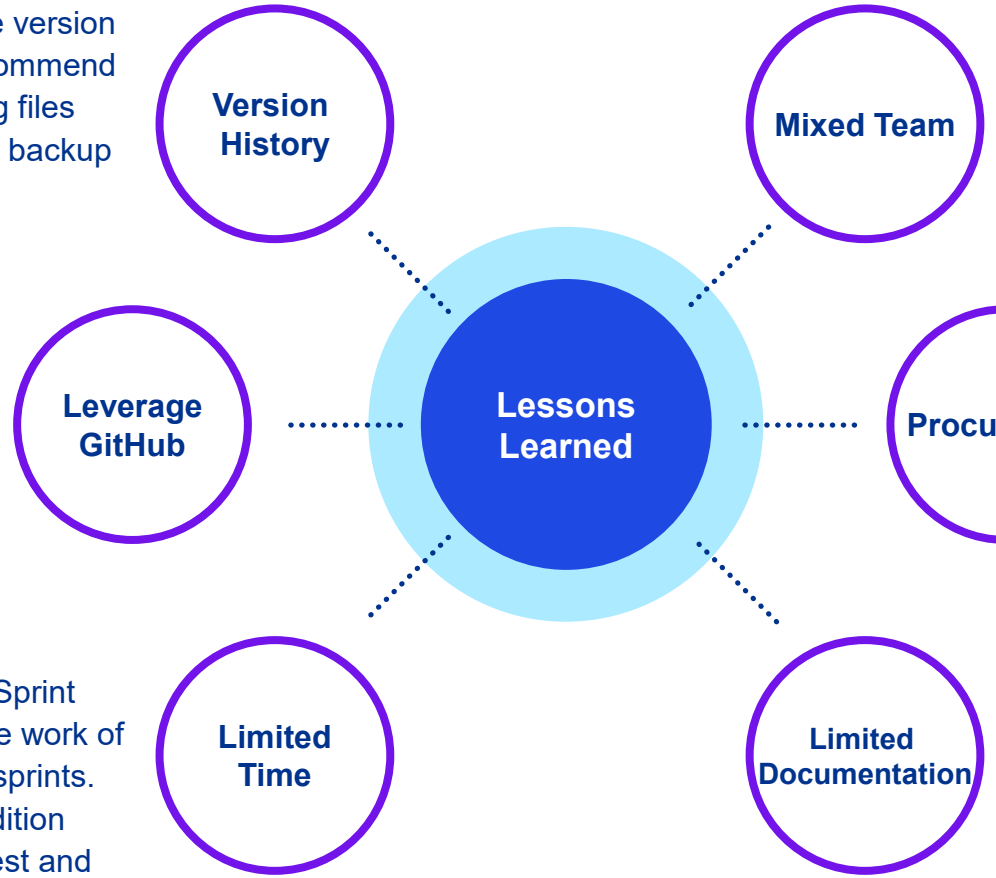
Working across functions opened a lot of discussions with not just the client, but other KPMG teams that have never worked in this domain.

Using the GitHub automation sets you up with the basic files needed to run the simplest flows. From there you can add your own spin to the workflows.

The client had to work through a cloud provider to receive needed resources, which delayed timelines & moved tasks around. Government cloud is much different than commercial cloud when it comes to regulations.

With the project being in Sprint format, we had to start the work of the next sprint in current sprints. This meant each new addition took around a month to test and deploy.

The Azure B2C GitHub documentation was outdated and poorly written, resulting in mixed and self-interpretations for many features. To overcome this, we performed various hands-on trails, backtracking, and extensive team discussions.



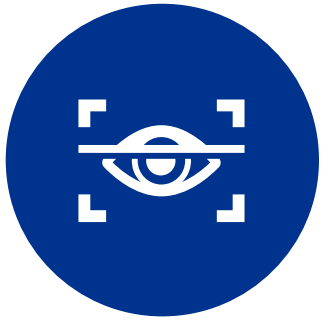
Possible Extensions

Connections & Additions

Below are just a subset of connections that can be added to your B2C environment for further customization

Additional Authentication

Azure B2C allows you to quickly set up you customers with phone/email verification. In addition, B2C provides documentation on connections to be able to incorporate authenticator applications and FIDO approved biometric authentication



Visual Studio Extensions

Visual Studio has a Azure B2C extension that allows you to quickly navigate the files & sections of your code. It even has the ability to deploy the code directly from Visual Studio

Azure Functions

Through a REST technical profile, Azure B2C is able to call on Azure Functions to be able to use all its functionalities

Much more

Azure B2C can integrate with many other companies and technologies. Below are just a few examples

Experian	Idology	Twilio
Jumio	Login.gov	TypingDNA
Nevis	ThreatMetrix	Cloudflare

Client Spotlight

Azure B2C was used to allow the Secretary of State to document and securely allow citizens to file for trademarks and other business services.

KPMG & TXSoS

KPMG has provided a solution that will allow Texas SoS to better record users who have signed up to apply for trademarks. Along with that the azure team has also kept security of your infrastructure as well as your users in mind when designing the MVP

Simple

The Azure B2C solution that keeps track of users and passes necessary data to Appian saves the user and the business time by discarding the use of hand written or faxed applications

