**Lab 21 – Embed Power BI**

**Creating a Service Principal**

To authenticate our app with Power BI when doing embed for customers, there are two Options: [Service Principal or Master User](https://docs.microsoft.com/en-us/power-bi/developer/embedded/embed-tokens#embed-for-your-customers). The Master User route uses an actual user account to authenticate and requires that user to have a Power BI Pro or Premium Per User (PPU) license. This is not ideal, so the recommended route is to setup and use a Service Principal.

To do so, first we create a new Azure AD app. Start by logging in to [Azure Portal](https://portal.azure.com/) and go to App registrations:

Graphical user interface, text, application

Description automatically generated

Click “New registration”:

Graphical user interface, text, application

Description automatically generated

Then setup the new registration with a “Name” and “Supported Account Type” of your choice. The default of Single Tenant works for now. Then click “Register”:

Graphical user interface, text, application, email

Description automatically generated

Now that we have an app registration, we are going to create a Client Secret that will be used in generating an Access Tokens needed to connect to Power BI Service. To do so, click on “Certificates & secrets”, hit the “New client secret” button, and fill out the “Description” and “Expires” fields. Currently, the max allowable time is 24 months, so we select that, then hit “Add”:

Graphical user interface, text, application, email

Description automatically generated

Now make sure to copy down the client secret “Value.” Once you leave this page, you won’t be able to view it again and will have to recreate one. We will use this value when we start to generate Access and Embed Tokens:

Graphical user interface, text, application, email

Description automatically generated

copy Value

Before moving on, head back to the “Overview” section to jot down a couple more values that will be used later. Namely, the **Client ID (Application ID)**and the **Tenant ID (Directory ID)**:

Graphical user interface, text, application, email

Description automatically generated

The last step in the Azure Portal for now is to create a Security Group and add the service principal into it. To do this, search for and click on “Azure Active Directory” in the portal:

Graphical user interface, application

Description automatically generated

Next click “New Group”:

Graphical user interface, text, application

Description automatically generated

Next, fill in the information for the Security Group:

* Group Type: Security
* Group Name: Can be anything you want. We put     “Power BI Service Principal”
* Membership: Assigned
* Members: Search and add the name of the App Registration created before. In our case, it’s “React Power BI Embed”

Now hit “Select” and then “Create”:

Graphical user interface, application

Description automatically generated

It might take a minute for the group to show up:

Graphical user interface, text, application, email

Description automatically generated

Now that the Security Principal is setup in a Security Group, we can configure the Power BI Tenant settings to enable Service Principals.

**Configure Power BI Tenant Settings**

Navigate to Power BI Service at <https://app.powerbi.com/> (for GCC customers, it would be <https://app.powerbigov.us/>). The first step is to configure our Power BI Tenant to allow embedding. Click on the three dots in the top right corner, and navigate to “Settings” > “Admin Portal”:

Graphical user interface, application

Description automatically generated

Note – You’ll need admin privileges in your tenant to update the following settings. If you’re not an admin, you will need to ask the tenant’s admin to enable these tenant settings for you.

Select “Tenant settings” and scroll down to the Developer settings section. Set “Embed content in apps” to Enabled, for “Apply To” select “Specific security groups,” and in the text field search and add the security group housing the service principal that was created in the previous step, and hit “Apply”:

Graphical user interface, application

Description automatically generated

Next, we will do the same thing for the next setting underneath. Set “Allow service principles to use Power BI APIs” to Enabled, for “Apply To” select “Specific security groups,” and in the text field search and add the security group housing the service principal that was created in the previous step, and hit “Apply”:

Graphical user interface, text, application

Description automatically generated

In order to implement Power BI Embedding, we will need a report to embed.

**Get Report Details**

Now you’ll be able to see the report we will soon embed. But before moving on, you’ll need to take note of two IDs in the URL. First is the group or **Workspace ID** which is found after the **/groups/** in the URL, and the second is the **Report ID** which is found after **/reports/**. We’ll be using these later to select which report to embed as well as for generating the Embed Token needed to authenticate with the app.

Example : <https://app.powerbi.com/groups/6528c507-c640-44b8-907b-f36861a609f8/reports/2761e4fd-3912-405c-9fe3-e3d60f452c4d/ReportSectiona105ea3d15387b029846>

A screenshot of a computer

Description automatically generated

**Adding the Service Principal to your Workspace**

One last step in Power BI is to enable your Azure AD App to access the reports in your workspace. To do so, open the “Workspaces” pane, hit the three dots, and select “Workspace Access”:

Search for and add our security group in the “Enter email addresses” as an admin or member. Click “Add” and then “Close”:

Graphical user interface, application

Description automatically generated

Now we will be able to use our service principal to authenticate and create an Embed Token to access content in this workspace.

