# The App-Owns-Data Starter Kit

The  **App-Owns-Data Starter Kit** is a developer sample built using the .NET 5 SDK to provide guidance for organizations and ISVs who are using App-Owns-Data embedding with Power BI in a multi-tenant environment. This solution consists of a custom database and three separate web applications which demonstrate common design patterns in App-Owns-Data embedding such as creating new Power BI workspaces for tenants, assigning user permissions and monitoring report usage and performance.

If you have worked with Azure AD, the word **"tenant"** might make you think of an Azure AD tenant. However, the concept of a tenant is different when designing a multi-tenant environment for App-Owns-Data embedding. In this context, each tenant represents a customer with one or more users for which you are embedding Power BI reports. In a multi-tenant environment, you must create a separate tenant for each customer. Provisioning a new customer tenant in a Power BI embedding solution typically involves writing code which programs the Power BI REST API to create a Power BI workspace, assign the workspace to a dedicated capacity, import PBIX files, patch datasource credentials and start dataset refresh operations.

There is a critical aspect to App-Owns-Data embedding that you must start thinking about during the initial design phase. A distinct advantage of App-Owns-Data embedding is that you pay Microsoft by licensing dedicated capacities instead of by licensing individual users. This allows organizations and ISVs to reach users that remain completely unknown to Power BI. While keeping users unknown to Power BI has its advantages, it also introduces a new problem that makes things more complicated than developing with User-Owns-Data embedding.

So, what's the problem? If Power BI doesn’t know anything about your users, Power BI cannot really provide any assistance when it comes to authorization and determining which users should have access to what content. This isn't a problem in a simplistic scenario where you intend to give every user the same level of access to the exact same content. However, it's far more common that your application requirements will define authorization policies to determine which users have access to which customer tenants. Furthermore, if you're planning to take advantage of the Power BI embedding support for report authoring, you'll also need to implement an authorization scheme that allows an administrator to assign permissions to users with a granularity of view permissions, edit permissions and content create permissions.

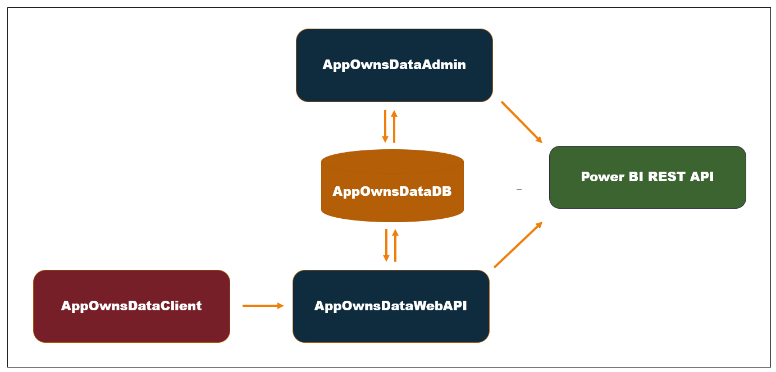
Now let's make three key observations about developing with App-Owns-Data embedding. First, you have the **flexibility** to design the authorization scheme for your application any way you'd like. Second, you have the ***responsibility*** to design and implement this authorization scheme from the ground up. Third, it's much easier to prototype and develop an authorization scheme if your application design includes a ***custom database*** to track whatever data and metadata you need to implement the authorization policies and policy enforcement you require.

The **App-Owns-Data Starter Kit solution** provides a starting point for organizations and ISVs who are beginning to develop with App-Owns-Data embedding. This solution was created to provide guidance and to demonstrate implementing the following application requirements that are common when developing with App-Owns-Data embedding in a multi-tenant.

* Onboarding new customer tenants
* Assigning and managing user permissions
* Implementing the customer-facing client as a Single Page Application (SPA)
* Creating a custom telemetry layer to log user activity
* Monitoring user actions such as ViewReport, EditReport and CreateReport
* Monitoring the performance of loading and rendering reports

## Solution Architecture

The **App-Owns-Data Starter Kit** solution is built on top of a custom SQL Server database named **AppOwnsDataDB**. In addition to the **AppOwnsDataDB** database, the solution contains three Web application projects named **AppOwnsDataAdmin**, **AppOwnsDataClient** and **AppOwnsDataWebApi** as shown in the following diagram.



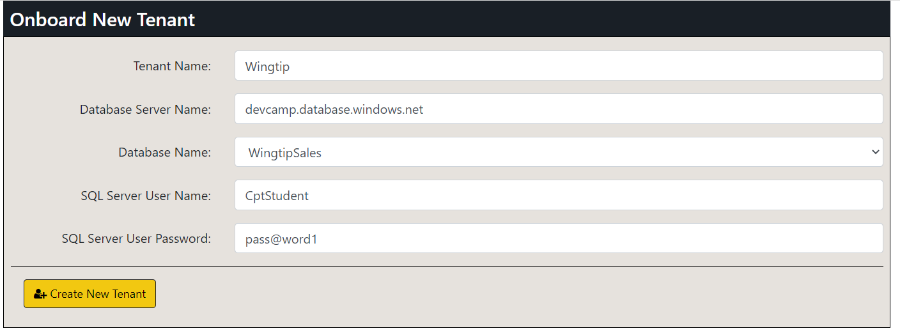
Let's begin with a brief description of each of these three web applications.

* **AppOwnsDataAdmin**: administrative application used to create tenants and manage user permissions.
* **AppOwnsDataClient**: customer-facing SPA used to view and author reports.
* **AppOwnsDataWebApi**: custom Web API used by the **AppOwnsDataClient** application.

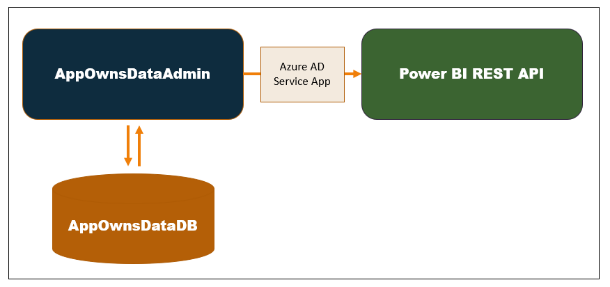
Now, we'll look at each of these web applications in a little more depth.

### Understanding the AppOwnsDataAdmin application

The **AppOwnsDataAdmin** application is used by the hosting company to manage its multi-tenant environment. The **AppOwnsDataAdmin** application provides administrative users with the ability to create new customer tenants. The **Onboard New Tenant** form of the **AppOwnsDataAdmin** application allows you the specify the **Tenant Name** along with the details to connect to a SQL Server database with the customer's data.

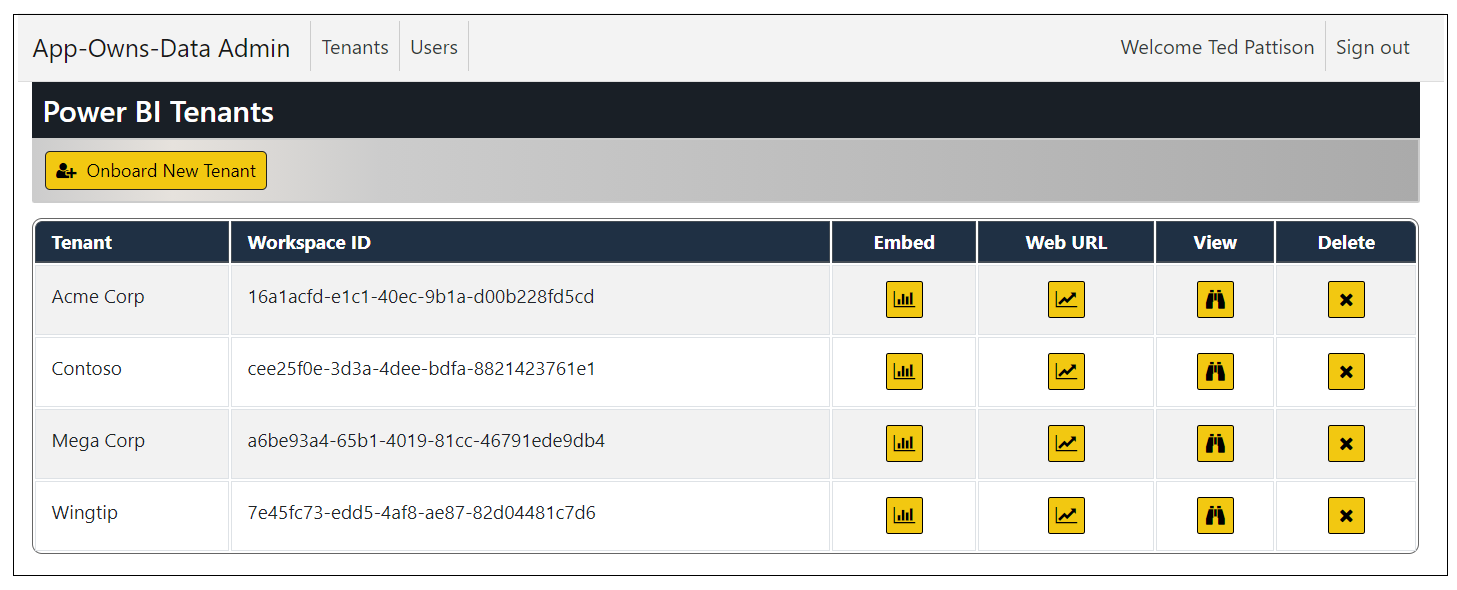


When you click the **Create New Tenant** button, the **AppOwnsDataAdmin** application makes several calls to the Power BI Service API to provision the new customer tenant. The **AppOwnsDataAdmin** application calls the Power BI Service under the identity of a service principal associated with an Azure AD application. When the service principal creates a new workspace, it is automatically included as workspace member in the role of Admin giving it full control over anything inside the workspace. When creating a new Power BI workspace, the **AppOwnsDataAdmin** application retrieves the new workspace ID and tracks it with a new record in the **PowerBiTenants** table in the **AppOwnsDataDB** database.

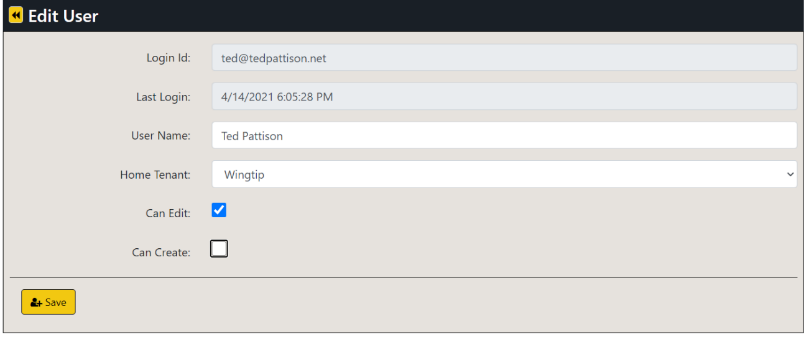


After creating a new Power BI workspace, the **AppOwnsDataAdmin** application continues the tenant onboarding process by importing a [template PBIX file](https://github.com/PowerBiDevCamp/TenantManagement/raw/main/TenantManagement/wwwroot/PBIX/DatasetTemplate.pbix) to create a new dataset and report that are both named **Sales**. Next, the tenant onboarding process updates dataset parameters redirect the **Sales** dataset to the SQL Server database that holds the customer's data. After that it patches the datasource credentials for the SQL Server database and starts a refresh operation to populate the **Sales** dataset with data from the customer's database.

After creating customers tenants in the **AppOwnsDataAdmin** application, they can be viewed, managed or deleted from the **Power BI Tenants** page.

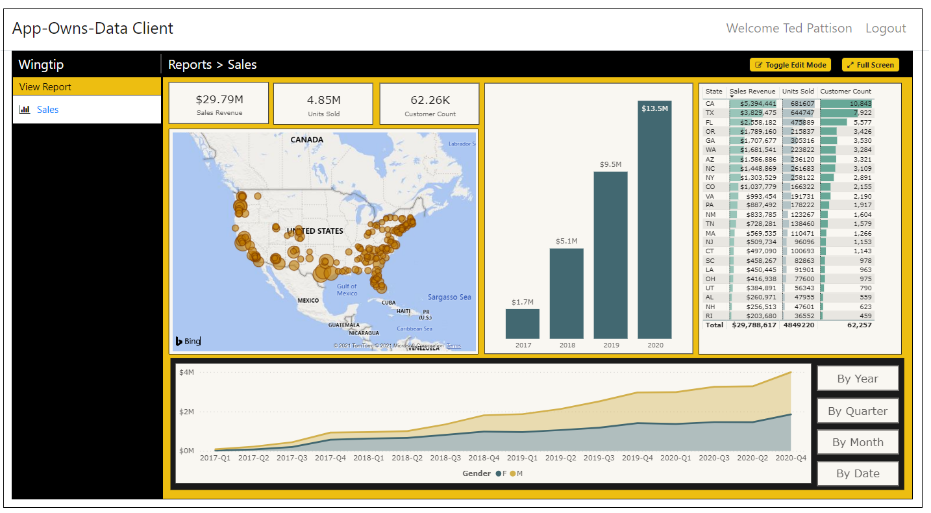


The **Edit User** form of **AppOwnsDataAdmin** application provides administrative users with a UI experience to assign users to a customer tenant. It also makes it possible to configure the user permission assignment within a tenant with a granularity of view permissions, edit permissions and create permissions.

