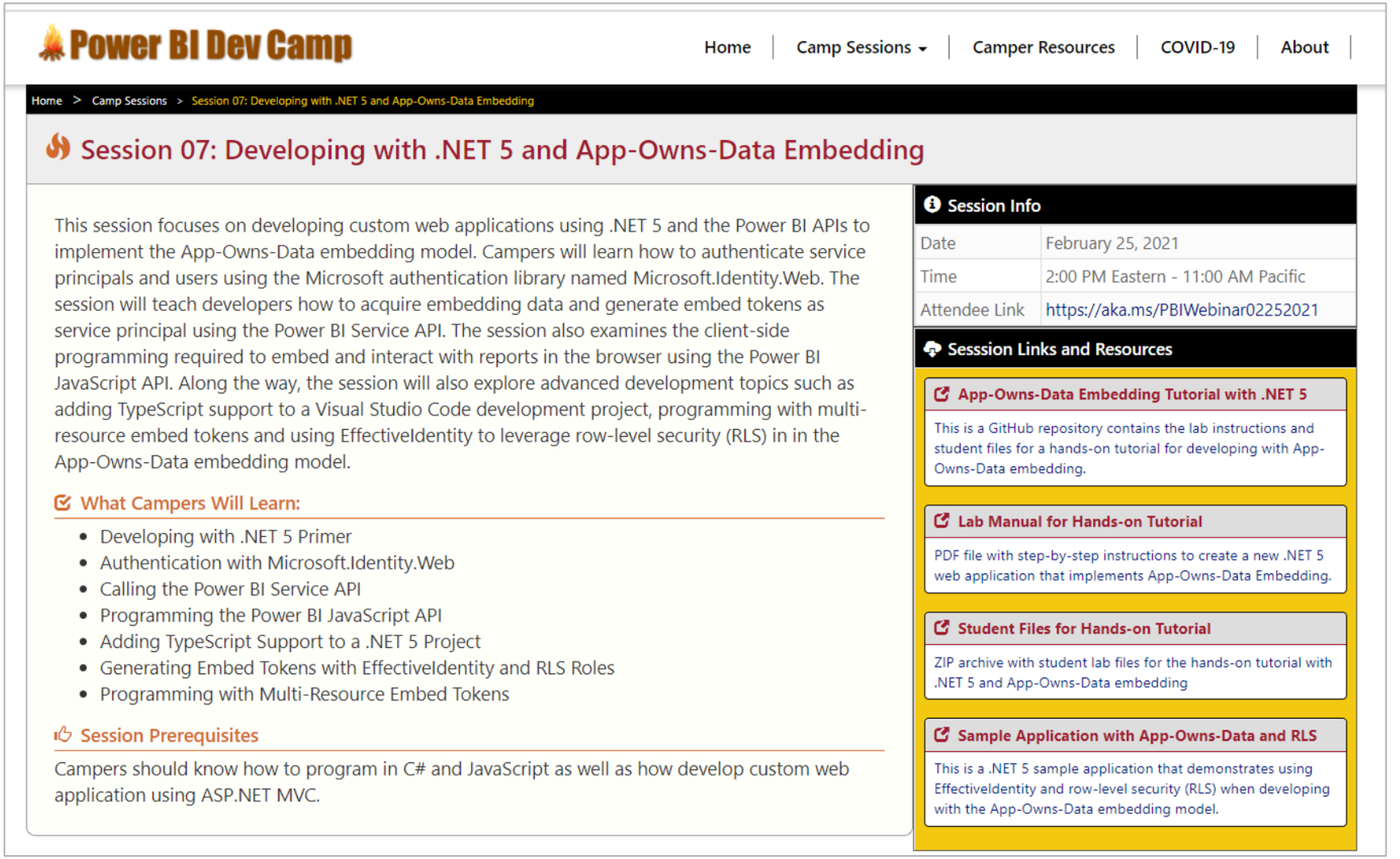
Developing with .NET 5 and App-Owns-Data Embedding