Secure

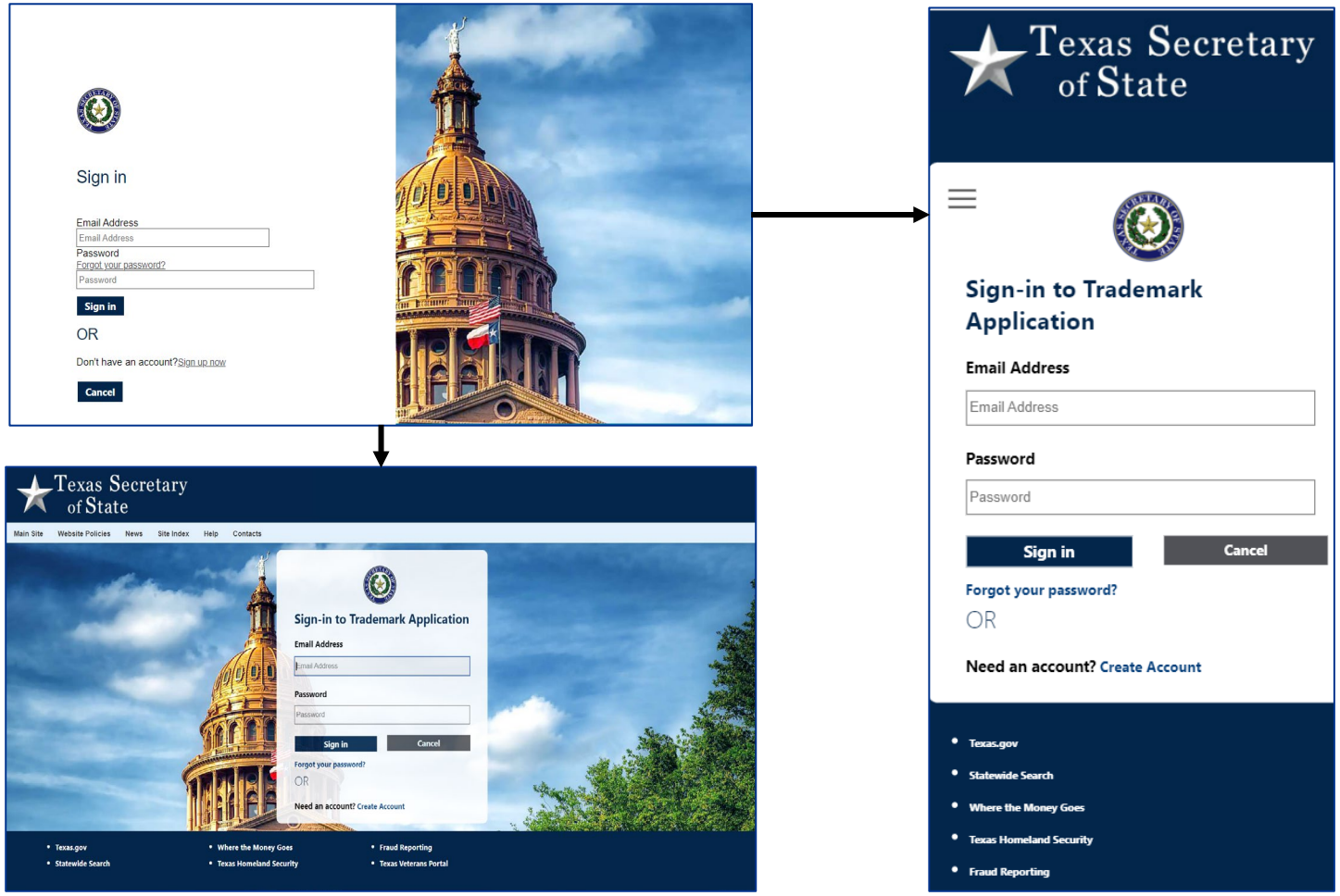
With anything going on the internet, security becomes a big priority. Attackers are always trying to find ways to get your information. That is why we built the solution with security in mind (MFA and MFA timeouts, account lockout. Etc)

Swift

Before the solution was implemented, registration and approval could take several hours, even days. Now with the quick and speedy sign-up and simple application process, applications can be reviewed instantly when submitted

Sprint Summary at Texas SoS

HTML/CSS Updates



In this story, KPMG was able to:

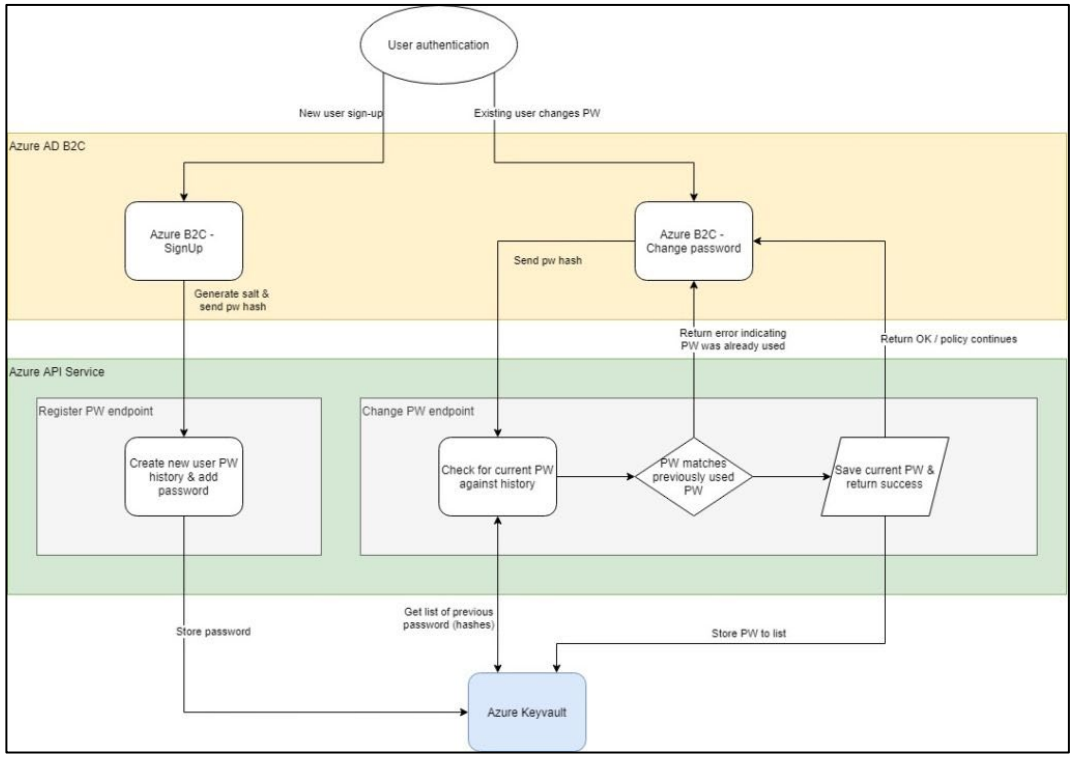
- ✓Redesign the entire user interface.
- ✓Separation of Desktop and Mobile view.
- ✓Utilized **AdvisoryGPT** to implement additional HTML/CSS features for a quick turnaround time.
- ✓Both desktop and mobile views went through accessibility reviews.