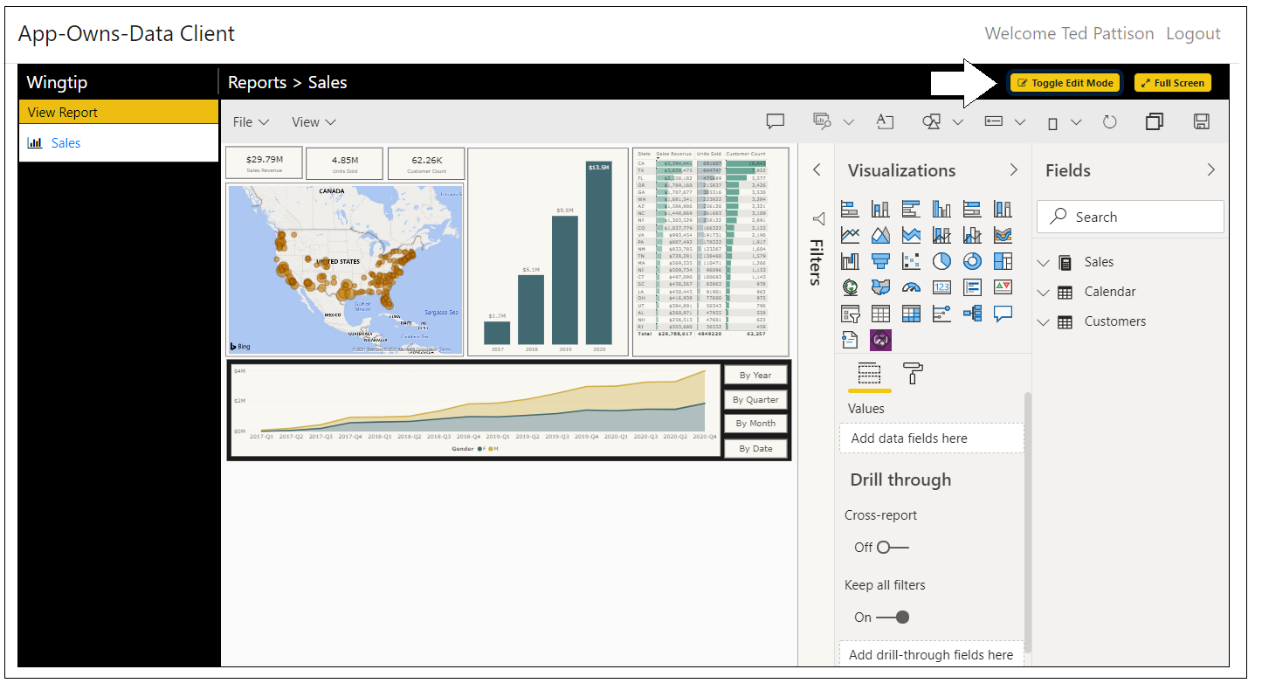


### Understanding the AppOwnsDataClient application

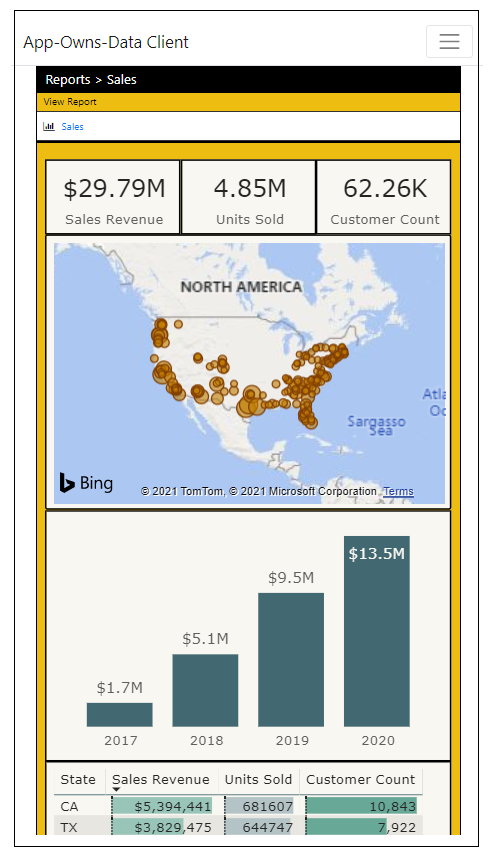
The **AppOwnsDataClient** application is the web application used by customers to access embedded reports within a customer tenant. This application has been created as an SPA to provide the best reach across different browsers and to provide a responsive design for users accessing the application using a mobile device or a tablet. Here is a screenshot of the **AppOwnsDataClient** application when run in the full browser experience on a desktop or laptop computer.



The **AppOwnsDataClient** application provides a report authoring experience when it see the current user has edit permission or create permissions. For example, the **AppOwnsDataClient** application displays a **Toggle Edit Mode** button when it sees the current user has edit permissions. This allows the user to customize a report using the same report editing experience provided to SaaS users in the Power BI Service. After customizing a report, a user with edit permissions can save the changes using the **File > Save** command.



We live in an age where targeting mobile devices and tablets is a common application requirement. The **AppOwnsDataClient** application was created with a responsive design. The PBIX template file for the **Sales** report provides a mobile view in addition to the standard master view. There is client-side JavaScript in the **AppOwnsDataClient** application which determines whether to display the master view or the mobile view depending on the width of the hosting device. If you change the width of the browser window, you can see the report transition between the master view and the responsive view. The following screenshot shows what the **AppOwnsDataClient** application looks like when viewed on a mobile device such as an iPhone.

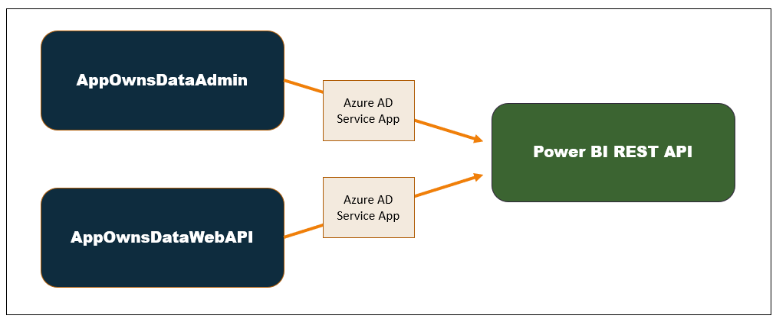


### Understanding the AppOwnsDataWebAPI application

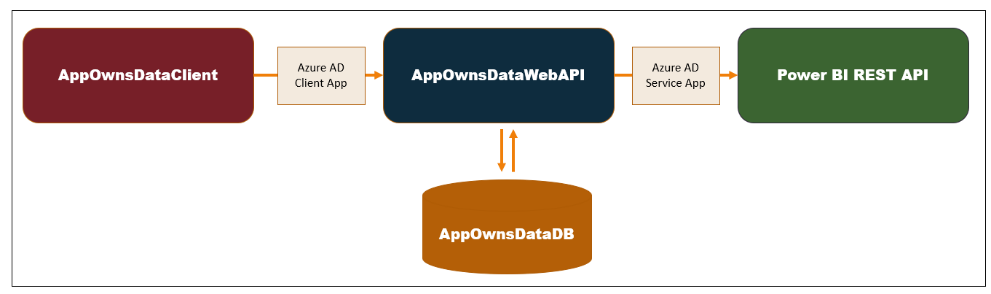
When developing with App-Owns-Data embedding, it's a best practice to call the Power BI Service API as a service principal. This requires an application to implement Client Credentials Flow to interact with Azure AD and acquire an app-only access token. From an architectural viewpoint, this type of code must be designed to run on the server-side and never as client-side code running in the browser. If you were to pass app-only tokens or the secrets used to acquire them to the browser, you would introduce a serious security vulnerability in your design. An attacker that was able to capture an app-only access token would be able to call into the Power BI Service API with full control over every tenant workspace in Power BI.

When using App-Owns-Data embedding, you must pass a security token back to the browser. However, this security token is not an Azure AD access token but instead an Power BI **embed token**. Unlike Azure AD access tokens, embed tokens are created by the Power BI REST API and not Azure AD. You generate embed tokens by calling the Power BI REST API using the trusted code that runs as the all-powerful service principal. When the service principal creates and embed token, it is the developer responsibility to determine exactly how much (or how little) permissions to give to the current user.

In the **App-Owns-Data Starter Kit** solution, **AppOwnsDataWebApi** authenticates using the same Azure AD application as **AppOwnsDataAdmin**. That means that both applications run under the identity of a single service principal giving **AppOwnsDataWebApi** admin access to any Power BI workspaces that has been created by **AppOwnsDataAdmin**.



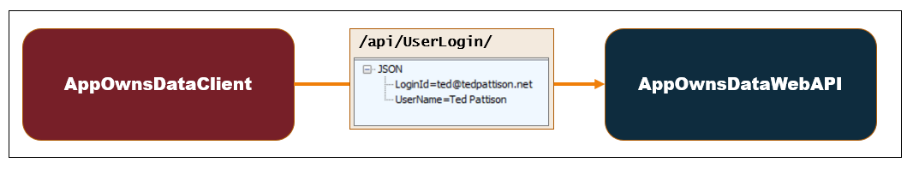
The **AppOwnsDataClient** application is designed to be a consumer of the Web API exposed by **AppOwnsDataWebApi**. The security requirements for this type of service-oriented architecture require a second Azure AD application which makes it possible for users of the **AppOwnsDataClient** application to login and to make secure APIs calls to **AppOwnsDataWebApi**.



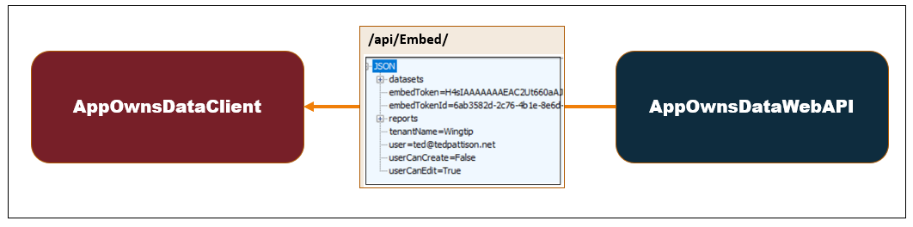
When the **AppOwnsDataClient** application executes an API call on **AppOwnsDataWebApi**, it's required to pass an access token that's been acquired from Azure AD. **AppOwnsDataWebApi** is able validate the access token and determine the user's login ID. Once **AppOwnsDataWebApi** determines the login ID for the current user, it can then retrieve user profile data from **AppOwnsDataDB** to determine what permissions have been assigned to this user and build the appropriate level of permissions into the embed token returned to the client application.

The Azure AD application for the **AppOwnsDataClient** application is configured to support organizational accounts from any Microsoft 365 tenant as well as Microsoft personal accounts for Skype and Xbox. You could take this further by using the support in Azure AD for authenticating users with other popular identity provides such as Google, Twitter and Facebook. After all, a key advantage of App-Owns-Data embedding is that you can use any identity provider you'd like.

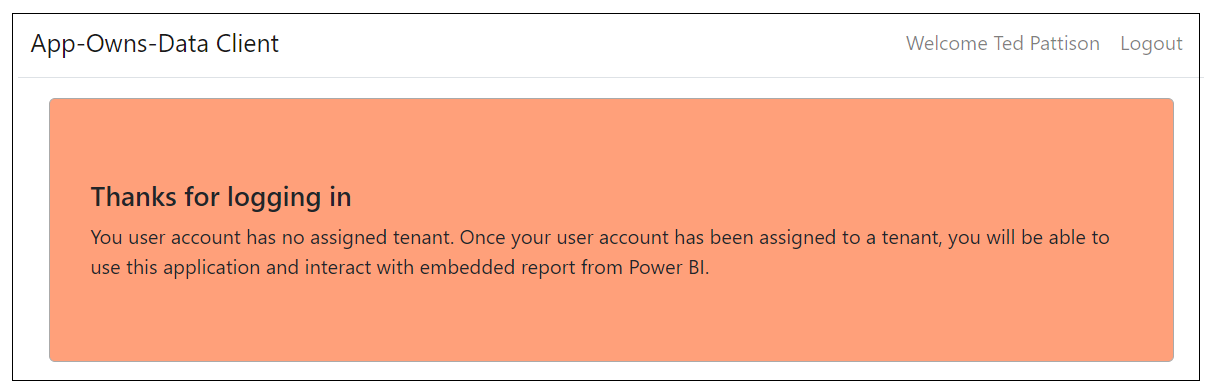
Now let's examine what goes on behind the scenes when a user launches the **AppOwnsDataClient** application. When the user first authenticates with Azure AD, the **AppOwnsDataClient** application calls to the **UserLogin** endpoint of **AppOwnsDataWebApi** and passes the user's **LoginId** and **UserName**. This allows **AppOwnsDataWebApi** to update the **LastLogin** value for existing users and to add a new record for any authenticated user who did not previous have an associated record in the **Users** table of **AppOwnsDataDB**.



After the user has logged in, the **AppOwnsDataClient** application calls the **Embed** endpoint to retrieve a view model which contains all the data required for embedding reports from the user's tenant workspace in Power BI. As you know, this view model includes the embed token which has been generated to give the user the correct level of permissions.

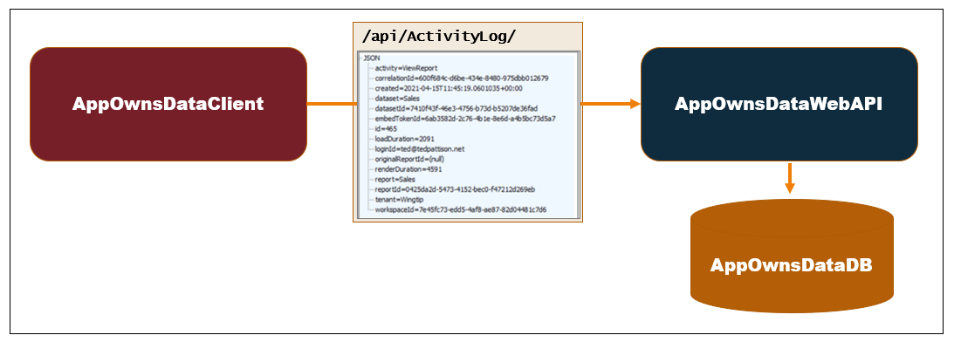


Any user with an organizational account or a personal account can log into the **AppOwnsDataClient** application. When a user logs in for the first time, **AppOwnsDataWebApi** automatically adds a new record for the user. However, when users are created on the fly in this fashion, they are not assigned to any customer tenant. In this scenario where the user is unassigned, **AppOwnsDataWebApi** returns a view model with no embedding data and a blank tenant name. The **AppOwnsDataClient** application responds to this view model with the following screen notifying the user that they need to be assigned to a tenant before they can begin to view reports.