Ted Pattison
Principal Program Manager
Customer Advisory Team (CAT) at Microsoft

Welcome to Power BI Dev Camp

- Power BI Dev Camp Portal - https://powerbidevcamp.net

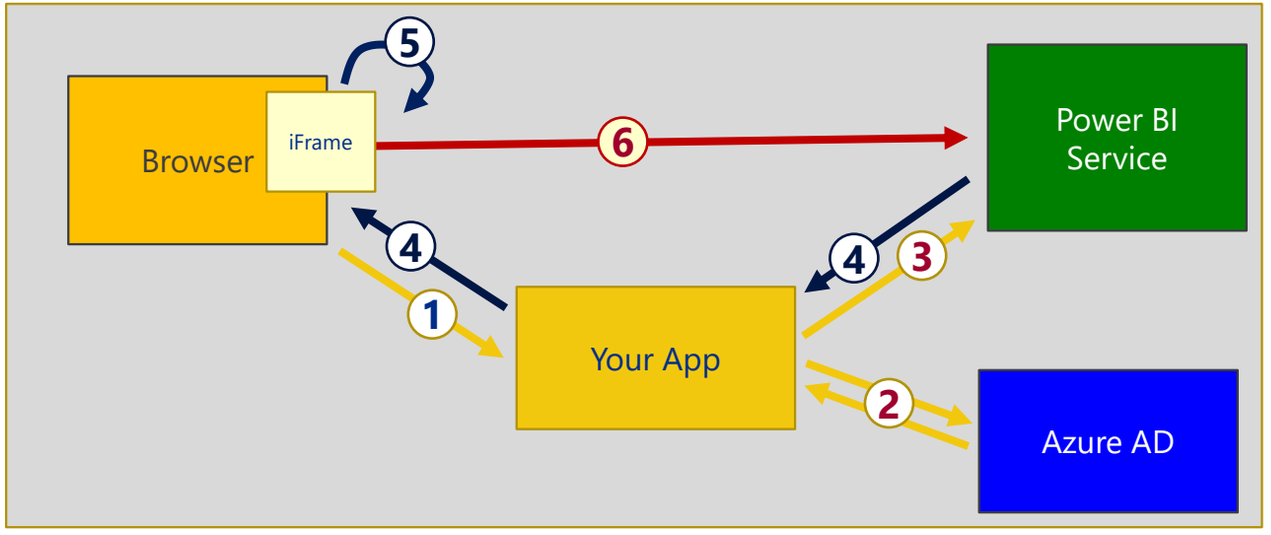


Agenda

- ➢ Developing for Power BI with .NET 5
- ▪ Introducing `Microsoft.Identity.Web`
- ▪ Calling to Power BI as Service Principal
- ▪ Programming the Power BI JavaScript API
- ▪ Adding TypeScript Support to a .NET 5 Project
- ▪ Programming with Multi-Resource Embed Tokens
- ▪ Integrating RLS using EffectiveIdentity

Power BI Embedding – The Big Picture

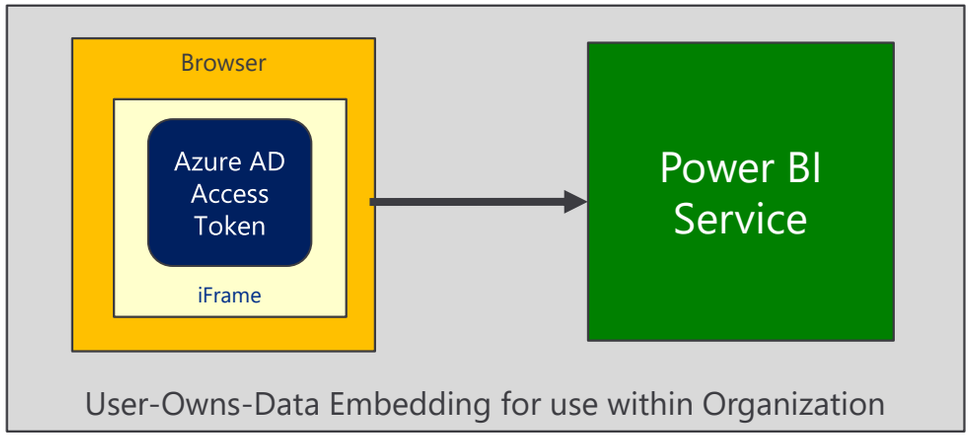
- User launches your app using a browser
- App authenticates with Azure Active Directory and obtains access token
- App uses access token to call to Power BI Service API
- App retrieves data for embedded resource and passes it to browser.
- Client-side code uses Power BI JavaScript API to create embedded resource
- Embedded resource session created between browser and Power BI service