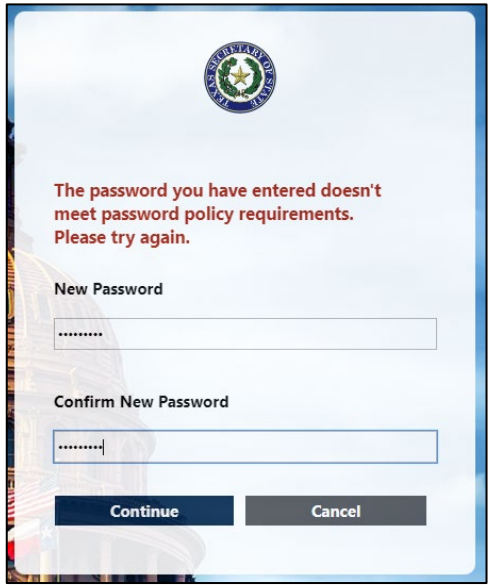
Split Sign-Up

In this story, KPMG was able to:

- ✓Split up the sign-up process, where it first verifies the user's email and then gradually presents the user with additional fields to create their account.

Password History



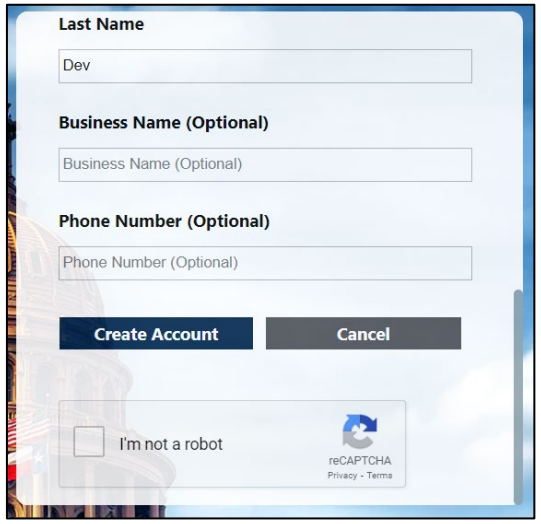


In this story, KPMG was able to:

- ✓Implement password history to retain a user's last 10 passwords.
- ✓This check is done during Password Change and Forgot Password.
- ✓This was accomplished using Azure Web App and Azure Key Vault.
- ✓The Web App is responsible for taking the user's credentials, hashing them, and sending them to Key Vault.
- ✓The Key Vault is responsible for using the hashed values to check if the user is attempting to use one of their last 10 passwords.

Azure Functions - reCAPTCHA

```
[Function("reCAPTCHA_HTTP_Trigger")]
    public static async Task<HttpResponseMessage> Run([HttpTrigger(AuthorizationLevel.Function, "get", "post")] HttpRequestData req)
    {

        if ( string.IsNullOrEmpty(captcha) ) {
            var respContent = new { version = "1.0.0", status = (int)HttpStatusCode.BadRequest, userMessage = "Invalid Captcha"};
            var json = JsonConvert.SerializeObject(respContent);
            return new HttpResponseMessage(HttpStatusCode.Conflict) {
            Content = new StringContent(json, System.Text.Encoding.UTF8, "application/json")
            };
        }

        // store your Captcha secret key in thge app config
        var captchaSecret = System.Environment.GetEnvironmentVariable("CAPTCHA_SECRET_KEY");

        HttpClient client = new HttpClient();
        var url = $"https://www.google.com/recaptcha/api/siteverify";
        var dict= new Dictionary<string, string>();
        dict.Add("secret", captchaSecret);
        dict.Add("response", captcha);

        HttpResponseMessage res = client.PostAsync(url, new FormUrlEncodedContent(dict)).Result;
        var contents = await res.Content.ReadAsStringAsync();
        client.Dispose();

        // return either good message that REST API expects or 409 conflict
        if ( res.StatusCode != HttpStatusCode.OK ) {
            var respContent = new { version = "1.0.0", status = (int)HttpStatusCode.BadRequest, userMessage = "Captcha API failed"};
            var json = JsonConvert.SerializeObject(respContent);
            return new HttpResponseMessage(HttpStatusCode.Conflict) {
            Content = new StringContent(json, System.Text.Encoding.UTF8, "application/json")
            };
        }

        JObject obj = JObject.Parse(contents);
        bool success = (bool)obj["success"];
        if ( success ) {
            return new HttpResponseMessage(HttpStatusCode.OK) {
                Content = new StringContent(contents, System.Text.Encoding.UTF8, "application/json")
```



In this story, KPMG was able to :

- ✓ Implement reCAPTCHA on user sign-up to prevent bots from creating mass junk accounts.
- ✓ Accomplished this using Azure functions to send the private and public reCAPTCHA tokens to Google's reCAPTCHA API for verification.

Azure Functions and ACS – Welcome Email

```
[Function("ACS_HTTP_Trigger")]
public async Task<HttpResponseData> RunAsync([HttpTrigger(AuthorizationLevel.Function, "get", "post")] HttpRequestData req)
{
    _logger.LogInformation("C# HTTP trigger function processed a request.");
    var response = req.CreateResponse(HttpStatusCode.OK);
    response.Headers.Add("Content-Type", "text/plain; charset=utf-8");
    response.WriteString("Email Sent Successfully.");
    string userEmailAddress = req.Query["readOnlyEmail"];
    _logger.LogInformation($"Email is ={userEmailAddress}");
    var connectionString = System.Environment.GetEnvironmentVariable("ACS_ACCESS_KEY");
    var emailClient = new EmailClient(connectionString);

    var sender = "DoNotReply@txsoscom.sos.texas.gov";
    var recipient = userEmailAddress;
    var subject = "Welcome to Your Trademarks Account";
    string url = "https://txsosstorage01.blob.core.windows.net/txsosb2cprod/Welcome_Email_Content.html";

    using (HttpClient client = new HttpClient()){
    using (HttpResponseMessage responseACS = client.GetAsync(url).Result){
        using (HttpContent content = responseACS.Content){
            string htmlContent = content.ReadAsStringAsync().Result;
            try{
                var emailSendOperation = await emailClient.SendAsync(
                    wait: WaitUntil.Completed,
                    senderAddress: sender,
                    recipientAddress: recipient,
                    subject: subject,
                    htmlContent: htmlContent);
                Console.WriteLine($"Email Sent. Status = {emailSendOperation.Value.Status}");

                string operationId = emailSendOperation.Id;
                Console.WriteLine($"Email operation id = {operationId}");
            }
            catch (RequestFailedException ex){
                Console.WriteLine($"Email send operation failed with error code: {ex.ErrorCode}, message: {ex.Message}");
            }

            return response;
        }
    }
    }
```

In this story, KPMG was able to:

- ✓Implement a Welcome Email to be sent to successful new account creations.
- ✓This was done using Azure Communication Services (ACS) and Azure Functions.
- ✓Azure Functions served as the HTTP trigger to call ACS and populate the recipient's email address field.
- ✓ACS was responsible for sending out the email to the customer.
- ✓The combination of these Azure services helped the client save thousands of dollars as a third-party email service wasn't required.