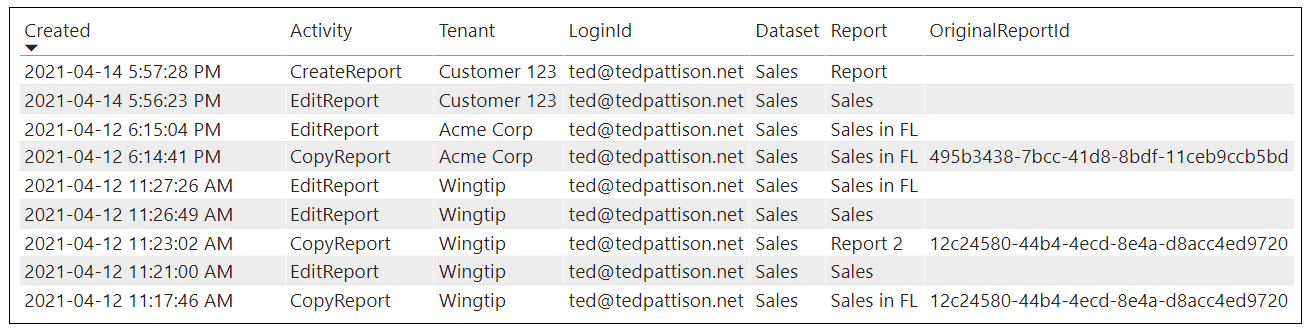


### Designing a custom telemetry layer

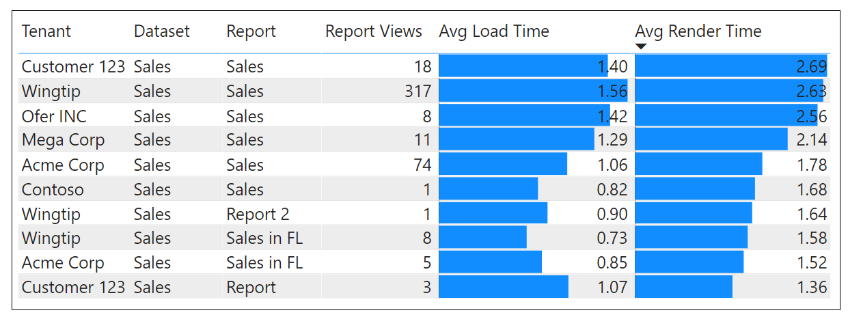
A valuable aspect of the **App-Owns-Data Starter Kit** architecture is it adds its own custom telemetry layer. The **AppOwnsDataClient** application has been designed to call the **ActivityLog** endpoint of **AppOwnsDataWebApi** whenever there is user activity that needs to be monitored. **AppOwnsDataWebApi** responds to calls to the **ActivityLog** endpoint by creating a new record in the **ActivityLog** table in **AppOwnsDataDB** to record the user activity. This makes it possible to monitor user activity such as viewing reports, editing reports, creating reports and copying reports.



Given the architecture of this custom telemetry layer, it's now possible to see all user activity for report viewing and report authoring by examining the records in the **ActivityLog** table.



In addition to capturing usage data focused on user activity, this telemetry layer also captures performance data which makes it possible to monitor how fast reports are loaded and rendered in the browser. This is accomplished by adding client-side code using the Power BI JavaScript API which records the load duration and the render duration anytime it embed a report. This makes it possible to monitor report performance across a multi-tenant environment to see if any reports require attention due to slow loading and rendering times.



Many developer who are beginning to develop with App-Owns-Data embedding spend time trying to figure out how to monitor user activity by using the [Power BI activity log](https://docs.microsoft.com/en-us/power-bi/admin/service-admin-auditing) which is automatically generated by the Power BI Service. However, this is not as straightforward as one might expect when developing with App-Owns-Data embedding. What happens in the scenario when a report is embedded using an embed token generated by a service principal? In this scenario, the Power BI activity log does not record the name of the actual user. Instead, the Power BI activity logging service adds the Application ID of the service principal as the current user. Unfortunately, that doesn't provide useful information with respect to user activity.

In order to map user names in an App-Owns-Data embedding scenario to events in the Power BI activity log, there is extra work required. When you embed a report with client-side code in the browser, it's possible to capture a ***correlation ID*** which maps back to the request ID for an event in the Power BI activity log. The idea is that you can map the correlation ID and the current user name back to a request ID in the Power BI activity log. However, that takes more work and this extra effort doesn't really provide any additional usage data beyond what being recorded with the custom telemetry layer that is demonstrated in the **App-Owns-Data Starter Kit** solution.

At this point, you might ask yourself whether it's important to integrate the Power BI activity log into a solution that uses App-Owns-Data embedding. The answer is no. It becomes unnecessary once you have created your own custom telemetry layer. Furthermore, it usually take about 15 minutes for activity to show up in the Power BI activity log. Compare this to a custom telemetry layer where usage data is available immediately after an event has been logged by the **AppOwnsDataClient** application.

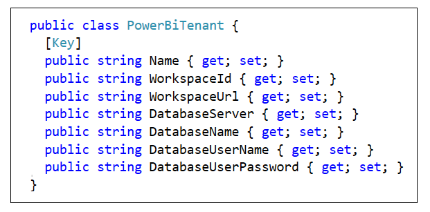
### Understanding the AppOwnsDataShared class library project

The **AppOwnsDataDB** database is built using the .NET 5 version of the Entity Framework known as [Entity Framework Core](https://docs.microsoft.com/en-us/ef/core/). Entity Framework supports the ***Code First*** approach where the developer starts by modeling database tables using classes defined in C#. The Code First approach has advantages while you're still in the development stage because its very easy to change the database schema in your C# code and then apply those changes to the database.

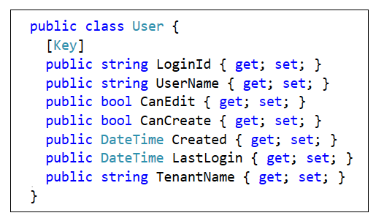
The C# code which creates and accesses the **AppOwnsDataDB** database is included in a class library project named **AppOwnsDataShared**. By adding the Entity Framework code to a class library project, it can be shared across the two web application projects for **AppOwnsDataAdmin** and **AppOwnsDataWebApi**.

One import thing to keep in mind is that the **AppOwnsDataShared** project is a class library which cannot have its own configuration file. Therefore, the connection string for the **AppOwnsDataDB** database is tracked in project configuration files for both **AppOwnsDataAdmin** and **AppOwnsDataWebApi**.

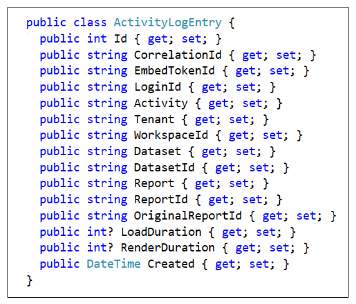
The **Tenants** table in **AppOwnsDataDB** is generated by a C# class named **PowerBITenant**.



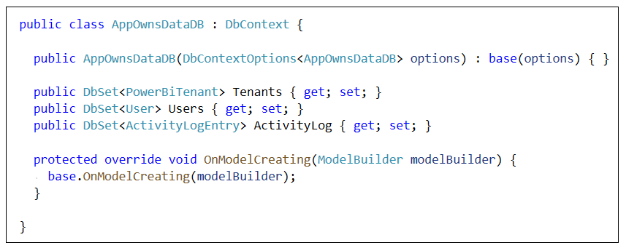
The **Users** table is generated using the table schema defined by the **User** class.



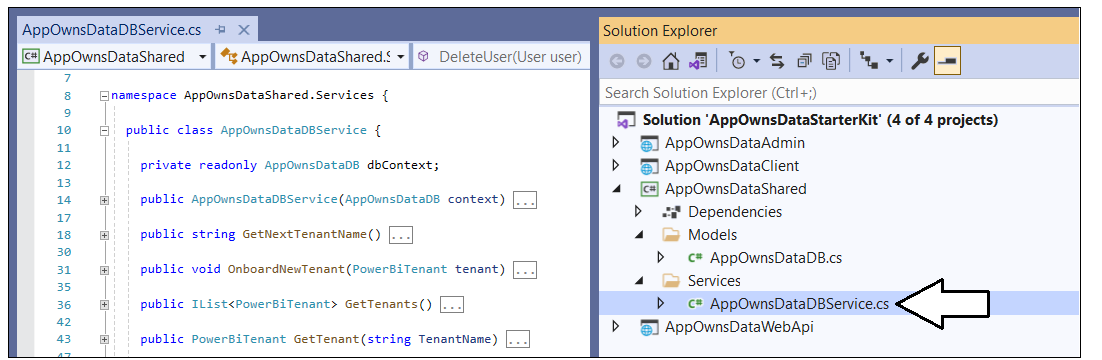
The **ActivityLog** table is generated using the table schema defined by the **ActivityLogEntry**.



The database model itself is created by the **AppOwnsDataDB** class which derives from **DBContext**.



The **AppOwsDataShared** project contains a public class named **AppOwnsDataDbService** which contains all the shared logic to execute read and write operations on the **AppOwnsDataDB** database. The **AppOwnsDataAdmin** application and **AppOwnsDataWebApi** both access **AppOwnsDataDB** by calling public methods in the **AppOwnsDataDbService** class.



## Set up your development environment

This section provides a step-by-step guide for setting up the **App-Owns-Data Starter Kit** solution for testing. To complete these steps, you will require a Microsoft 365 tenant in which you have permissions to create and manage Azure AD applications and security groups. You will also need Power BI Service administrator permissions to configure Power BI settings to give the service principal for an Azure AD application to ability to access the Power BI REST API. If you do not have a Microsoft 365 environment for testing, you can create one for free by following the steps in [Create a Development Environment for Power BI Embedding](https://github.com/PowerBiDevCamp/Camp-Sessions/raw/master/Create%20Power%20BI%20Development%20Environment.pdf).

To set up the  **App-Owns-Data Starter Kit** solution for testing, you will need to configure a Microsoft 365 tenant by completing the following tasks.

* Create an Azure AD security group named **Power BI Apps**
* Configure Power BI tenant-level settings for service principal access
* Create the Azure AD Application named **App-Owns-Data Service App**
* Create the Azure AD Application named **App-Owns-Data Client App**

The following four sections will step through each of these setup tasks in step-by-step detail.

### Create an Azure AD security group named Power BI Apps

Navigate to the [Groups management page](https://portal.azure.com/#blade/Microsoft_AAD_IAM/GroupsManagementMenuBlade/AllGroups) in the Azure portal. Once you get to the **Groups** page in the Azure portal, click the **New group** link.

[Graphical user interface, text, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image1.png)

In the **New Group** dialog, Select a **Group type** of **Security** and enter a **Group name** of **Power BI Apps**. Click the **Create** button to create the new Azure AD security group.

[Graphical user interface, text, application, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image2.png)

Verify that you can see the new security group named **Power BI Apps** on the Azure portal **Groups** page.

[Graphical user interface, text, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image3.png)

### Configure Power BI tenant-level settings for service principal access

Next, you need you enable a tenant-level setting for Power BI named **Allow service principals to use Power BI APIs**. Navigate to the Power BI Service admin portal at <https://app.powerbi.com/admin-portal>. In the Power BI Admin portal, click the **Tenant settings** link on the left.

[Graphical user interface, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image4.png)

Move down in the **Developer settings** section and expand the **Allow service principals to use Power BI APIs** section.

[Graphical user interface, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image5.png)

Note that the **Allow service principals to use Power BI APIs** setting is initially set to **Disabled**.

[Graphical user interface, text, application, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image6.png)

Change the setting to **Enabled**. After that, set the **Apply to** setting to **Specific security groups** and add the **Power BI Apps** security group as shown in the screenshot below. Click the **Apply** button to save your configuration changes.

[Graphical user interface, text, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image7.png)

You will see a notification indicating it might take up to 15 minutes to apply these changes to the organization.

[Text

Description automatically generated with medium confidence](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image8.png)

Now scroll upward in the **Tenant setting** section of the Power BI admin portal and locate **Workspace settings**.

[Graphical user interface, application, Teams

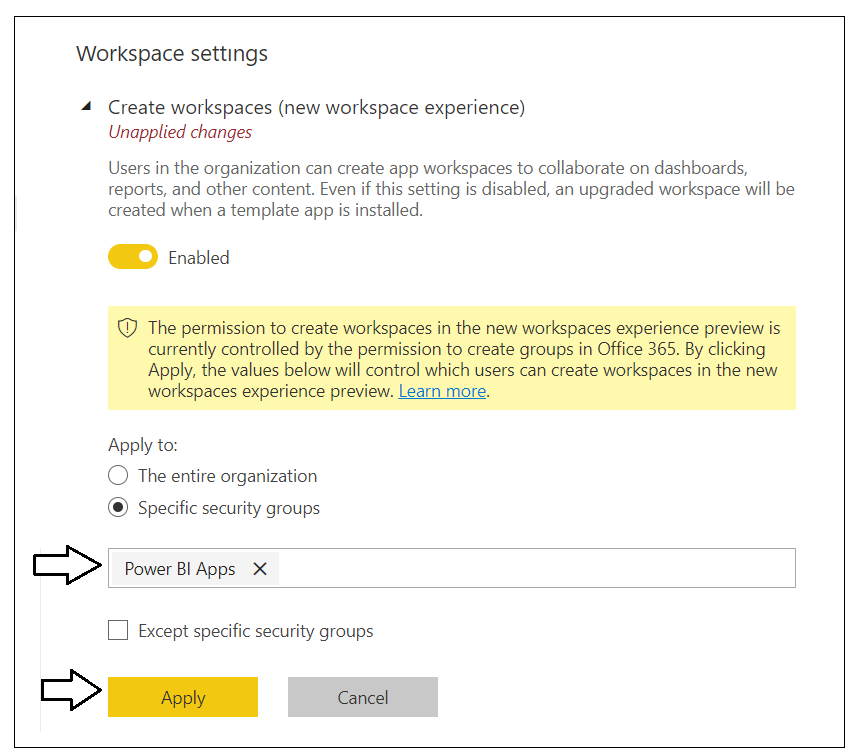
Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image9.png)

Note that a new Power BI tenant has an older policy where only users who have the permissions to create Office 365 groups can create new Power BI workspaces. You must reconfigure this setting so that service principals in the **Power BI Apps** group will be able to create new workspaces.