Choosing the Correct Embedding Model

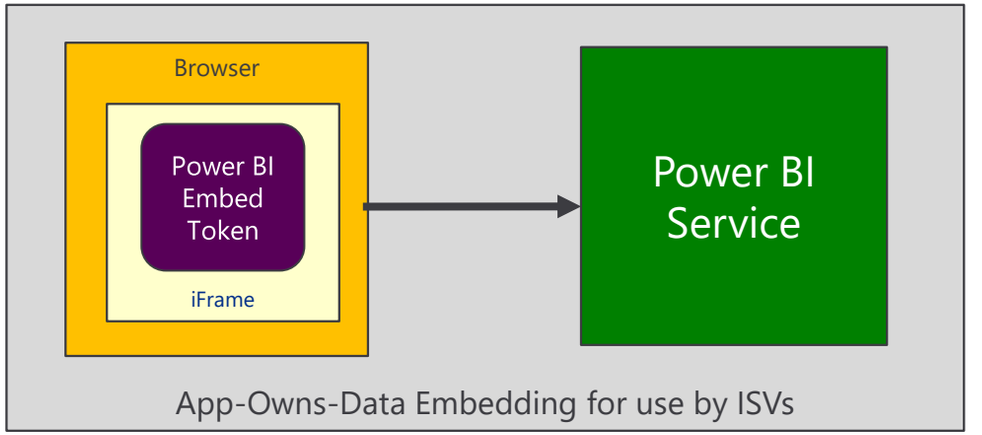
User-Owns-Data Embedding

- All users require a Power BI license
- Useful in corporate environments
- App authenticates as current user
- Your code runs with user's permissions
- User's access token passed to browser