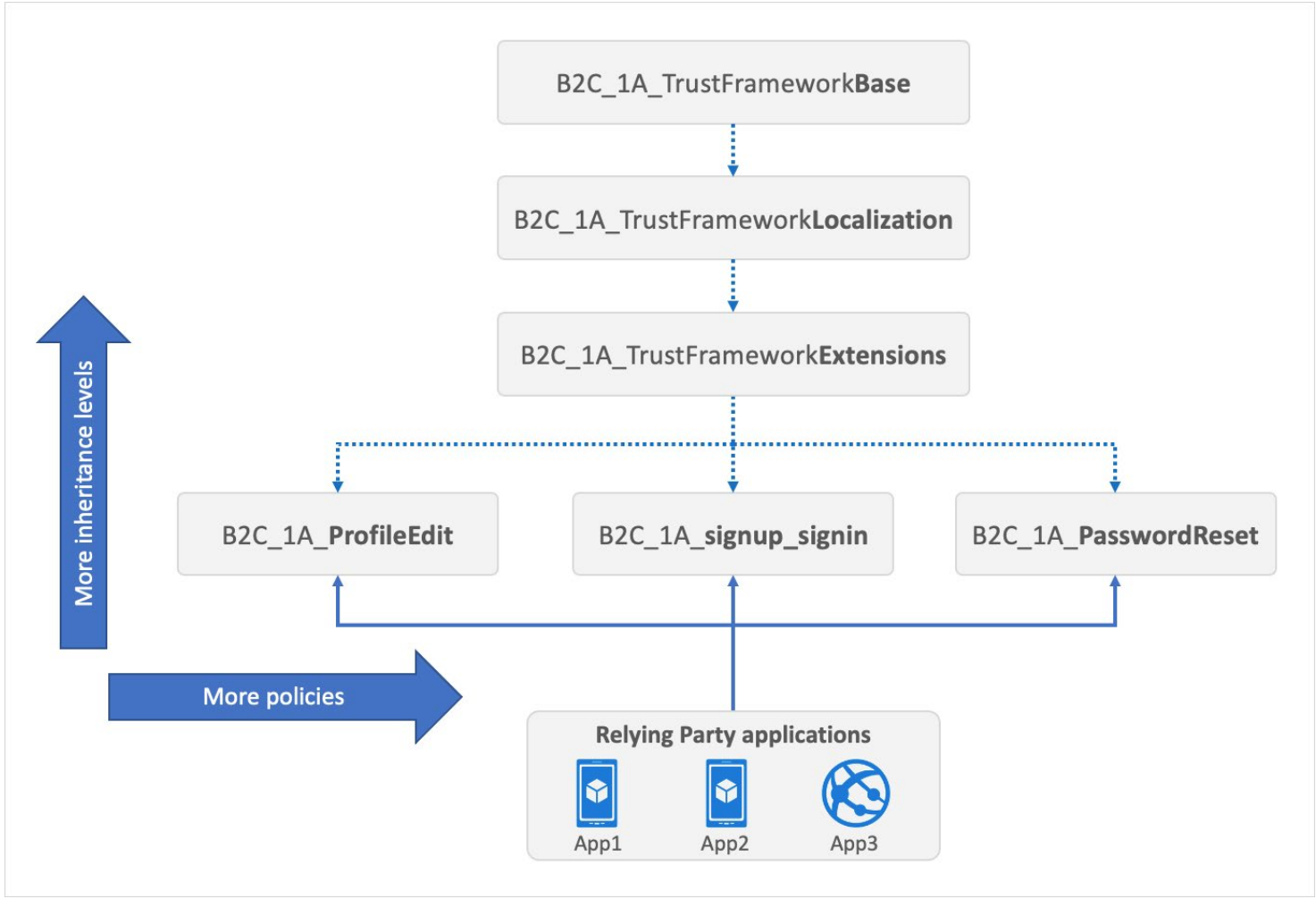
Under the Hood

of Split Sign-Up/Sign-in

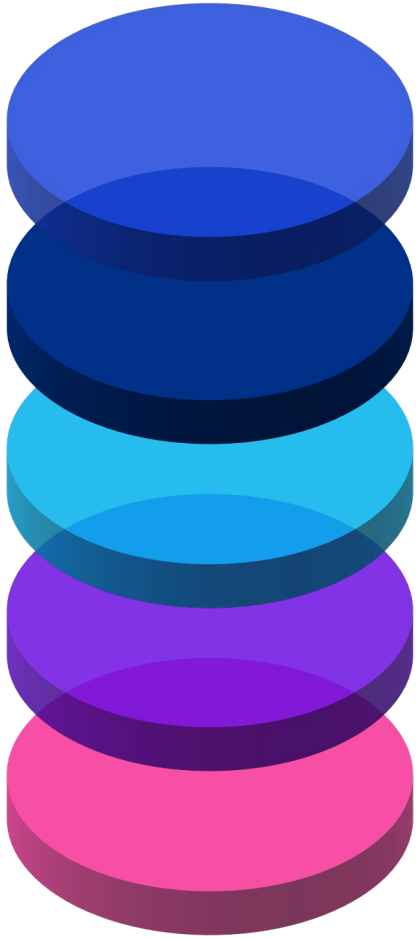
Hierarchy

Azure B2C Custom Policies are set up in this hierarchical pattern with the Base being the policy that overrules all the rest of the policies.

Other extension files can reference extension just as long as the last extension references localization and the localization references base.



Breakdown of Claims



01

<ContentDefinitions>

Contains the settings that control the appearance of web pages presented to users during the user journey.