[Graphical user interface, text, application, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image10.png)

In **Workspace settings**, set **Apply to** to **Specific security groups, add the Power BI Apps security group and** click the **Apply** button to save your changes.



You have now completed the configuration of Power BI tenant-level settings.

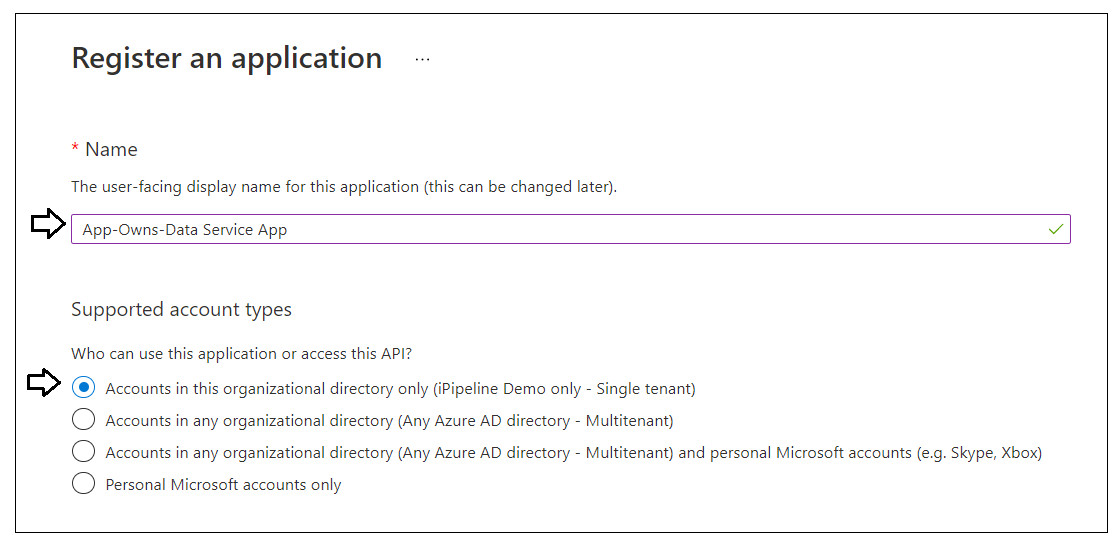
### Create the ****App-Owns-Data Service App in Azure AD****

Navigate to the [App registration](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps) page in the Azure portal and click the **New registration** link.

[Graphical user interface, text, application, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image12.png)

On the **Register an application** page, enter an application name of **App-Owns-Data Service App** and accept the default selection for **Supported account types** of **Accounts in this organizational directory only**.



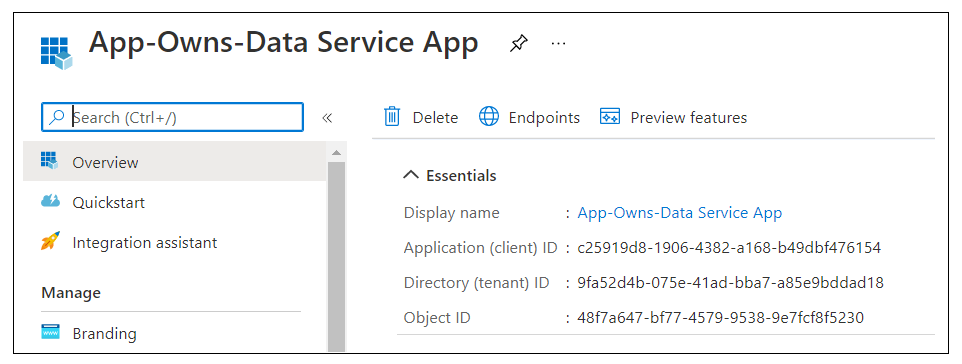
Complete the following steps in the **Redirect URI** section.

1. Leave the default selection of **Web** in the dropdown box
2. Enter a **Redirect URI** of [**https://localhost:44300/signin-oidc**](https://localhost:44300/signin-oidc)
3. Click the **Register** button to create the new Azure AD application.

[Graphical user interface, text, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image14.png)

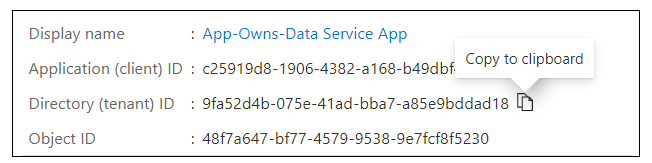
After creating a new Azure AD application in the Azure portal, you should see the Azure AD application overview page which displays the **Application ID**. Note that the ***Application ID*** is often called the ***Client ID***, so don't let this confuse you. You will need to copy this Application ID and store it so you can use it later to configure the project's support for Client Credentials Flow.



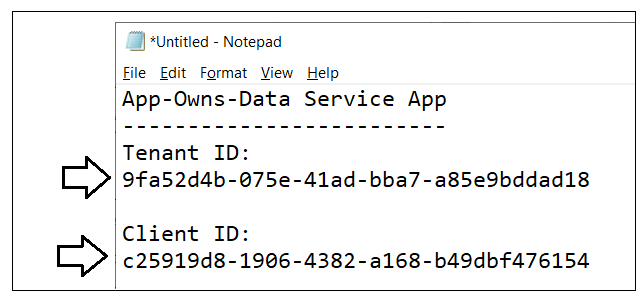
Copy the **Client ID** (aka Application ID) and paste it into a text document so you can use it later in the setup process. Note that this **Client ID** value will be used by both the **AppOwnsDataAdmin** project and the **AppOwnsDataWebApi** project to configure authentication for the service principal with Azure AD.



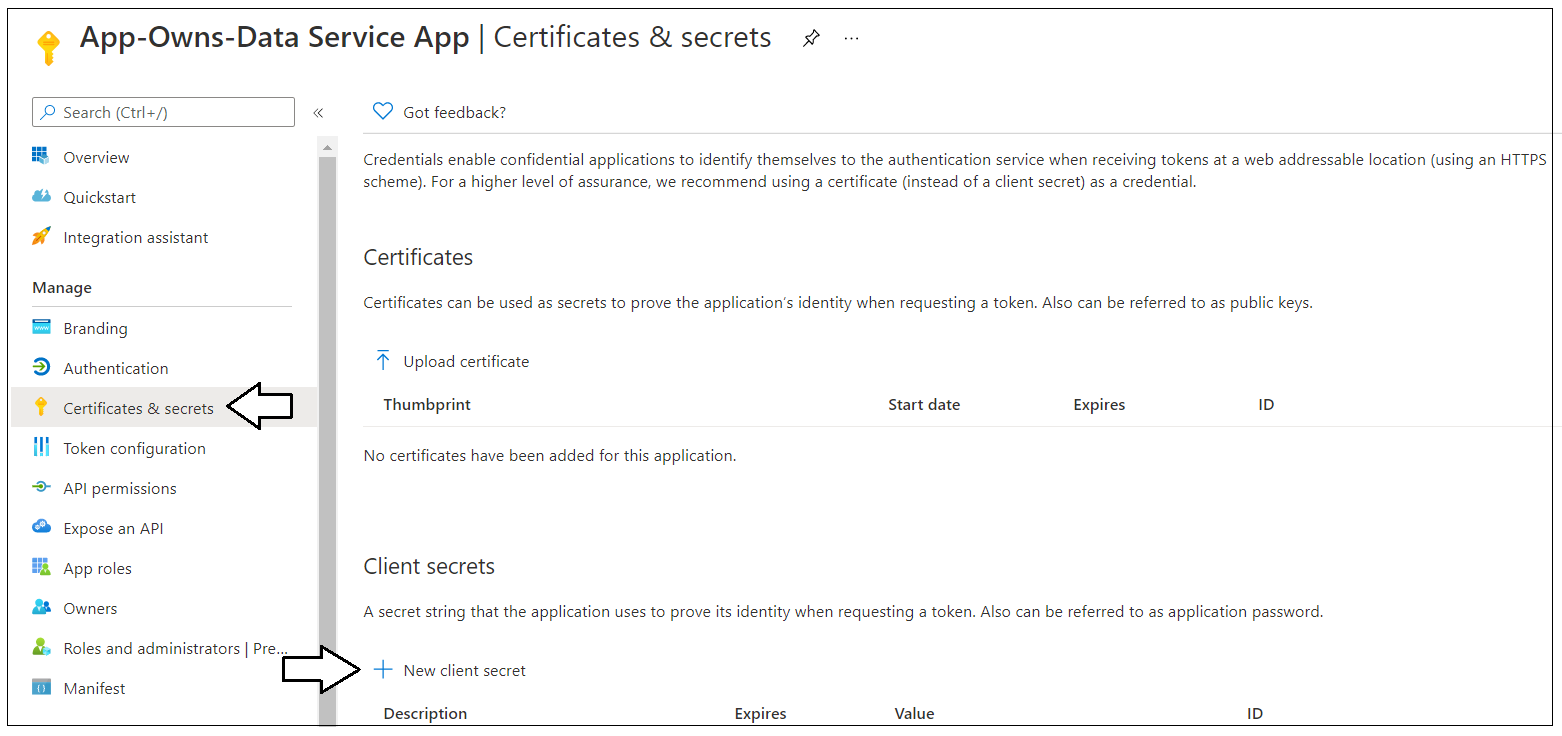
Next, repeat the same step by copying the **Tenant ID** and copying that into the text document as well.



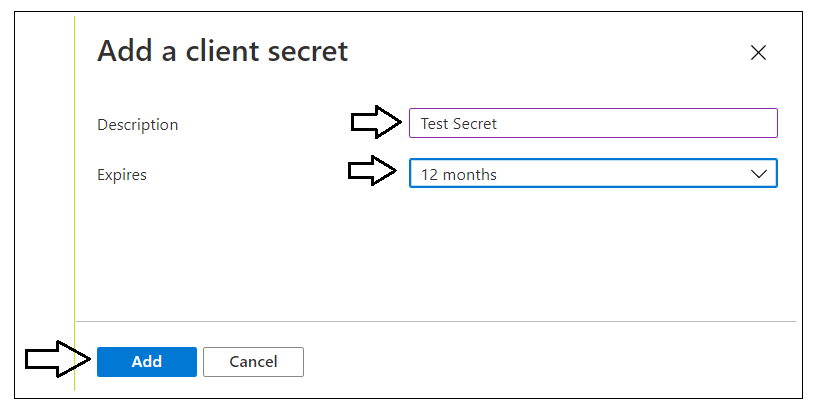
Your text document should now contain the **Client ID** and **Tenant ID** as shown in the following screenshot.



Next, you need to create a Client Secret for the application. Click on the **Certificates & secrets** link in the left navigation to move to the **Certificates & secrets** page. On the **Certificates & secrets** page, click the **New client secret** button as shown in the following screenshot.



In the **Add a client secret** dialog, add a **Description** such as **Test Secret** and set **Expires** to any value you'd like from the dropdown list. Click the **Add** button to create the new Client Secret.

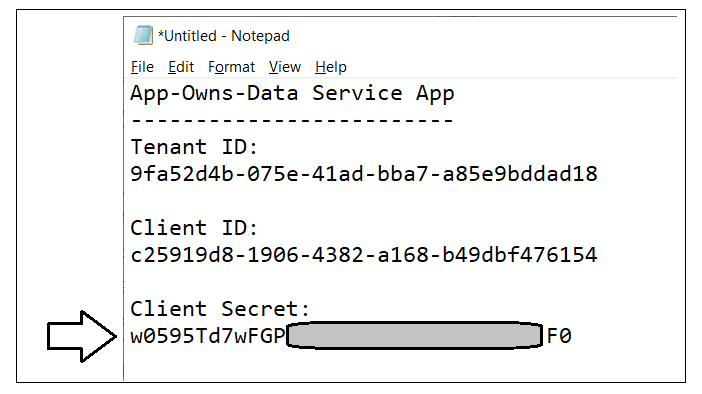


Once you have created the Client Secret, you should be able to see its **Value** in the **Client secrets** section. Click on the **Copy to clipboard** button to copy the **Value** for the Client Secret into the clipboard.

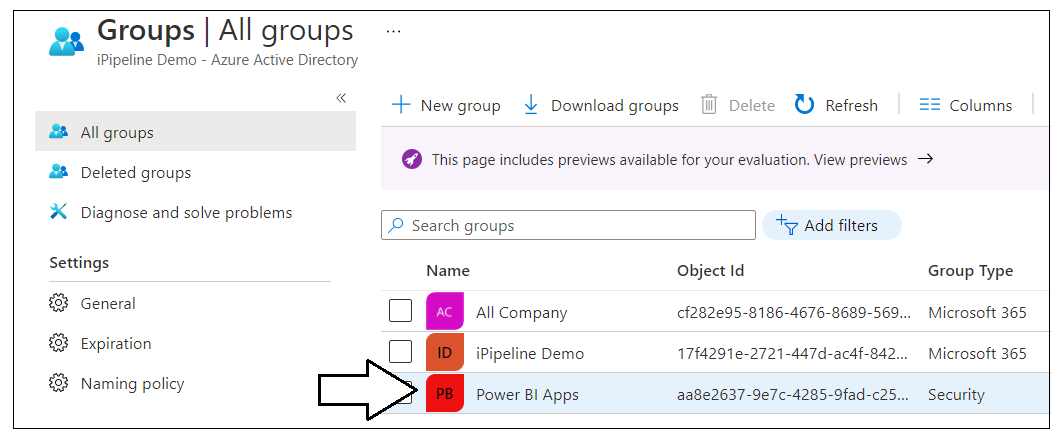
[Graphical user interface, text, application, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image21.png)

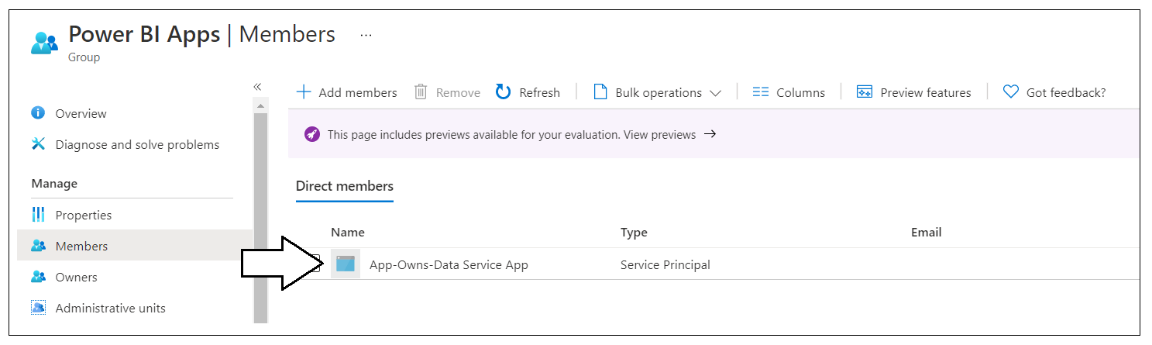
Paste the **Client Secret** into the same text document with the **Client ID** and **Tenant ID**.