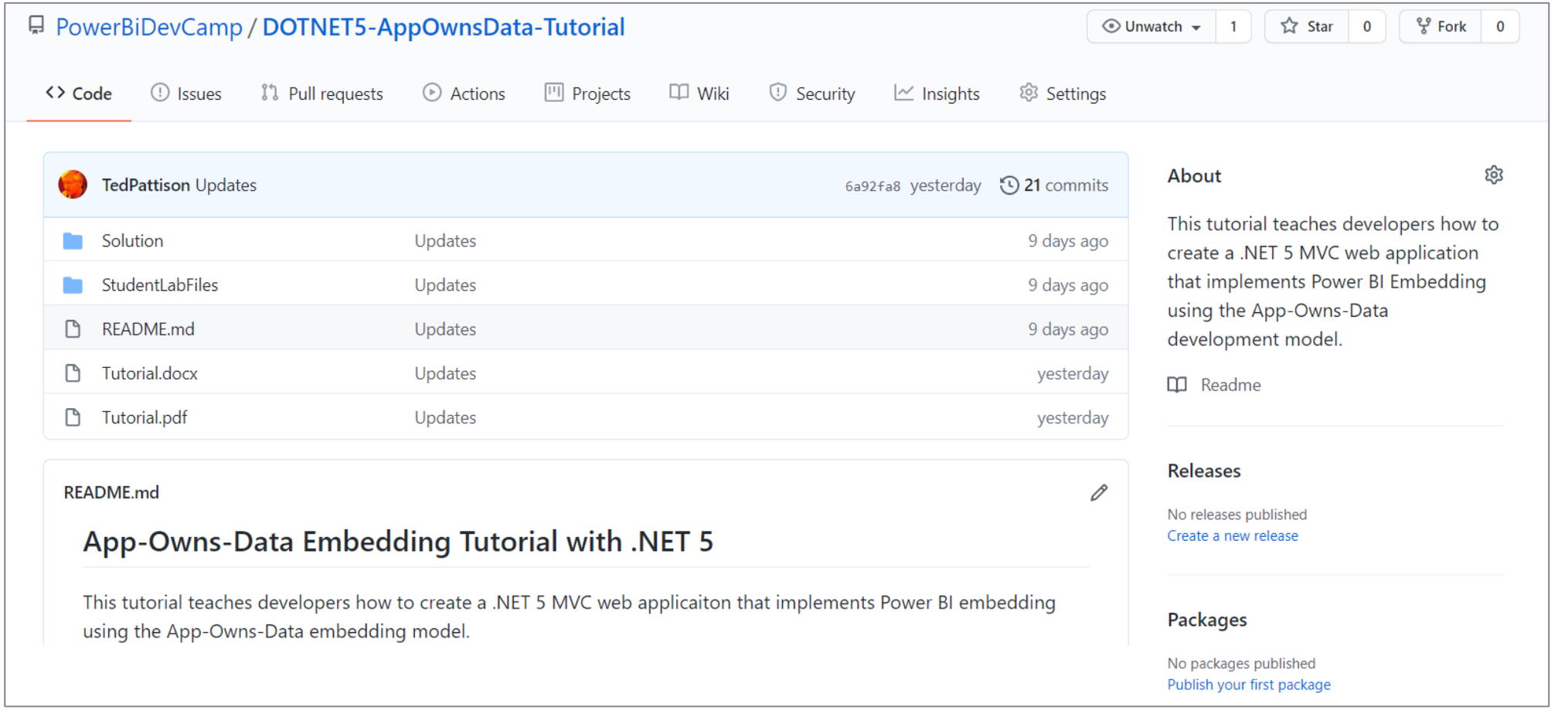
App-Owns-Data Embedding

- No users require Power BI license
- Useful in commercial applications
- App authenticates with app-only identity
- Your code runs with admin permissions
- Embed token passed to browser