02

<ClaimsTransformations>

Describes the transformations applied to claims during the policy execution.

03

<ClaimsSchema>

Defines the custom claim types used in the policy.

04

<ClaimsProviders>

Defines the interaction between Azure AD B2C and various identity providers like Email Verification and Local Account. It has <TechnicalProfiles> that describe the behavior and protocol for each provider.

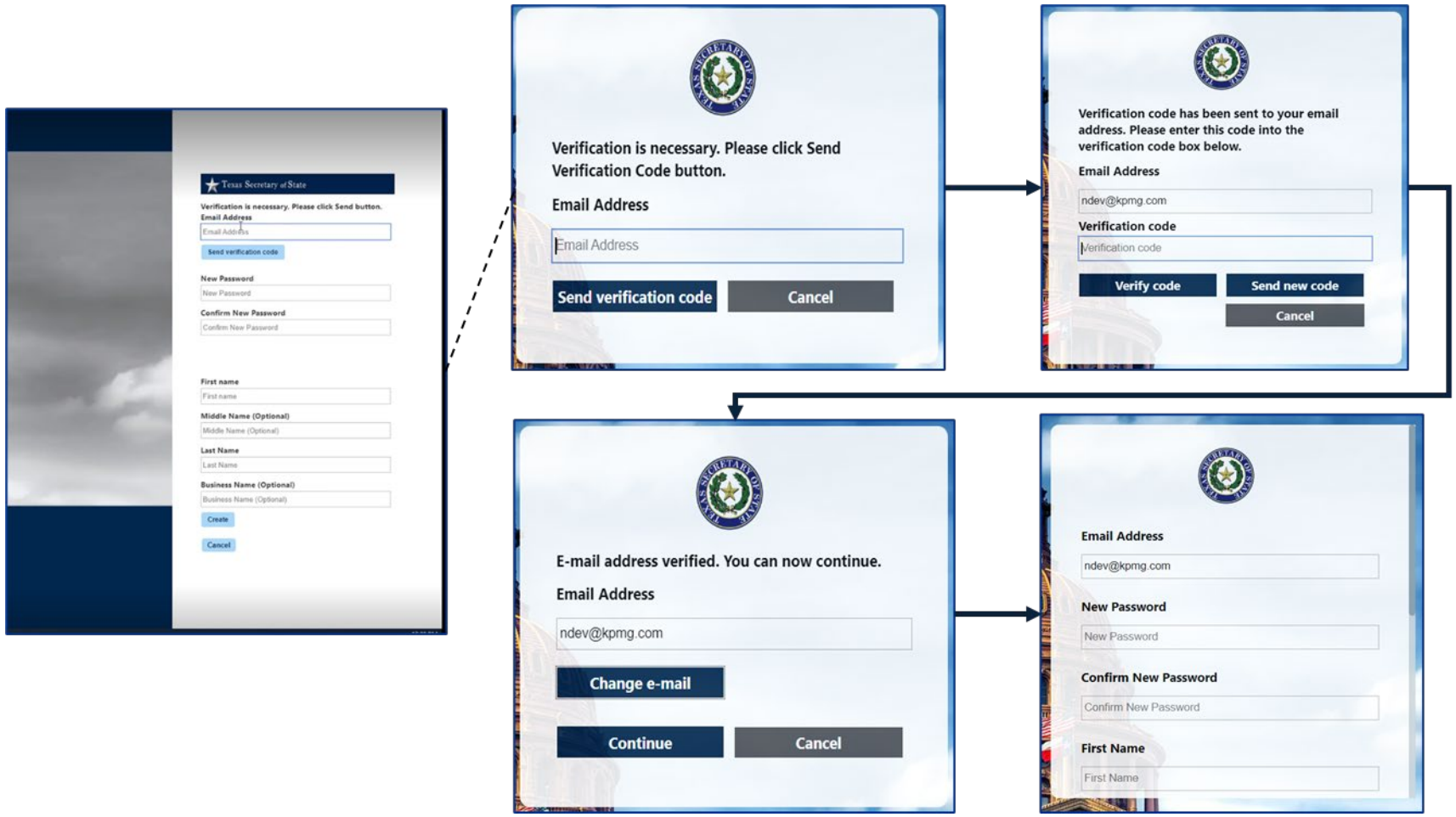
05

<UserJourneys>

Describes the sequence of steps users will go through during the authentication process. It contains <OrchestrationSteps> and <ClaimsExchanges> to represent the user journey's progress and claim exchanges between various entities.

The End Result

samples/policies/split-email-verification-and-signup/policy/SignUpOrSignIn_SplitEmailVerificationAndSignUp.xml at master · azure-ad-b2c/samples · GitHub



Building Blocks

The BuildingBlock in the Signup flow shows the User's read-only email address, which is used to prefill the email claim, helping in the email verification process.

```
11      <BuildingBlocks>
12        <ClaimsSchema>
13          <!-- Read only email address to present to the user-->
14          <ClaimType Id="readonlyEmail">
15            <DisplayName>E-mail Address</DisplayName>
16            <DataType>string</DataType>
17            <UserInputType>Readonly</UserInputType>
18          </ClaimType>
19        </ClaimsSchema>
20        <ClaimsTransformations>
21          <ClaimsTransformation Id="CreateReadonlyEmailClaim" TransformationMethod="FormatStringClaim">
22            <InputClaims>
23              <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="inputClaim" />
24            </InputClaims>
25            <InputParameters>
26              <InputParameter Id="stringFormat" DataType="string" Value="{0}" />
27            </InputParameters>
28            <OutputClaims>
29              <OutputClaim ClaimTypeReferenceId="readonlyEmail" TransformationClaimType="outputClaim" />
30            </OutputClaims>
31          </ClaimsTransformation>
32        </ClaimsTransformations>
```

Content Definition

The ContentDefinition code references a theme template to control the appearance of the web page that is presented to the User when signing up.

```
34        <ContentDefinitions>
35          <ContentDefinition Id="api.localaccount.emailVerification">
36            <LoadUri>~/tenant/templates/AzureBlue/selfAsserted.cshtml</LoadUri>
37            <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
38            <DataUri>urn:com:microsoft:aad:b2c:elements:contract:selfasserted:2.1.8</DataUri>
39            <Metadata>
40              <Item Key="DisplayName">Collect information from user page</Item>
41            </Metadata>
42            <LocalizedResourcesReferences MergeBehavior="Prepend">
43              <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.localaccount.emailVerification.en" />
44            </LocalizedResourcesReferences>
45          </ContentDefinition>
46        </ContentDefinitions>
```