Last thing is to add the service principal for this app to Azure AD Security group named **Power BI Apps**.



Navigate to the **Members** page for the **Power BI Apps** security group using the **Members** link in the left navigation. Add the Azure AD application named **App-Owns-Data Service App** as a group member.



You have now complete the registration of the first Azure AD application named **App-Owns-Data Service App**. This is the application will be used to authenticate as a service principal which will be used to call the Power BI Service API. The **App-Owns-Data Service App** will also be used to authenticate administrative users who needs to use the **AppOwnsDataAdmin** application.

In the next section, you will create a new Azure AD application named **App-Owns-Data Client App**. This Azure AD application will be used to secure the custom web API exposed by **AppOwnsDataWebApi**. The **AppOwnsDataClient** application will be configured to use this Azure AD application to authenticate it users and to acquire access tokens in the browser to make secure API calls on **AppOwnsDataWebApi**.

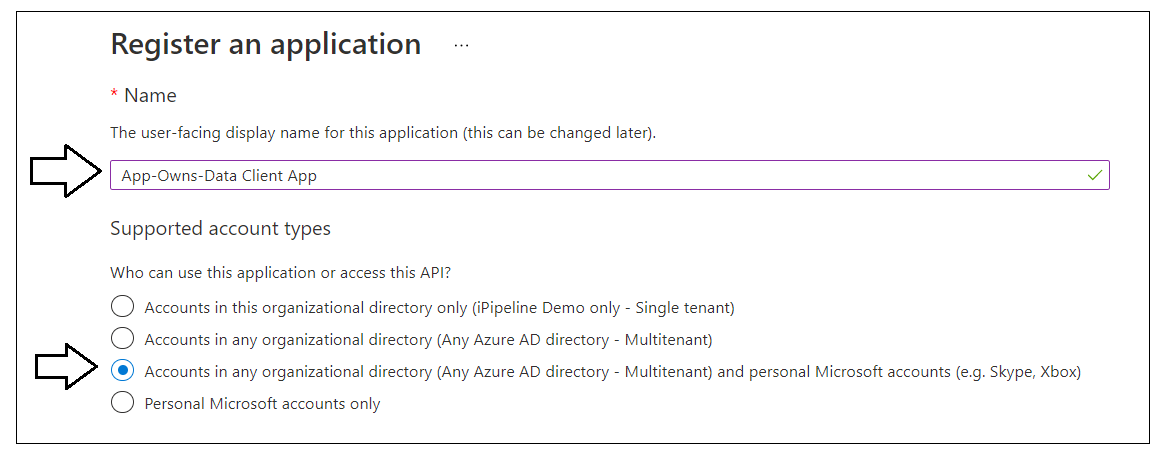
### Create the Azure AD Application for the ****App-Owns-Data Client App****

Navigate to the [App registration](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps) page in the Azure portal and click the **New registration** link.

[Graphical user interface, text, application, email

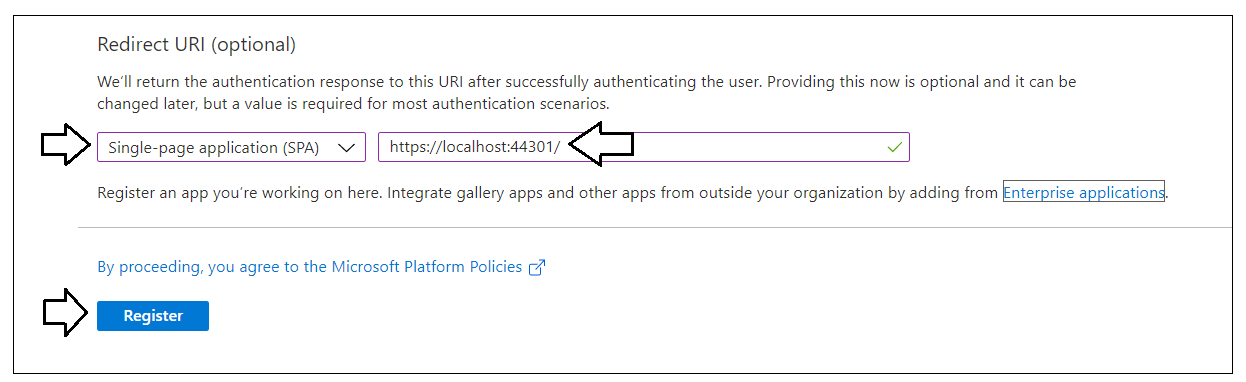
Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image12.png)

On the **Register an application** page, enter an application name of **App-Owns-Data Client App** and change **Supported account types** to **Accounts in any organizational directory and personal Microsoft accounts**.

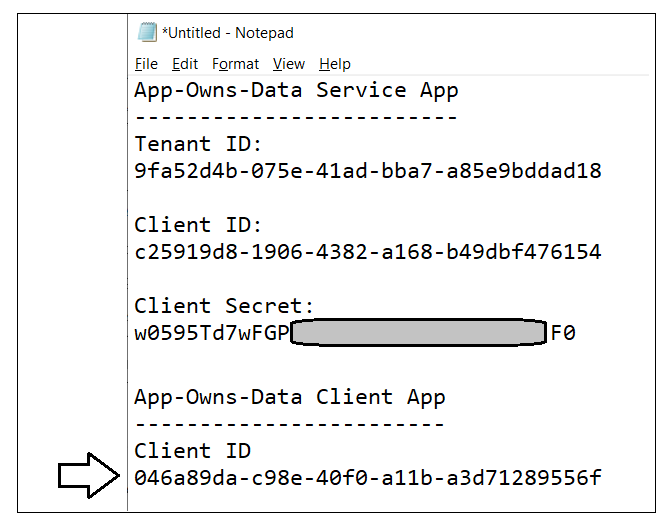


Complete the following steps in the **Redirect URI** section.

1. Change the setting of the dropdown box to **Single page application (SPA)**
2. Enter a Redirect URI of <https://localhost:44301/>.
3. Click the **Register** button to create the new Azure AD application.

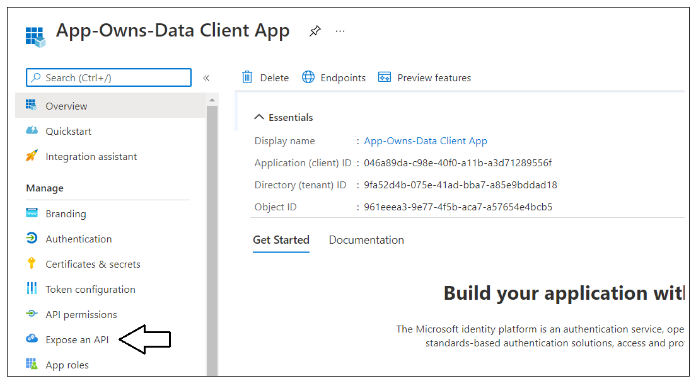


After creating a new Azure AD application in the Azure portal, you should see the Azure AD application overview page which displays the **Application ID**. Copy the **Client ID** (aka Application ID) and paste it into a text document so you can use it later in the setup process. Note that this **Client ID** value will be used by **AppOwnsDataClient** project and the **AppOwnsDataWebApi** project to configure authentication with Azure AD.

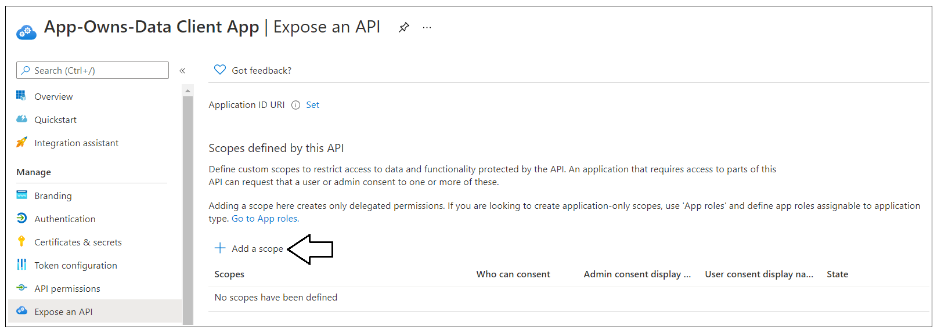


The **App-Owns-Data Client App** is being used to secure the API endpoint of **AppOwnsDataWebApi**. When creating an Azure AD application to secure a custom Web API like this it is necessary to create a custom scope for a delegated permission. As a developer, you can give a custom scope any name you'd like. In the solution for the **App-Owns-Data Starter Kit**, the custom scope will be given a name of **Reports.Embed**.

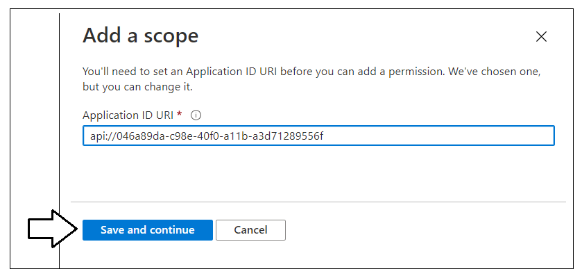
On the summary page for **App-Owns-Data Client App**, click the **Expose an API** link in the left navigation.



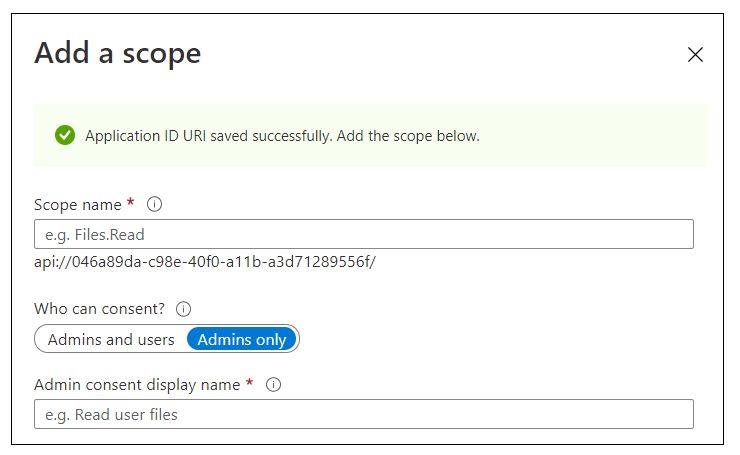
On the **Expose an API** page, click the **Add a scope** button.



On the **Add a scope** pane, you will be prompted to set an **Application ID URI** before you will be able to create a new scope. Accept the default setting of **api://** followed the application ID by clicking **Save and continue**.

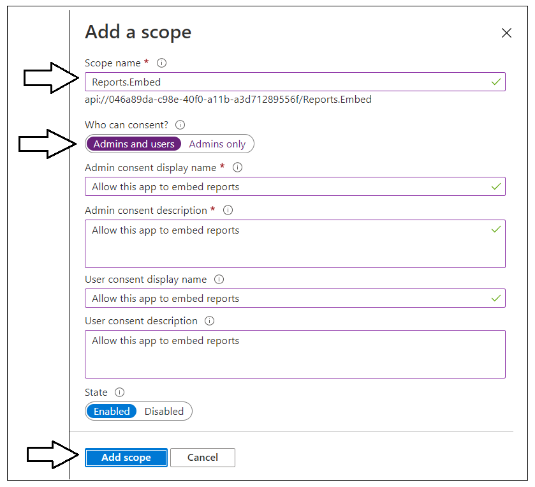


The **Add a scope** pane should present a form to enter data for the new scope.

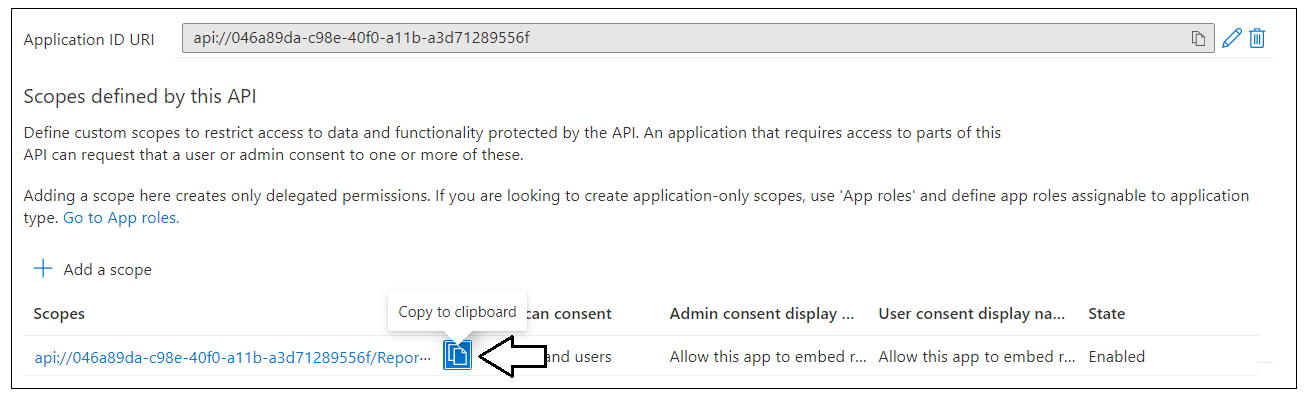


Fill out the data in the App a scope page using these steps.

