Salesforce App-Owns-Data Embedding Sample

**SalesforceAppOwnsDataEmbedding** is a sample project which demonstrates how to implement App-Owns-Data embedding with Power BI reports using the Salesforce Developer Experience (SFDX) and the SFDX CLI. This solution is built on top of an Apex class named [PowerBiEmbedManager](https://github.com/PowerBiDevCamp/SalesforceAppOwnsDataEmbedding/blob/main/SalesforceAppOwnsDataEmbedding/force-app/main/default/classes/PowerBiEmbedManager.cls) which is programmed to interact with both Azure AD and the [Power BI REST API](https://docs.microsoft.com/en-us/rest/api/power-bi/) as shown in the following diagram.



**PowerBiEmbedManager** implements [Client Credentials Flow](https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-client-creds-grant-flow) to acquire an app-only access token from Azure AD. The app-only access token makes it possible for the **PowerBiEmbedManager** to call the Power BI REST API under the identity of a service principal instead of the identity of a user which is a best practice for developing with App-Owns-Data embedding.

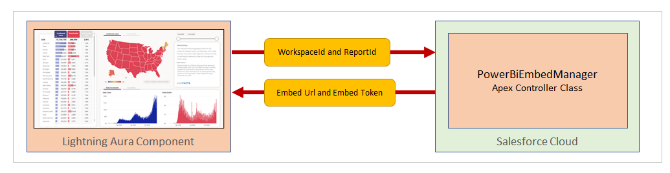
**PowerBiEmbedManager** calls the Power BI REST API to acquire metadata associated with a specific report ID such as the report's Embed Url and the underlying dataset Id. **PowerBiEmbedManager** also calls the Power BI REST API to generate embed tokens which are required when developing with App-Owns-Data embedding.

**PowerBiEmbedManager** exposes a public method named **getEmbeddingDataForReport** which is marked with the **AuraEnabled** annotation making it accessible to both Lighting Aura components and Lightning web components running in the browser. This is what makes it possible to move the embed data and the embed token for a report back to the browser where they can be used to embed a report.

The **SalesforceAppOwnsDataEmbedding** project contains a Lighting Aura component named [powerBiReportAura](https://github.com/PowerBiDevCamp/SalesforceAppOwnsDataEmbedding/tree/main/SalesforceAppOwnsDataEmbedding/force-app/main/default/aura/powerBiReportAura). When you add an instance of this component to a Lightning application page, you must configure it with the Workspace ID and the Report ID for a report in a Power BI workspace.



Once you have configure a **powerBiReportAura** component with a workspace Id and Report Id, this component will pass these two parameter values when it calls **getEmbeddingDataForReport**. The **PowerBiEmbedManager** class responds to a call to **getEmbeddingDataForReport** by returning the embedding data and the embed token requires to embed a report in the browser.



Once the call to **getEmbeddingDataForReport** returns back to the browser, the **powerBiReportAura** component has all the embedding data and the embed token required to embed a report. In a final step, the **powerBiReportAura** component uses the Power BI JavaScript API complete the report embedding process in the browser.



As the embedded report is loaded, it establishes a direct connection to the Power BI Service. As users begin to interact with the report by setting filters and adjusting slicers, these user actions result in direct calls to the Power BI Service.



## Setting Up This Sample Project

In order to setup and run this sample, you need to install the following software.

* [Install Node.JS](https://nodejs.org/en/download/)
* [Install Visual Studio Code](https://code.visualstudio.com/Download)

When you have installed Visual Studio Code, you must install a Visual Studio Code extension the Salesforce Expansion Pack.



Great blog article in 2017. But so much has changed.

So much has changed.

* Service principal can be used for App-Owns-Data embedding



Here is the GitHub repo with the sample code discussed in this article. This code is provided in an SFDX project. This is not an introduction to Salesforce development. It is expect the reader either knows the fundamentals or is willing to learn the fundamentals. Salesforce has done a great job at providing developer material at places such as trailhead.

Here are the Salesforce features

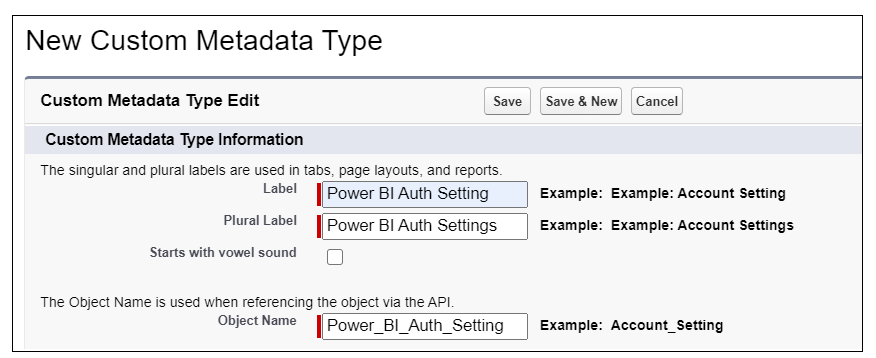
* Apex controller class
* Custom Metadata Type
* Remote Site Settings
* Lightning Aura component
* Lightning Web Component

# Getting Started with the Sample

Create remote site settings



Create Custom Metadata Types with auth settings



This is placeholder text.



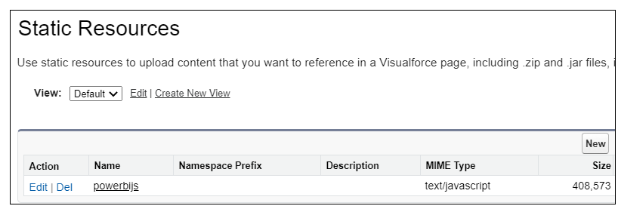
This is placeholder text.



This is placeholder text.



Upload powerbi.js as a Resource



## Salesforce DX Project: Next Steps

Now that you’ve created a Salesforce DX project, what’s next? Here are some documentation resources to get you started.

## How Do You Plan to Deploy Your Changes?

Do you want to deploy a set of changes, or create a self-contained application? Choose a [development model](https://developer.salesforce.com/tools/vscode/en/user-guide/development-models).

**Configure Your Salesforce DX Project**

The **sfdx-project.json** file contains useful configuration information for your project. See [Salesforce DX Project Configuration](https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_ws_config.htm) in the *\_Salesforce DX Developer Guide\_* for details about this file.

**Read All About It**

* [Salesforce Extensions Documentation](https://developer.salesforce.com/tools/vscode/)
* [Salesforce CLI Setup Guide](https://developer.salesforce.com/docs/atlas.en-us.sfdx_setup.meta/sfdx_setup/sfdx_setup_intro.htm)
* [Salesforce DX Developer Guide](https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_intro.htm)
* [Salesforce CLI Command Reference](https://developer.salesforce.com/docs/atlas.en-us.sfdx_cli_reference.meta/sfdx_cli_reference/cli_reference.htm)