ClaimsProvider #1

This ClaimsProvider code is responsible for Email Verification. The "Email Verification" technical profile requires users to provide their email address in the email claim and sends them a code to enter in a designated field. After entering the correct code, the email claim gets verified, allowing the user to continue.

```
60      <ClaimsProviders>
61        <ClaimsProvider>
62          <DisplayName>Email Verification</DisplayName>
63          <TechnicalProfiles>
64            <!--Email verification only-->
65            <TechnicalProfile Id="EmailVerification">
66              <DisplayName>Initiate Email Address Verification For Local Account</DisplayName>
67              <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
68              <Metadata>
69                <Item Key="ContentDefinitionReferenceId">api.localaccount.emailVerification</Item>
70                <Item Key="language.button_continue">Continue</Item>
71              </Metadata>
72              <OutputClaims>
73                <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="Verified.Email" Required="true" />
74              </OutputClaims>
75            </TechnicalProfile>
76          </TechnicalProfiles>
77        </ClaimsProvider>
```

ClaimsProvider #2

This ClaimsProvider code is responsible for creating a local account with the email provided from the previous claim. After email verification, the user is prompted to enter their password, name, and other personal information.

```
78        <ClaimsProvider>
79          <DisplayName>Local Account</DisplayName>
80          <TechnicalProfiles>
81            <!--Sign-up self-asserted technical profile without Email verification-->
82            <TechnicalProfile Id="LocalAccountSignUpWithReadOnlyEmail">
83              <DisplayName>Email signup</DisplayName>
84              <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
85              <Metadata>
86                <Item Key="IpAddressClaimReferenceId">IpAddress</Item>
87                <Item Key="ContentDefinitionReferenceId">api.localaccountsignup</Item>
88                <Item Key="language.button_continue">Create</Item>
89                <!-- Remove sign-up email verification -->
90                <Item Key="EnforceEmailVerification">False</Item>
91              </Metadata>
92              <InputClaimsTransformations>
93                <InputClaimsTransformation ReferenceId="CreateReadonlyEmailClaim" />
94              </InputClaimsTransformations>
95              <InputClaims>
96                <!--Sample: Set input the ReadOnlyEmail claim type to prefilled the email address-->
97                <InputClaim ClaimTypeReferenceId="readOnlyEmail" />
98              </InputClaims>
99              <OutputClaims>
100               <OutputClaim ClaimTypeReferenceId="objectId" />
101               <!-- Sample: Display the ReadOnlyEmail claim type (instead of email claim type)-->
102               <OutputClaim ClaimTypeReferenceId="readOnlyEmail" Required="true" />
103               <OutputClaim ClaimTypeReferenceId="newPassword" Required="true" />
104               <OutputClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
105               <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
106               <OutputClaim ClaimTypeReferenceId="authenticationSource" />
107               <OutputClaim ClaimTypeReferenceId="newUser" />
108               <!-- Optional claims, to be collected from the user -->
109               <OutputClaim ClaimTypeReferenceId="displayName" />
110               <OutputClaim ClaimTypeReferenceId="givenName" />
111               <OutputClaim ClaimTypeReferenceId="surName" />
112             </OutputClaims>
113             <ValidationTechnicalProfiles>
114               <ValidationTechnicalProfile ReferenceId="AAD-UserWriteUsingLogonEmail" />
115             </ValidationTechnicalProfiles>
116             <!-- Sample: Disable session management for sign-up page -->
117             <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
118           </TechnicalProfile>
119         </TechnicalProfiles>
120       </ClaimsProvider>
```

User Journeys

```
<UserJourneys>
  <UserJourney Id="SignUpOrSignIn_Custom">
    <OrchestrationSteps>
      <OrchestrationStep Order="1" Type="CombinedSignInAndSignUp" ContentDefinitionReferenceId="api.signuporsignin">
        <ClaimsProviderSelections>
          <ClaimsProviderSelection ValidationClaimsExchangeId="LocalAccountSigninEmailExchange" />
        </ClaimsProviderSelections>
        <ClaimsExchanges>
          <ClaimsExchange Id="LocalAccountSigninEmailExchange" TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
        </ClaimsExchanges>
      </OrchestrationStep>

      <!-- Check if the user has selected to sign in using one of the social providers -->
      <OrchestrationStep Order="2" Type="ClaimsExchange">
        <Preconditions>
          <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
            <Value>objectId</Value>
            <Action>SkipThisOrchestrationStep</Action>
          </Precondition>
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="SignUpWithLogonEmailExchange_EmailVerification" TechnicalProfileReferenceId="EmailVerification" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="3" Type="ClaimsExchange">
        <Preconditions>
          <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
            <Value>objectId</Value>
            <Action>SkipThisOrchestrationStep</Action>
          </Precondition>
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="SignUpWithLogonEmailExchange_WithReadOnlyEmail" TechnicalProfileReferenceId="LocalAccountSignUpWithReadOnlyEmail" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="4" Type="ClaimsExchange">
        <ClaimsExchanges>
          <ClaimsExchange Id="AADUserReadWithObjectId" TechnicalProfileReferenceId="AAD-UserReadUsingObjectId" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="5" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer" />
    </OrchestrationSteps>
  </UserJourney>
```

1

2

3

4

The User Journey is made up of Orchestration Steps which help dictate the order of which the claims are called/executed.