1. Add a **Scope name** of **Reports.Embed**.
2. For the **Who can consent** setting, select **Admins and users**.
3. Fill in all display names and descriptions using the text shown in the following screenshot.
4. Click the **Add scope** button.



You should now see the new scopes in the **Scopes defined by this API** section. If you copy the scope value to the clipboard, you will see that is created in the format of **api://**[ApplicationID]**/Reports.Embed**.



## Open the App-Owns-Data Starter Kit solution in Visual Studio 2019

In order to run and test the **App-Owns-Data Starter Kit** solution on a developer workstation, you must install the .NET 5 SDK and Visual Studio 2019. While this document will walk through the steps of opening and running the projects of the **App-Owns-Data Starter Kit** solution using Visual Studio 2019, you can also use Visual Studio Code if you prefer that IDE. Here are links to download this software if you need them.

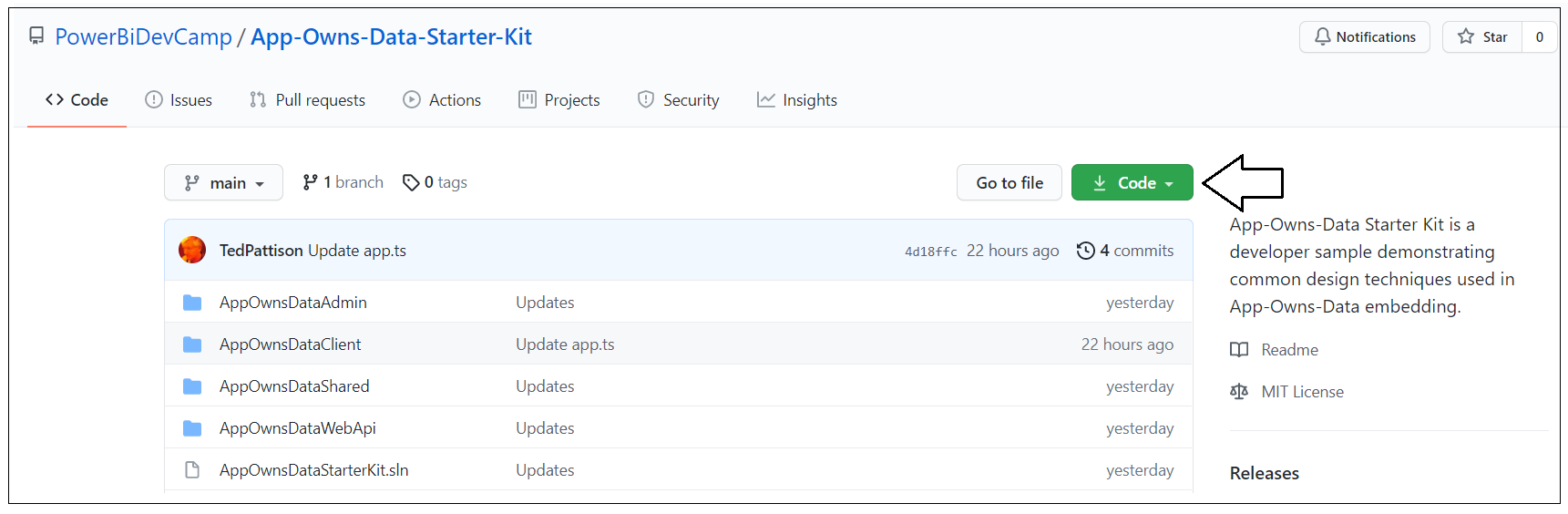
* .NET 5 SDK – [[download](https://dotnet.microsoft.com/download/dotnet/5.0)]
* Visual Studio 2019 – [[download](https://visualstudio.microsoft.com/downloads/)]
* Visual Studio Code – [[download](https://code.visualstudio.com/Download)]

### Download the Source Code

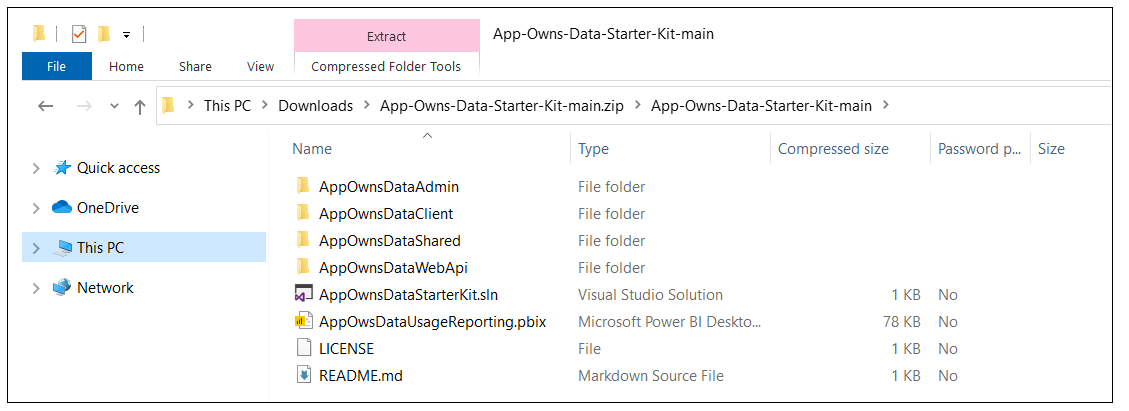
The project source files for the **App-Owns-Data Starter Kit solution** are maintained in a GitHub repository at the following URL.

<https://github.com/PowerBiDevCamp/App-Owns-Data-Starter-Kit>

On the home page for this GitHub repository is the **Code** dropdown menu which provides a few options for downloading the source files to your local machine.



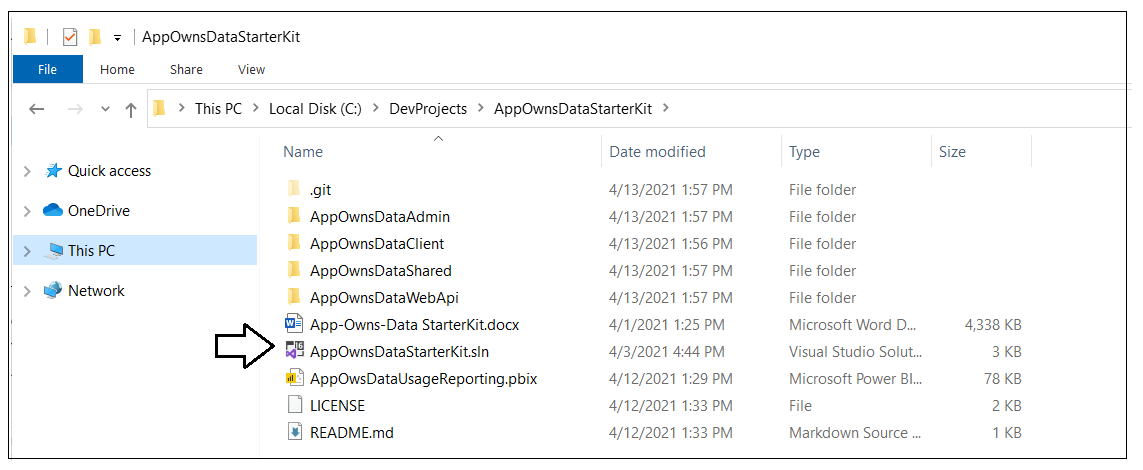
You can download the **App-Owns-Data Starter Kit** project source files in a single ZIP archive using [this link](https://github.com/PowerBiDevCamp/App-Owns-Data-Starter-Kit/archive/refs/heads/main.zip).



If you are familiar with the **git** utility, you can clone the project source files to your local developer workstation using the following **git** command:

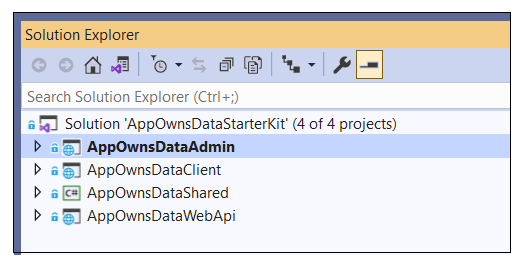
**git clone https://github.com/PowerBiDevCamp/App-Owns-Data-Starter-Kit.git**

Once you have downloaded the project source files for the **App-Owns-Data Starter Kit** solution to your developer workstation, you will see there is a top-level solution folder which contains folders for four projects named **AppOwnsDataAdmin**, **AppOwnsDataClient**, **AppOwnsDataShared** and **AppOwnsDataWebApi**. You can open the Visual Studio solution containing all four projects by double-clicking the solution file named **AppOwnsDataStarterKit.sln**.



### Open AppOwnsDataStarterKit.sln in Visual Studio 2019

Launch Visual Studio 2019 and use the **File > Open > Project/Solution** menu command to open the solution file named **AppOwnsDataStarterKit.sln**. You should see the four projects named **AppOwnsDataAdmin**, **AppOwnsDataClient**, **AppOwnsDataShared** and **AppOwnsDataWebApi**.



Here is a brief description of each of these projects.

* **AppOwnsDataAdmin**: ASP.NET MVC Web Application built using .NET 5
* **AppOwnsClient**: Single page application built using HTML, CSS and Typescript
* **AppOwnsDataShared**: Class library project used to generate **AppOwnsDataDB**
* **AppOwnsDataWebApi**: ASP.NET Web API which provides embedding data to **AppOwnsDataClient**

### Update the appsettings.json file of AppOwnsDataAdmin project

Before you can run the application in the Visual Studio debugger, you must update several important application settings in the **appsettings.json** file. Open the **appsettings.json** file and examine the JSON content inside. There is three important sections named **AzureAd**, **AppOwnsDataDB** and **DemoSettings**.



Inside the **AzureAd** section, update the **TenantId**, **ClientId** and **ClientSecret** with the data you collected when creating the Azure AD application named **App-Owns-Data Service App.**

[Text

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image28.png)

If you are using Visual Studio 2019, you should be able leave the database connection string the way it is with the **Server** setting of **(localdb)\\MSSQLLocalDB**. You can change this connection string to a different SQL Server instance if you'd rather create the project database named **AppOwnsDataDB** in a different location.

[Text

Description automatically generated with low confidence](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image29.png)

In the **DemoSettings** section there is a property named **AdminUser**. The reason that this property exists has to with you being able to see Power BI workspaces as they are created by a service principal. There is code in the **AppOwnsDataAdmin** application that will add the user specified by the **AdminUser** setting as a workspace admin any time a new Power BI workspace is created. This support has been included to make things much easier for you to see what's going on when you begin to run and test the application.

Update the **AdminUser** setting with your Azure AD account name so that you will be able to see all the Power BI workspaces created by this application.

[Graphical user interface, text, application, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image30.png)

### Create the ****AppOwnsDataDB**** database

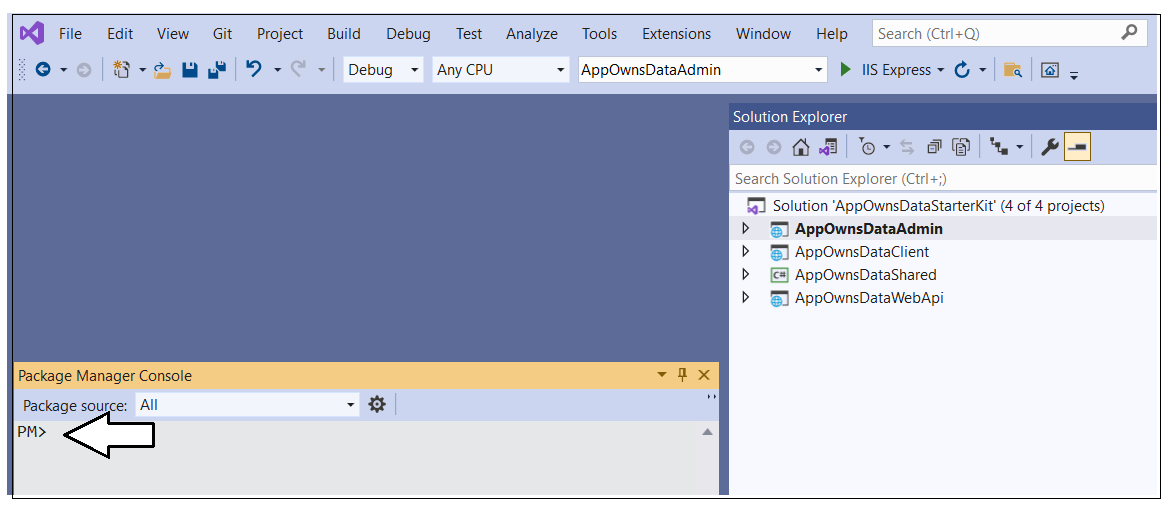
Before you can run the application in Visual Studio, you must create the project database named **AppOwnsDataDB**. This database schema has been created using the .NET 5 version of the Entity Framework. In this step, you will execute two PowerShell cmdlets provided by Entity Framework to create the database.

Open the Package Manager console using **Tools > NuGet Package Manager > Package Manager Console**.