**Generating the Embed Token**

Before we start building our React app, we need a way to generate an Embed Token that grants us access to embed Power BI content in our application. These tokens can’t be generated from a client device, so they will typically be part of an API or backend service for your app. If you have an Azure subscription, one option to easily do this is to use [Azure Functions](https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference?tabs=blob). That’s what we’ll use here. To get started, let’s go back to the [Azure Portal](https://portal.azure.com/) and select “Function App”:

Graphical user interface, text, application

Description automatically generated

Next, click the “Create” button and fill in the information for the function app for:

* Subscription: Use your active Azure subscription
* Resource Group: Create a new one if you don’t have one already
* Function App Name: Can be anything you want
* Runtime stack: .NET
* Version: Keep as default
* Region: Select whichever is closest to you

Once it’s filled out, hit the “Review + Create” button, then “Create” your function:

Graphical user interface, application

Description automatically generated

It will take a minute or so to complete. Once done, click “Go to resource”:

Graphical user interface, text, application

Description automatically generated

Inside the function app, where going to use the Webhook + API function to generate the Embed Token using the HTTP Trigger template. To get started, click on “Functions” under the Functions grouping, and select “Create”:

Graphical user interface, application

Description automatically generated

For Development Environment, we’re going to do “Develop in portal.” Select “HTTP trigger” as the template, then hit “Create”:

Graphical user interface, text, application, email

Description automatically generated

Once created, open the function and click the “Code + Test” pane to view the editor. Click the “Get function URL” and **copy down the default URL to our endpoint**. As the content, add the following below and be sure to set the correct values.

Set the isGCC flag to true or false so the correct values will be used for the APIs and scope. Also insert the values saved earlier for the following:

* groupId : the Power BI workspace ID
* reportId : The Power BI Report Id to embed
* clientId : The Azure App Registration client ID
* clientSecret : The client secret created for the App Registration
* tenantId : The tenant ID for the App Registration

**Code below: (Function Code)**

#r "Newtonsoft.Json" using System.Net; using System.Text; using Newtonsoft.Json; public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, ILogger log) { HttpClient authclient = new HttpClient(); var isGCC = false; var powerBI\_API\_URL = "api.powerbi.com"; var powerBI\_API\_Scope = "https://analysis.windows.net/powerbi/api/.default"; if(isGCC) { powerBI\_API\_Scope = "https://analysis.usgovcloudapi.net/powerbi/api/.default"; powerBI\_API\_URL = "api.powerbigov.us"; } // Power BI var groupId = "[INSERT WORKSPACE ID HERE]"; var reportId = "[INSERT REPORT ID HERE]"; // Azure App Registration var clientId = "[INSERT APP CLIENT ID HERE]"; var clientSecret = "[INSERT APP CLIENT SECRET HERE]"; var tenantId = "[INSERT APP TENANT ID HERE]"; var content = new FormUrlEncodedContent(new[] { new KeyValuePair<string, string>("grant\_type", "client\_credentials"), new KeyValuePair<string, string>("client\_id", clientId), new KeyValuePair<string, string>("scope", powerBI\_API\_Scope), new KeyValuePair<string, string>("client\_secret", clientSecret) }); // Generate Access Token to authenticate for Power BI var accessToken = await authclient.PostAsync($"https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/token", content).ContinueWith<string>((response) => { log.LogInformation(response.Result.StatusCode.ToString()); log.LogInformation(response.Result.ReasonPhrase.ToString()); log.LogInformation(response.Result.Content.ReadAsStringAsync().Result); AzureAdTokenResponse tokenRes = JsonConvert.DeserializeObject<AzureAdTokenResponse>(response.Result.Content.ReadAsStringAsync().Result); return tokenRes?.AccessToken;; }); // Get PowerBi report url and embed token HttpClient powerBiClient = new HttpClient(); powerBiClient.DefaultRequestHeaders.Add("Authorization", $"Bearer {accessToken}"); var embedUrl = await powerBiClient.GetAsync($"https://{powerBI\_API\_URL}/v1.0/myorg/groups/{groupId}/reports/{reportId}") .ContinueWith<string>((response) => { log.LogInformation(response.Result.StatusCode.ToString()); log.LogInformation(response.Result.ReasonPhrase.ToString()); PowerBiReport report = JsonConvert.DeserializeObject<PowerBiReport>(response.Result.Content.ReadAsStringAsync().Result); return report?.EmbedUrl; }); var tokenContent = new FormUrlEncodedContent(new[] { new KeyValuePair<string, string>("accessLevel", "view") }); var embedToken = await powerBiClient.PostAsync($"https://{powerBI\_API\_URL}/v1.0/myorg/groups/{groupId}/reports/{reportId}/GenerateToken", tokenContent) .ContinueWith<string>((response) => { log.LogInformation(response.Result.StatusCode.ToString()); log.LogInformation(response.Result.ReasonPhrase.ToString()); PowerBiEmbedToken powerBiEmbedToken = JsonConvert.DeserializeObject<PowerBiEmbedToken>(response.Result.Content.ReadAsStringAsync().Result); return powerBiEmbedToken?.Token; }); // JSON Response EmbedContent data = new EmbedContent { EmbedToken = embedToken, EmbedUrl = embedUrl, ReportId = reportId, AccessToken = accessToken }; string jsonp = JsonConvert.SerializeObject(data); // Return Response return new HttpResponseMessage(HttpStatusCode.OK) { Content = new StringContent(jsonp, Encoding.UTF8, "application/json") }; } public class AzureAdTokenResponse { [JsonProperty("access\_token")] public string AccessToken { get; set; } } public class PowerBiReport { [JsonProperty(PropertyName = "id")] public string Id { get; set; } [JsonProperty(PropertyName = "name")] public string Name { get; set; } [JsonProperty(PropertyName = "webUrl")] public string WebUrl { get; set; } [JsonProperty(PropertyName = "embedUrl")] public string EmbedUrl { get; set; } [JsonProperty(PropertyName = "datasetId")] public string DatasetId { get; set; } } public class PowerBiEmbedToken { [JsonProperty(PropertyName = "token")] public string Token { get; set; } [JsonProperty(PropertyName = "tokenId")] public string TokenId { get; set; } [JsonProperty(PropertyName = "expiration")] public DateTime? Expiration { get; set; } } public class EmbedContent { public string EmbedToken { get; set; } public string EmbedUrl { get; set; } public string ReportId { get; set; } public string AccessToken { get; set; } }