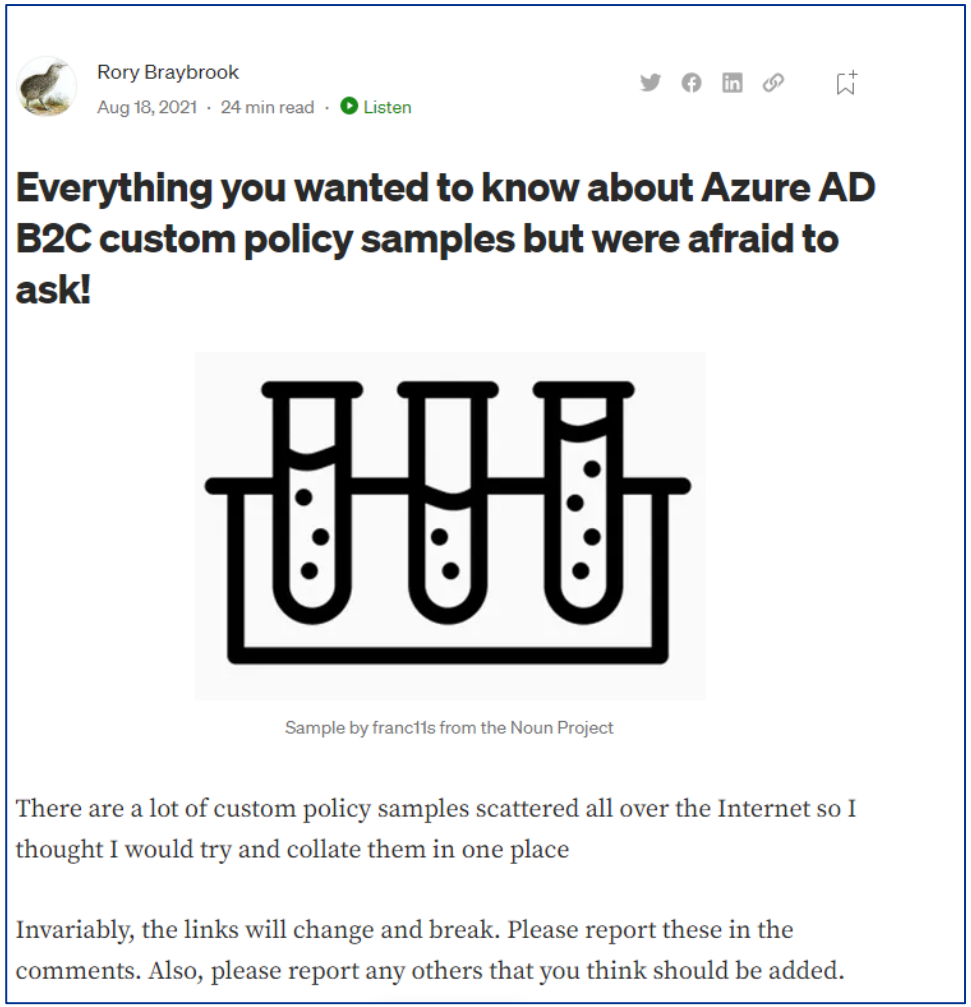
- **Step 1** – If User selects sign in, then the Local Account Sign In Email Exchange claims provider is selected.
- **Step 2 & 3** – If the User has an account, these steps are skipped. If the User does not have an account, then the 2nd step will send them an email verification and the 3rd step will ask the User to input their personal account details.
- **Step 4** – Retrieve the User's information from Azure AD.

Appendix

Helpful links

Helpful Links

[Everything you wanted to know about Azure AD B2C custom policy samples but were afraid to ask! | by Rory Braybrook | The new control plane | Medium]



Rory Braybrook

Aug 18, 2021 · 24 min read · Listen

Everything you wanted to know about Azure AD B2C custom policy samples but were afraid to ask!

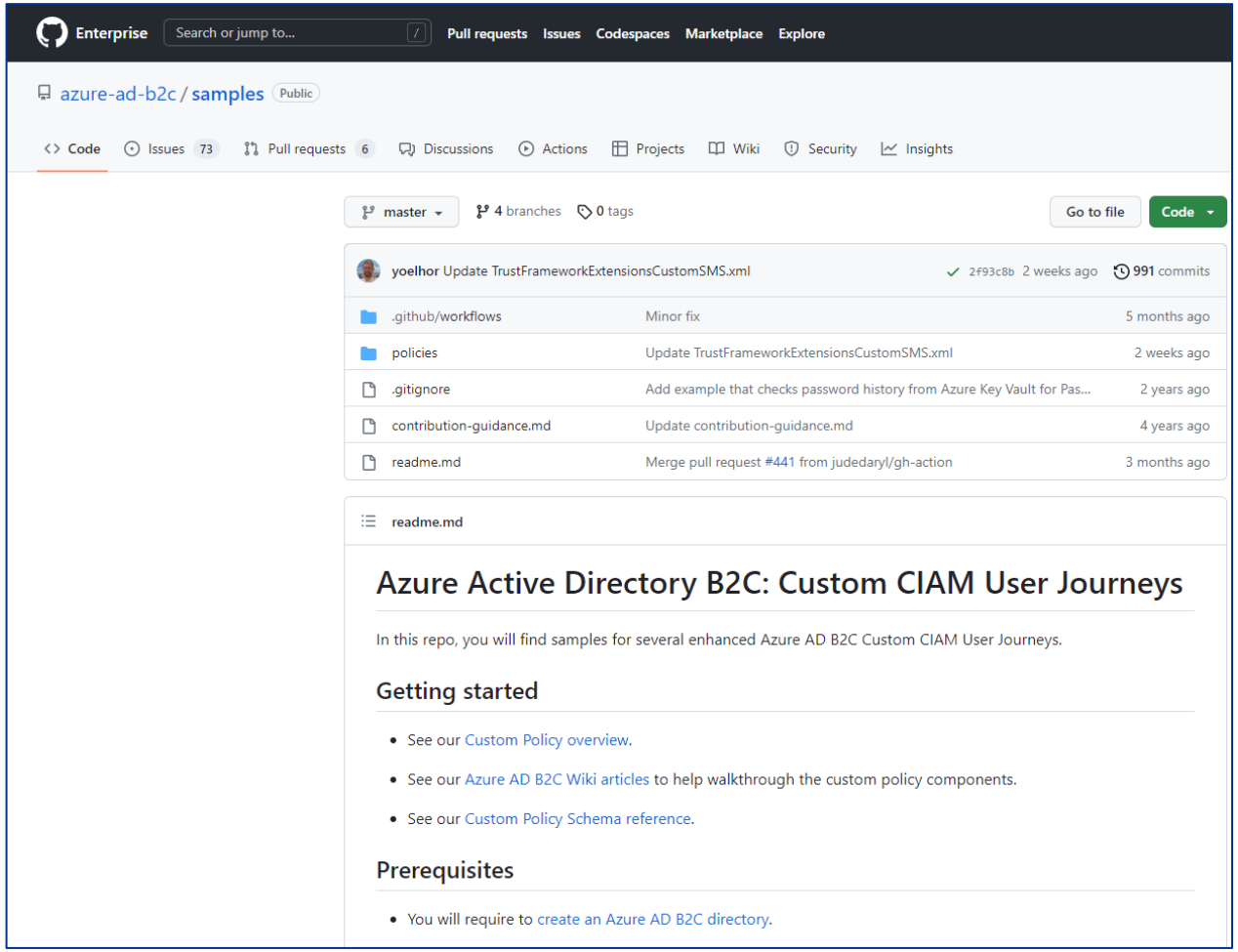
Sample by franc11s from the Noun Project