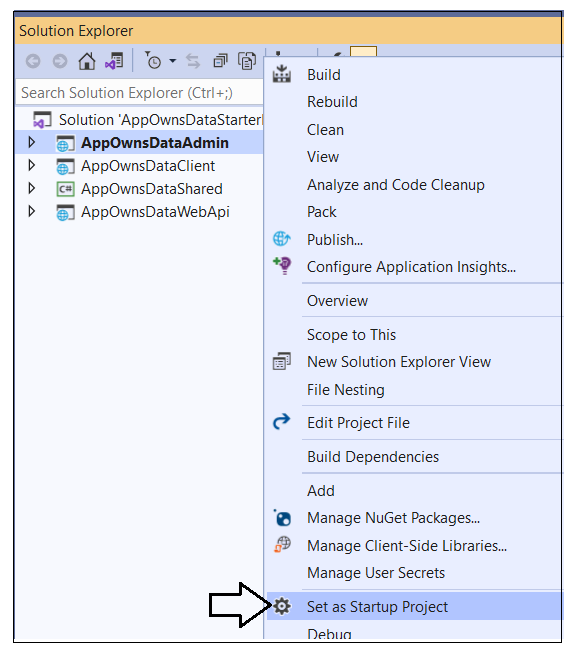
[Graphical user interface, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image34.png)

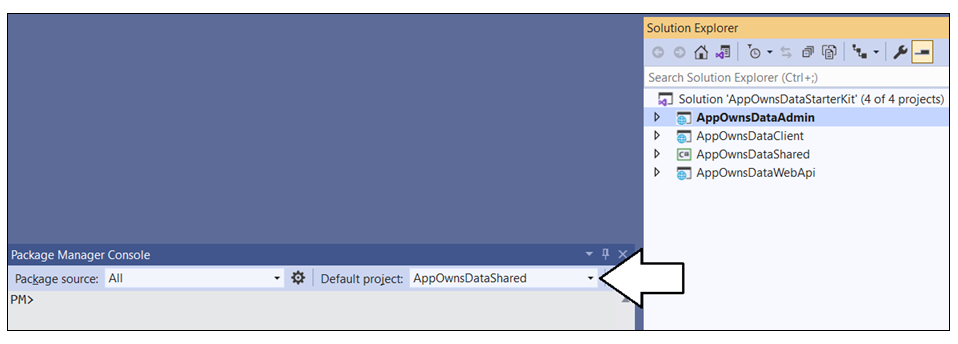
You should see the **Package Manager Console** command prompt where you can type and execute PowerShell commands.



Next, you must configure the **AppOwnsDataAdmin** project as the solution's startup project so that Entity Framework Core can retrieve the database connection string from that project's **appsettings.json** file to create the new database. You can accomplish that by right-click on the **AppOwnsDataAdmin** project in the solution explorer and select **Set as Start Project**.



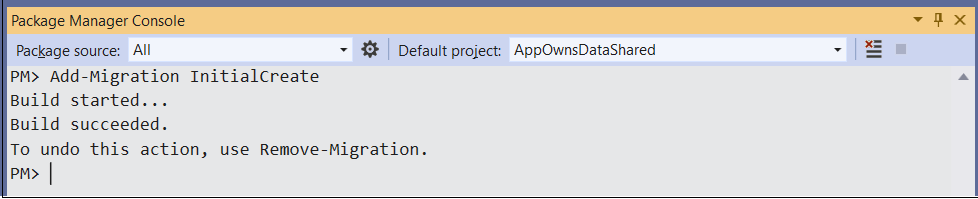
Inside the **Package Manager Console** window, set the **Default project** to **AppOwnsDataShared**.



Type and execute the following **Add-Migration** command to create a new migration in the project.

**Add-Migration InitialCreate**

The **Add-Migration** command should run without errors. If this command fails you might have to modify the database connection string in **appsettings.json**.



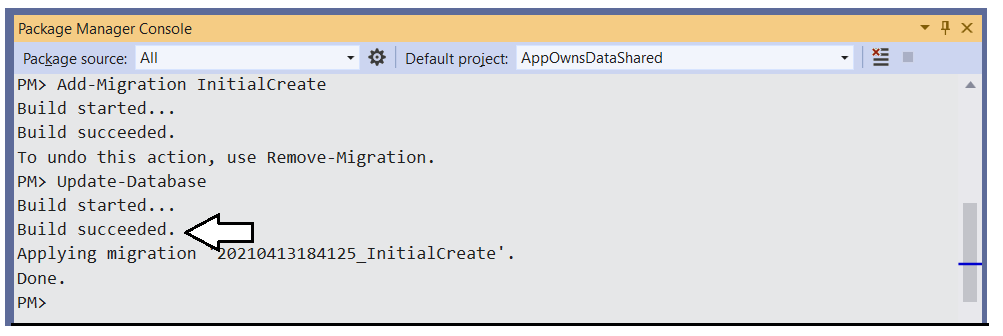
After running the **Add-Migration** command, you will see a new folder has been automatically created in the **AppOwnsDataShared** project named **Migrations** with several C# source files. There is no need to change anything in these source files but you can inspect what's inside them if you are curious how the Entity Framework Core does its work.



Return to the **Package Manager Console** and run the following **Update-Database** command to generate the database named **AppOwnsDataDB**.

**Update-Database**

The **Update-Database** command should run without errors and generate the **AppOwnsDataDB** database.

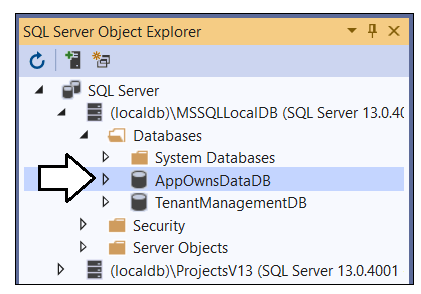


In Visual Studio, you can use the **SQL Server Object Explorer** to see the database that has just been created. Open the **SQL Server Object Explorer** by invoking the **View >** **SQL Server Object Explorer** menu command.

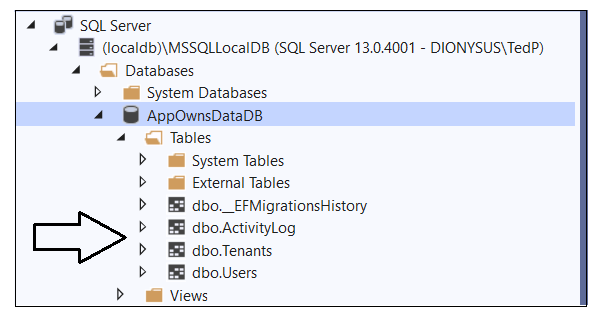
[Graphical user interface, text, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image39.png)

Expand the **Databases** node for the server you're using and verify you see the **AppOwnsDataDB** database.



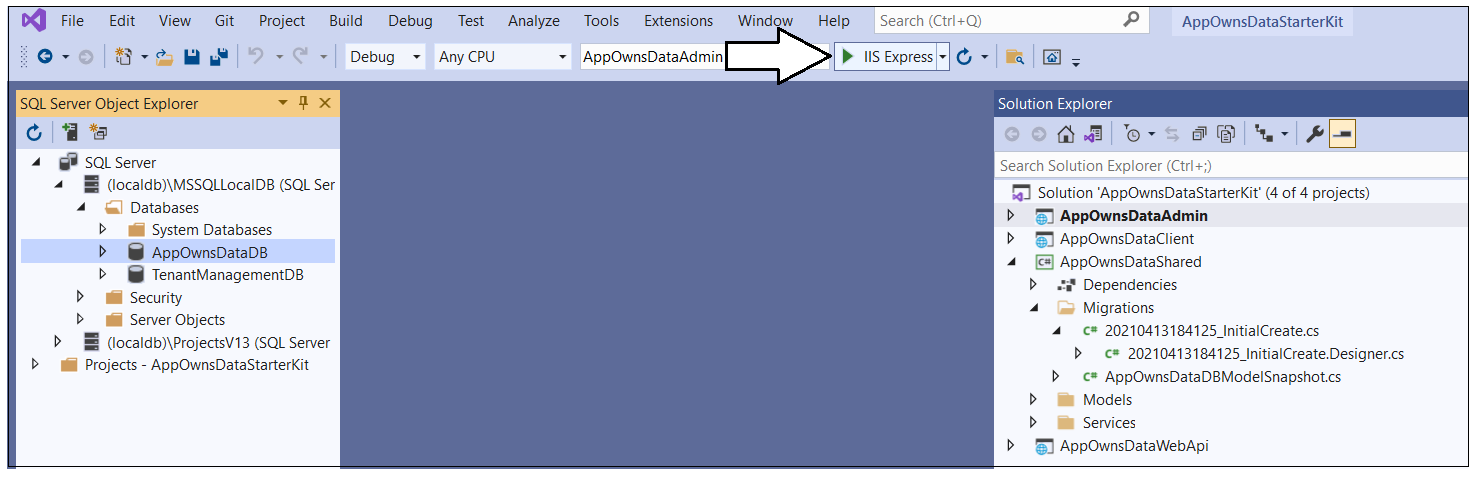
If you expand the **Tables** node, you should see the three tables named **ActivityLog, Tenants and Users**.



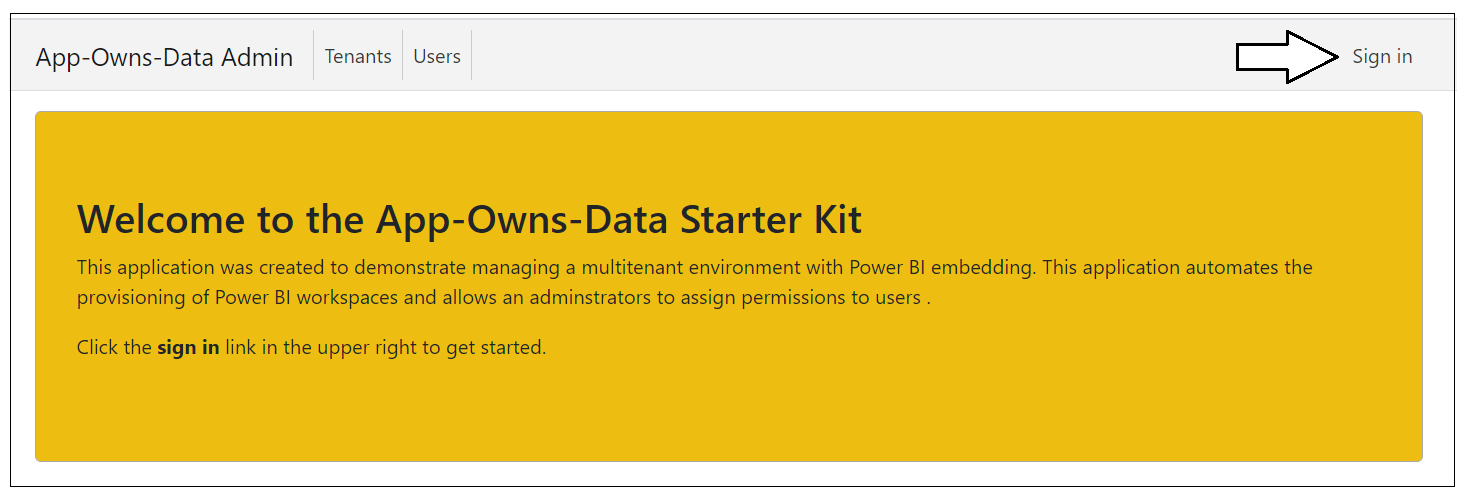
With **AppOwnsDataDB** set up, you're ready to run and test **AppOwnsDataAdmin** in Visual Studio 2019.

## Test the AppOwnsDataAdmin Application

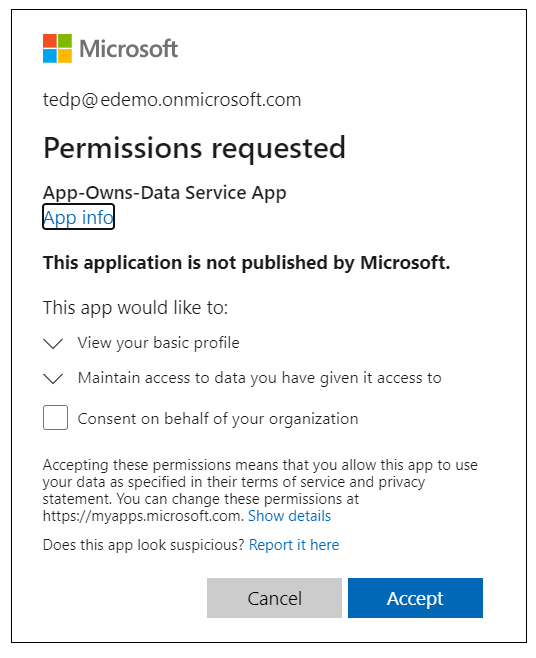
Launch the **AppOwnsDataAdmin** web application in the Visual Studio debugger by pressing the **{F5}** key or by clicking the Visual Studio **Play** button with the green arrow and the caption **IIS Express**.



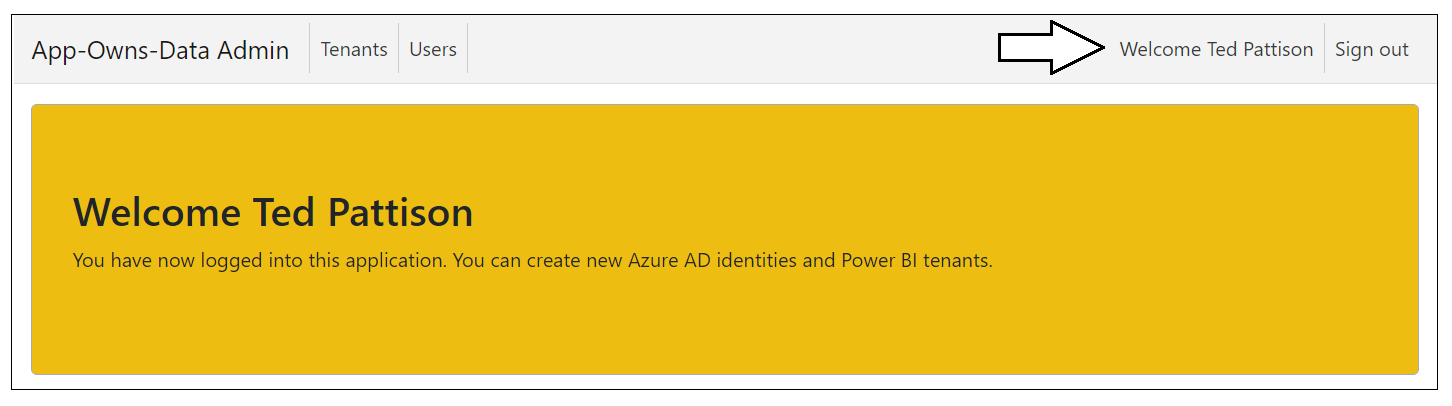
When the application starts, click the **Sign in** link in the upper right corner to begin the user login sequence.



