# Salesforce App-Owns-Data Embedding Sample

[**SalesforceAppOwnsDataEmbedding**](https://github.com/PowerBiDevCamp/SalesforceAppOwnsDataEmbedding/tree/main/SalesforceAppOwnsDataEmbedding/force-app/main/default/aura/powerBiReportAura) is a sample project which demonstrates how to implement App-Owns-Data embedding with Power BI reports. This project has been created using the [**Salesforce Developer Experience (SFDX)**](https://developer.salesforce.com/developer-centers/developer-experience/) and the [**Salesforce CLI**](https://developer.salesforce.com/tools/sfdxcli). The goal of this sample project is to provide guidance and demonstrate best practices to developers who need to implement Power BI embedding in a Salesforce environment.

App-Owns-Data embedding has a big advantage over User-Owns-Data embedding when developing for Salesforce. More specifically, App-Owns-Data embedding does not require each user to have an Azure AD organizational account and a Power BI license. When developing with the App-Owns-Data embedding model, your Salesforce users can remain unknown to Power BI and your code has the flexibility to embed reports for any users you want.

## Project Architecture

The architecture of 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) when it interacts with Azure AD to acquire an app-only access token. App-only access tokens are important because they makes it possible to call the Power BI REST API under the identity of a service principal instead of calling under the identity of a user. Making calls to the Power BI REST API as service principal is a best practice for developing with App-Owns-Data embedding.

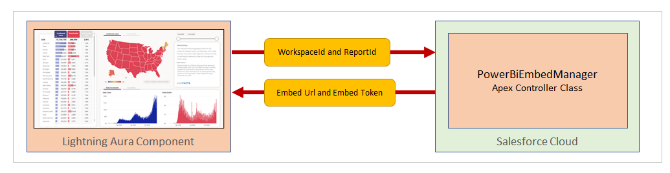
**PowerBiEmbedManager** must call the Power BI REST API for two different reasons. First, it much acquire embedding data such as the Embed Url associated with a specific report ID. Second, **PowerBiEmbedManager** must call the Power BI REST API to generate embed tokens which are required with App-Owns-Data embedding.

**PowerBiEmbedManager** has been designed as a controller class by exposing a public **getEmbeddingDataForReport** method which has been marked with the **AuraEnabled** annotation making it accessible to Lighting Aura components and to Lightning web components. A client-side component can call **getEmbeddingDataForReport** to retrieve the embedding data and the embed token for a specific 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 the **powerBiReportAura** component to a Lightning application page, you must configure it with the Workspace ID and the Report ID for a specific report in a Power BI workspace. This design makes it possible to add multiple instances of the **powerBiReportAura** component and configure each one to embed a different Power BI report.



Once you have configured a **powerBiReportAura** component instance with a workspace ID and Report ID, these two configuration valued will be passed as parameters when the component calls **getEmbeddingDataForReport**. The **PowerBiEmbedManager** class responds to this remote function call by returning the embedding data and the embed token which will be used to embed a report in the browser.



Once the **powerBiReportAura** component has successfully called **getEmbeddingDataForReport**, it has the embedding data and the embed token it needs to embed a report on the hosting web page. In a final step, the **powerBiReportAura** component executes JavaScript code in the browser and uses the [**Power BI JavaScript API**](https://docs.microsoft.com/en-us/javascript/api/overview/powerbi/overview) to implement the report embedding process.



When a Power BI report is embedded on a Web page such as a Lightning application page, it establishes a direct connection back to the Power BI Service. Once the report has loaded, the user can begin to interact with it by setting filters and navigating between pages. As users interact with the report, these interactions are handled by direct communications between the report and the Power BI Service.



## Setting Up This Sample Project

In order to set up and test this sample project, you'll need a Power BI report in a Microsoft 365 tenant in which you can create a new Azure AD application. You'll also need a Salesforce development environment. If you don't already have a Salesforce development environment, you can sign up for one for free using the [Salesforce lightning platform signup page](https://developer.salesforce.com/signup).

Once you have a Salesforce organization for testing, you will need to complete the following three tasks to configure the environment for the **SalesforceAppOwnsDataEmbedding** project.

* Add remote site settings for the Azure AD token endpoint and the Power BI Service API
* Create an Azure AD application to call the Power Service API
* Create a Custom Metadata Type to store client credentials for the Azure AD application
* Create a static resource by uploading a copy of the Power BI JavaScript API (powerbi.js)

Start by navigating to the Salesforce organization **Setup page** and searching for the **Remote Site Settings** page. Use the Remote Site Setting to add remote site settings for **https://login.microsoftonline.com** and **https://api.powerbi.com**. Once again, these configuration settings are required so that code in the Apex class can execute HTTP requests to Azure AD and the Power BI REST API.

Graphical user interface, text, application, email

Description automatically generated

After adding the remote site settings, the next step is to create a new Custom Metadata Type.

Create Custom Metadata Types with auth settings

Graphical user interface, text, application, email

Description automatically generated

This is placeholder text.

Graphical user interface, text, application

Description automatically generated

This is placeholder text.

Graphical user interface, application

Description automatically generated

This is placeholder text.

Graphical user interface, application

Description automatically generated

Upload powerbi.js as a Resource

Graphical user interface, text, application, email

Description automatically generated

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

## 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)