There are a lot of custom policy samples scattered all over the Internet so I thought I would try and collate them in one place

Invariably, the links will change and break. Please report these in the comments. Also, please report any others that you think should be added.

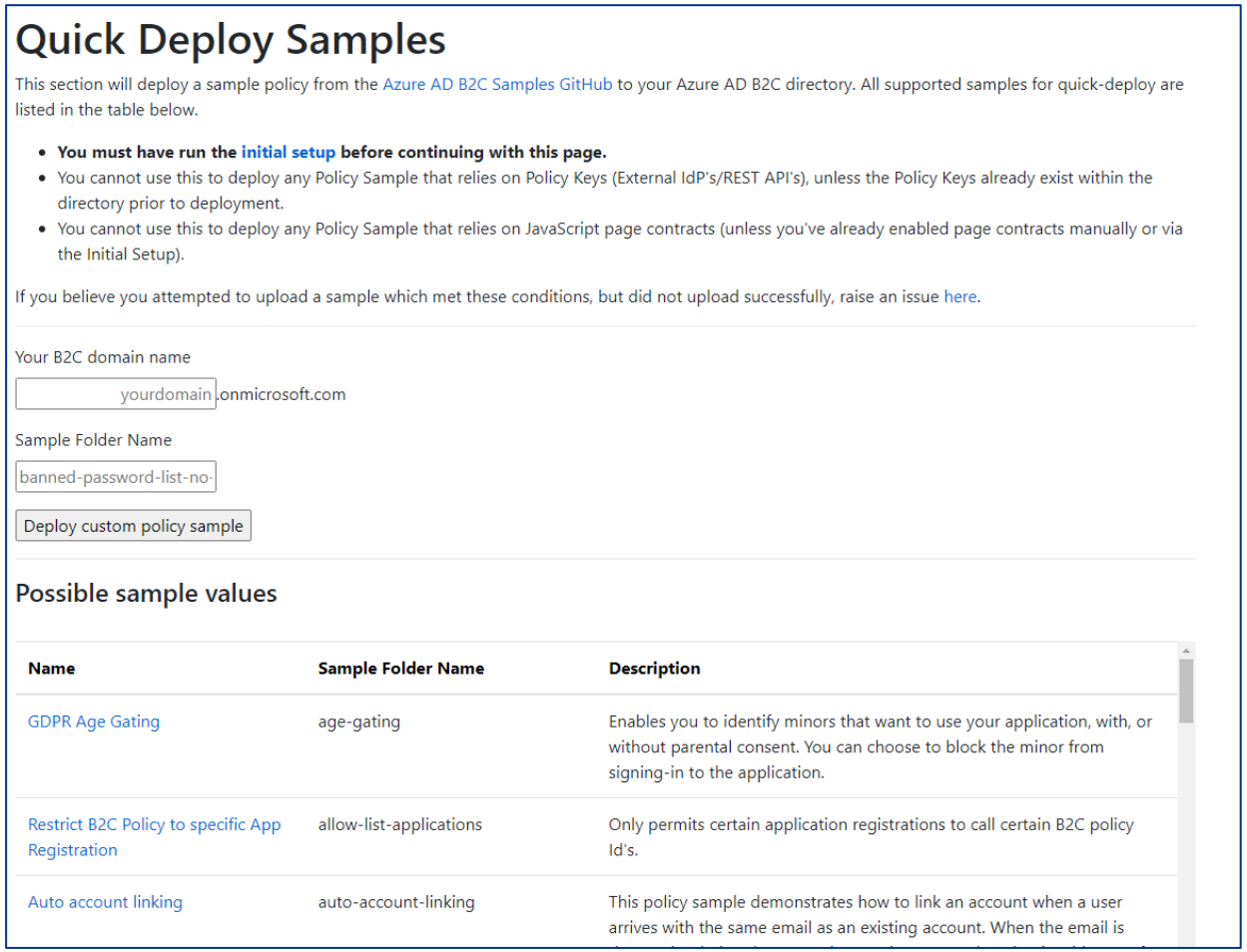
Helpful Links

[azure-ad-b2c/samples: Azure AD B2C custom policy solutions and samples. (github.com)]



Helpful Links

[Quick Deploy Samples - IEF Setup (b2ciefsetupapp.azurewebsites.net)]



Quick Deploy Samples

This section will deploy a sample policy from the Azure AD B2C Samples GitHub to your Azure AD B2C directory. All supported samples for quick-deploy are listed in the table below.

- **You must have run the initial setup before continuing with this page.**
- You cannot use this to deploy any Policy Sample that relies on Policy Keys (External IdP's/REST API's), unless the Policy Keys already exist within the directory prior to deployment.
- You cannot use this to deploy any Policy Sample that relies on JavaScript page contracts (unless you've already enabled page contracts manually or via the Initial Setup).

If you believe you attempted to upload a sample which met these conditions, but did not upload successfully, raise an issue here.

Your B2C domain name

yourdomain .onmicrosoft.com

Sample Folder Name

banned-password-list-no-

Deploy custom policy sample

Possible sample values

Name	Sample Folder Name	Description
GDPR Age Gating	age-gating	Enables you to identify minors that want to use your application, with, or without parental consent. You can choose to block the minor from signing-in to the application.
Restrict B2C Policy to specific App Registration	allow-list-applications	Only permits certain application registrations to call certain B2C policy Id's.
Auto account linking	auto-account-linking	This policy sample demonstrates how to link an account when a user arrives with the same email as an existing account. When the email is