The first time you authenticate with Azure AD, you'll be prompted with the **Permissions requested** dialog asking you to accept the Permissions requested by the application. Click the **Accept** button to grant these permissions and continue.

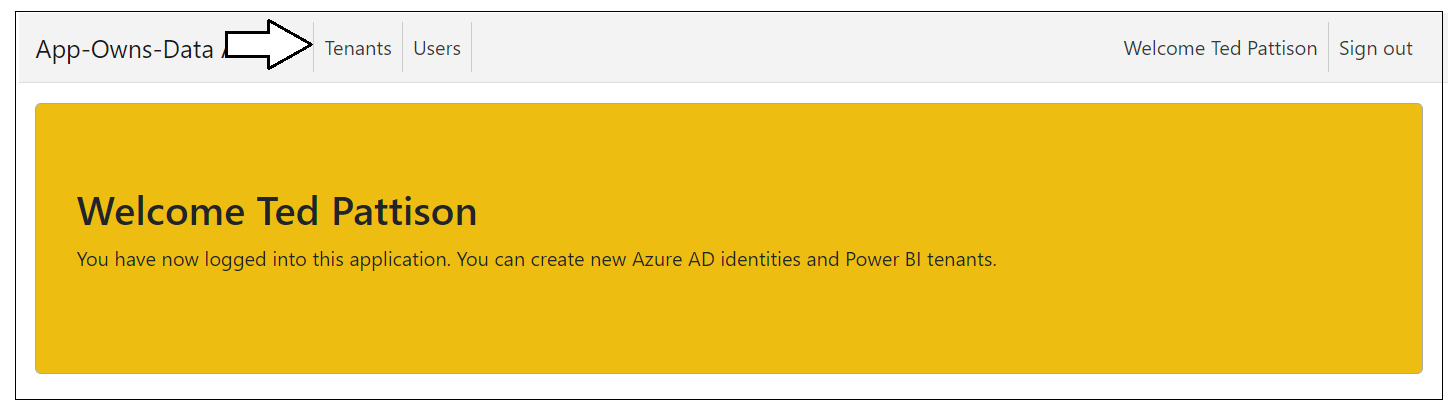


Once you have logged in, you should see your name in the welcome message.

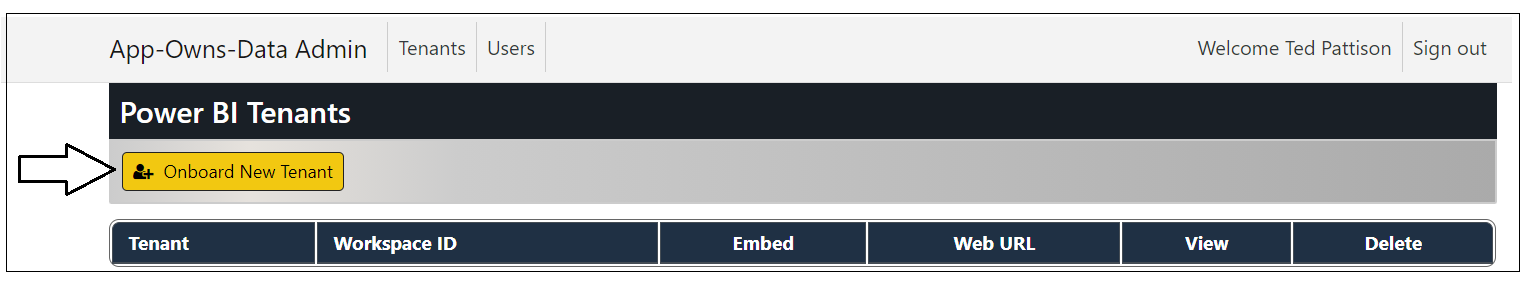


### Create New Customer Tenants

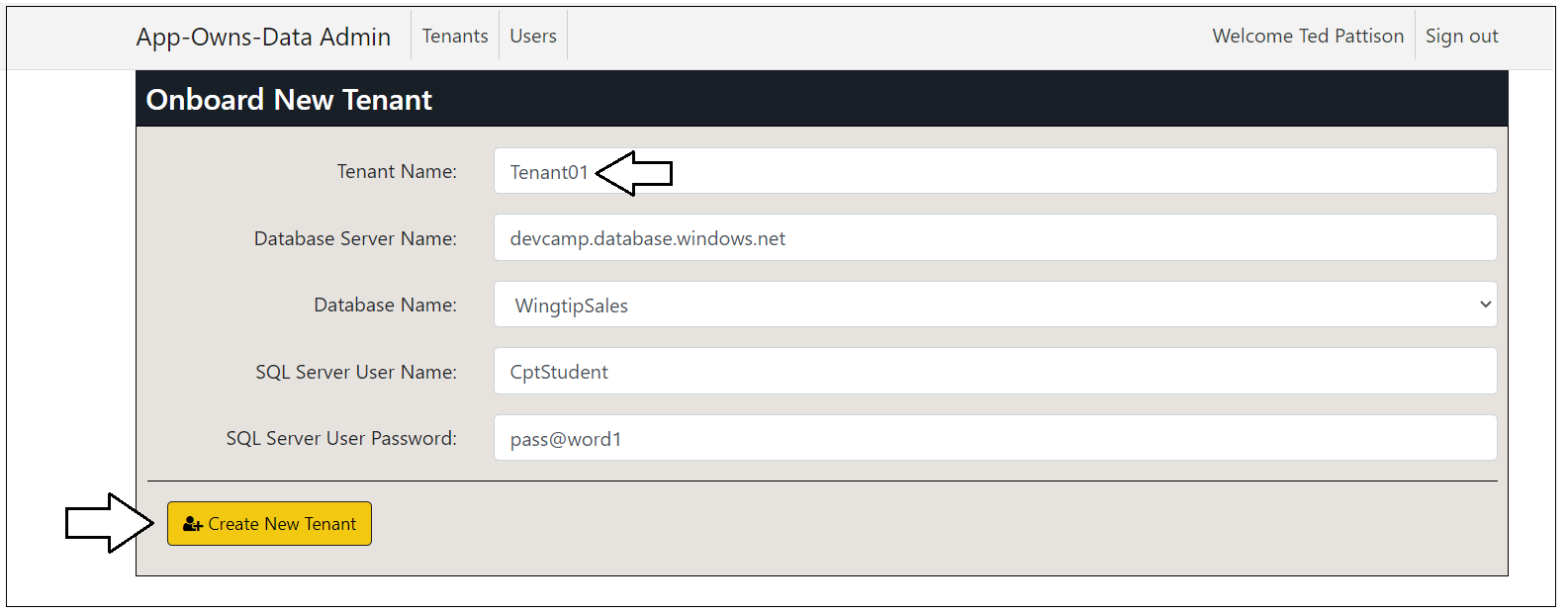
Start by creating a few new customer tenants. Click the **Tenants** link to navigate to the **Tenants** page.



Click the **Onboard New Tenant** button to display the **Onboard New Tenant** page.



When you open the **Onboard New Tenant** page, it will automatically populate the **Tenant Name** textbox with a value of **Tenant01**. You can create the first tenant using the default value for **Tenant Name** or supply a different name. Click to **Create New Tenant** button to begin the process of creating a new customer tenant.



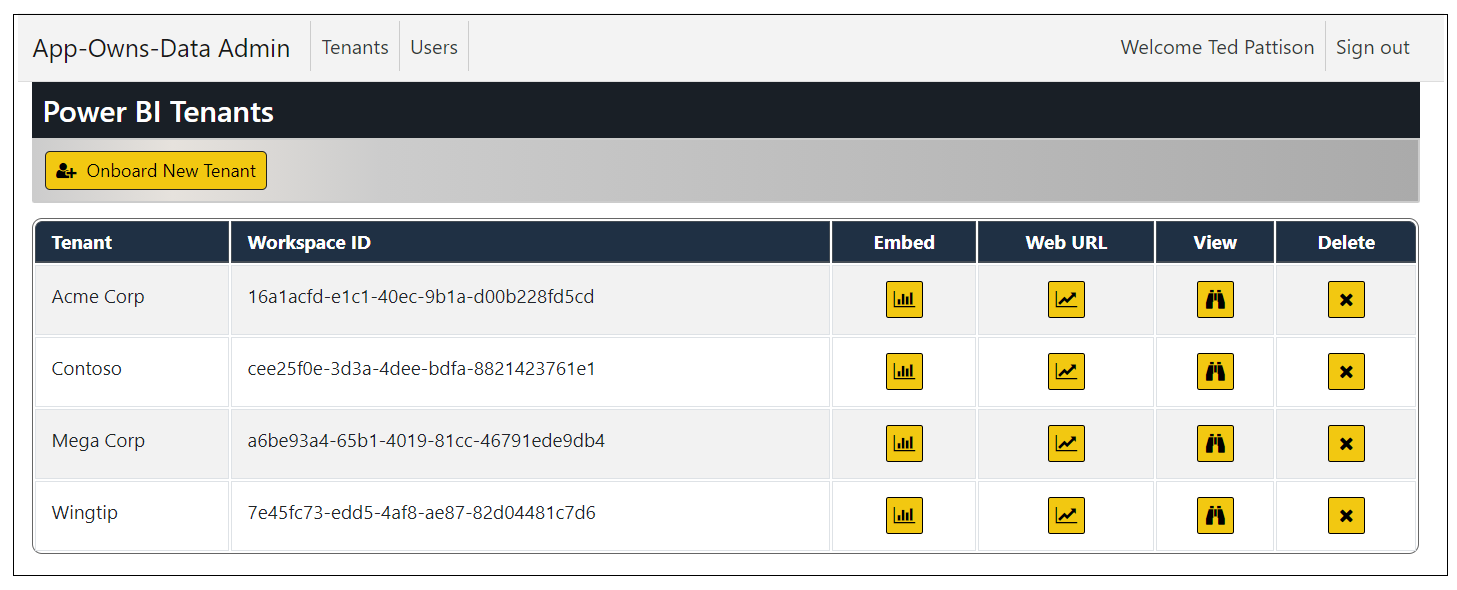
After a few seconds, you should see the new customer tenant has been created.

Click the **Onboard New Tenant** button again to create a second tenant.

This time, select a different database for **Database Name** and then click **Create New Tenant**.

You should now have two customer tenants. Note they each tenant has its own unique workspace ID.

Follow the same steps to create two more customer tenants.



Let's review what's going on behind the scenes. As you create a new customer tenant, the **AppOwnsDataAdmin** application uses the Power BI REST API to implement the following onboarding logic.

* Create a new Power BI workspace
* Upload a [template PBIX file](https://github.com/PowerBiDevCamp/TenantManagement/raw/main/TenantManagement/wwwroot/PBIX/DatasetTemplate.pbix) to create the Sales dataset and the Sales report
* Update dataset parameters on Sales dataset to point to this customer's database
* Patch credentials for the SQL datasource used by the Sales dataset
* Start a refresh operation on the Sales database

If you want to inspect the C# code in **AppOwnsDataAdmin** that that implements this logic using the Power BI .NET SDK, you can examine the **OnboardNewTenant** method in the source file named [PowerBiServiceApi.cs](https://github.com/PowerBiDevCamp/App-Owns-Data-Starter-Kit/blob/main/AppOwnsDataAdmin/Services/PowerBiServiceApi.cs).

The **AppOwnsDataAdmin** application also create a new record in the **Tenants** table of the **AppOwnsDataDB** database. Note that the application identity associated with this customer tenant is tracked in the **Owner** column.

Click on the **View** button for a specific tenant on the **Power BI Tenants** page to drill into the **Tenant Details** page.

The **Tenant Details** page displays Power BI workspace details including its members, datasets and reports.

Click on the back arrow to return to the **Power BI Tenants** page.

If you're interested, you can examine the details of other tenants as well.

### Embed Reports

Now it's time to make use of the **AppOwnsDataAdmin** application's ability to embed reports. Click the Embed button of a customer tenant to navigate to the **Embed** page and view the **Sales** report.

The **AppOwnsDataAdmin** application should now present a view with the embedded report. Navigate back to the **Power BI Tenants** page and click on the **Embed** button for the first customer tenant.

You should now see a page with an embedded report for that tenant. Click on the back arrow button to return to the **Tenants** page.

1. Now test clicking the **Embed** button for other customer tenants. As you can see, the **AppOwnsDataAdmin** application has the ability to embed reports from any f the tenant workspaces it has created.

### Inspect the Power BI Workspaces

If you're curious about what's been created in Power BI, you can see for yourself by navigating to the Power BI Service portal at [https://app.powerbi.com](https://app.powerbi.com/). You should be able to see and navigate to any of the Power BI workspaces that have been created by the **AppOwnsDataAdmin** application.

[A picture containing graphical user interface

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image70.png)

Navigate to one of these workspaces such as **Tenant01**.

[Graphical user interface, text, email

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image71.png)

Drill into the **Setting** page for the dataset named **Sales**.

[Graphical user interface, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image72.png)

You should be able to verify that the **Sales** dataset has been configured by one of the Azure AD applications that was created by the **AppOwnsDataAdmin** application. You should also be able to see the **Last refresh succeeded** message for the dataset refresh operation that was started by the **AppOwnsDataAdmin** as part of its tenant onboarding logic.

[Graphical user interface, application

Description automatically generated](https://github.com/PowerBiDevCamp/TenantManagement/blob/main/Images/ReadMe/media/image73.png)

## Test the AppOwnsDataClient Application

xxxx

### Configure the WebAPI appsettings.

This concludes the walkthrough of the **AppOwnsDataAdmin** application.

### Configure the AppOwnsDataClient application

### Run the AppOwnsDataClient application

### Assign User Permissions

### Create and Edit Reports using AppOwnsDataClient

## Use the Activity Log to monitor usage and report performance

### Inspect the Usage Data in AppOwnsDataDB

### Inspect Usage Data using AppOwsDataUsageReporting.pbix