Feel free to hit the Test/Run button to verify everything works. What this function will return is the Embed Code and Embed URL needed for embedding the Power BI report in the React application. Additionally, the ReportID is shown to verify that the correct report is loaded, and the Access Token to verify that the authentication piece is working. Next, we will finally create the React App to embed the Power BI report.

**Creating a React Project**

To create the project, we will use a command line interface (CLI). Make sure you have Node and npm installed. Open a command prompt or terminal window, and navigate to the directory created in the previous step. Once there, enter the following command:

**npx create-react-app react-powerbi-embed**

Replace “react-powerbi-embed” with whatever name you want for this project. After setup is finished, let’s add the following libraries: [PowerBI-Client](https://github.com/microsoft/PowerBI-JavaScript) and [PowerBI-Client-React](https://github.com/microsoft/PowerBI-Client-React). The easiest way is to add use npm to install by using the following commands, and start your project:

**cd react-powerbi-embed**

**npm install powerbi-client**

**npm install powerbi-client-react**

**npm start**

This will open up the default browser, and if successful, you will see the following:

Adding Power BI Content

Now we will add the code to show the Power BI report. Load this project in your favorite IDE, and edit/replace the contents of these two files with the following code:

App.js (replace [INSERT FUNCTION URL HERE] with the Function URL created earlier). Also if working in GCC, uncomment the line hostname: “https://app.powerbigov.us/” :

**App.jsx**

import React, { useState, useEffect } from "react"; import { models, Report, Embed, service, Page } from "powerbi-client"; import { PowerBIEmbed } from "powerbi-client-react"; import './App.css'; function App() { const [responseConfig, setResponseConfig] = useState({}); useEffect(() => { var url = "[INSERT FUNCTION URL HERE]" method: "GET", headers: { "Content-Type": "application/json", }, }) .then((res) => res.json()) .then((response) => { setResponseConfig(response); }) .catch((error) => console.log(error)); }, []); return ( <div className="App"> <header className="App-header"> <h1>Power BI Embed: </h1> <PowerBIEmbed embedConfig={{ //hostname: "https://app.powerbigov.us/", type: "report", // Supported types: report, dashboard, tile, visual and qna id: responseConfig.ReportId, embedUrl: responseConfig.EmbedUrl, accessToken: responseConfig.EmbedToken, tokenType: models.TokenType.Embed, settings: { panes: { filters: { expanded: false, visible: false, }, pageNavigation: { visible: false, }, }, background: models.BackgroundType.Transparent, }, }} eventHandlers={ new Map([ [ "loaded", function () { console.log("Report loaded"); }, ], [ "rendered", function () { console.log("Report rendered"); }, ], [ "error", function (event) { console.log(event.detail); }, ], ]) } cssClassName={"report-style-class"} /> </header> </div> ); } export default App;

App.css:

.report-style-class {

height: 600px;

width: 100%;

}

.App {

text-align: center;

}

After updating these files hit save and view the results in your browser. You should see the Power BI report fully embedded and interactive