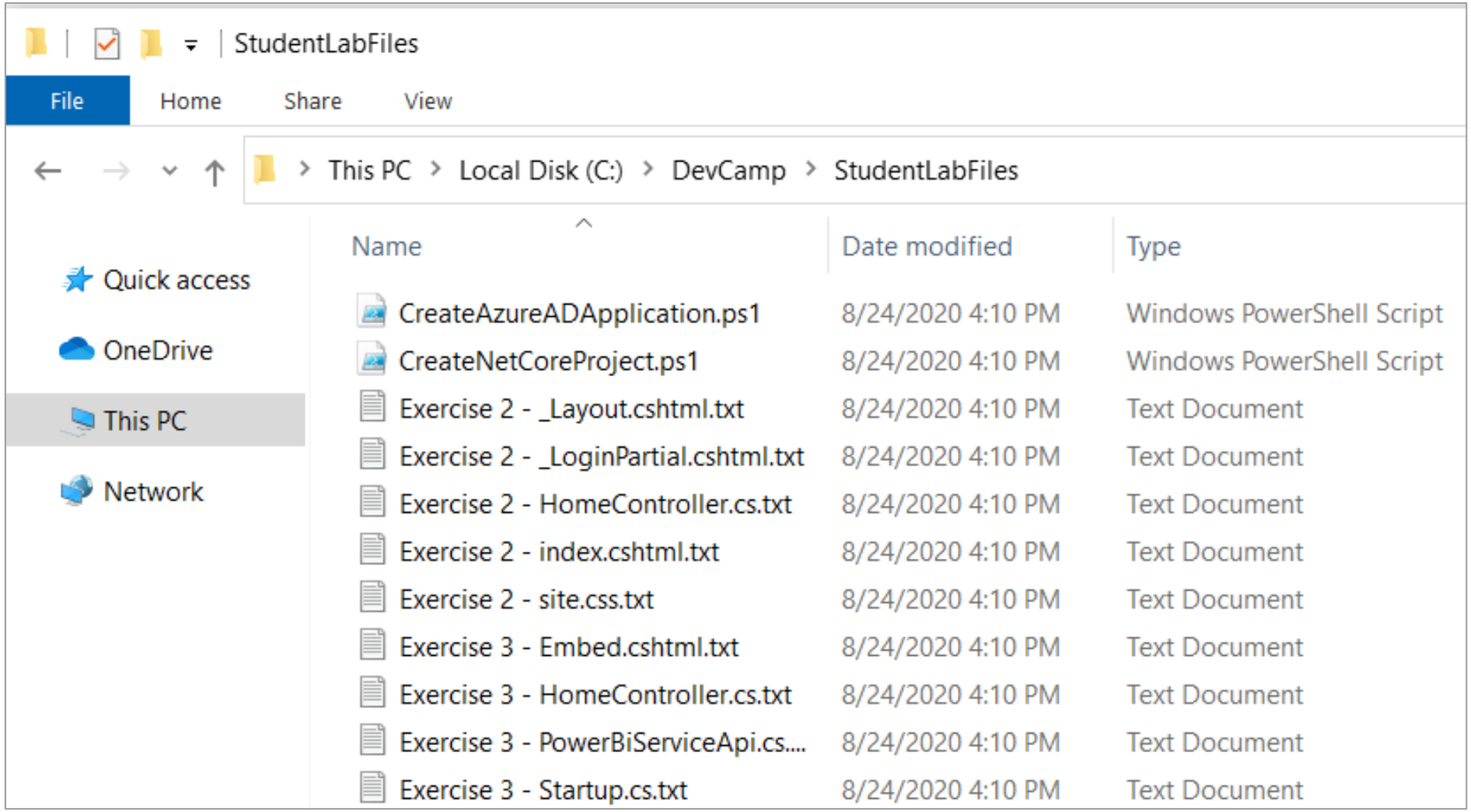


App-Owns-Data .NET 5 Tutorial

- Stored in a GitHub repository for easy download
  - https://github.com/PowerBiDevCamp/DOTNET5-AppOwnsData-Tutorial



Copying-and-Pasting Code

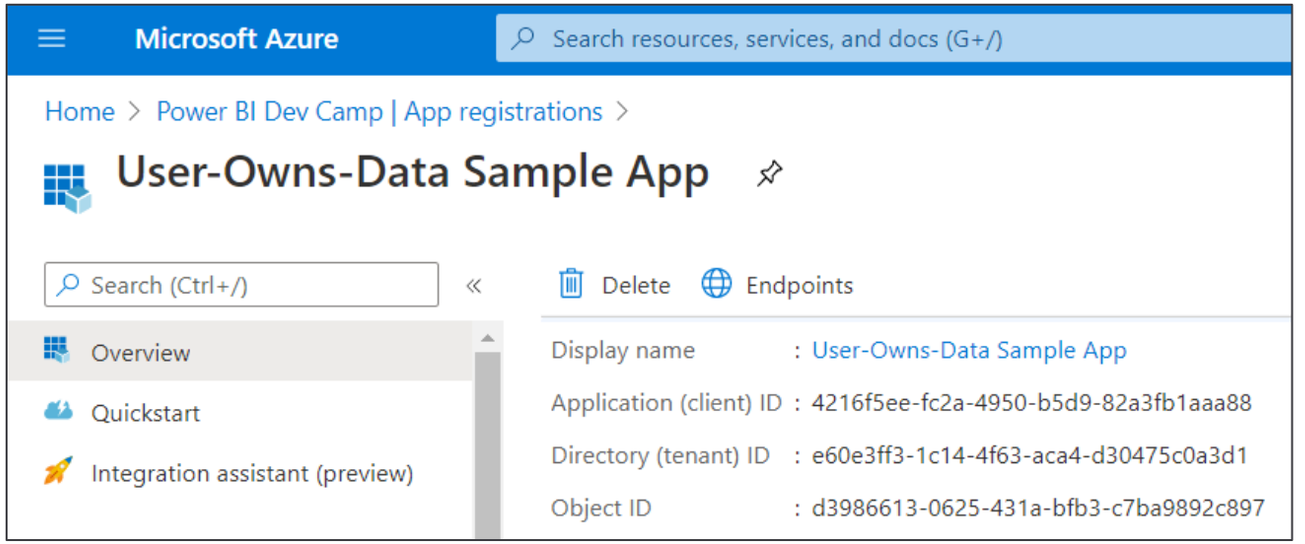


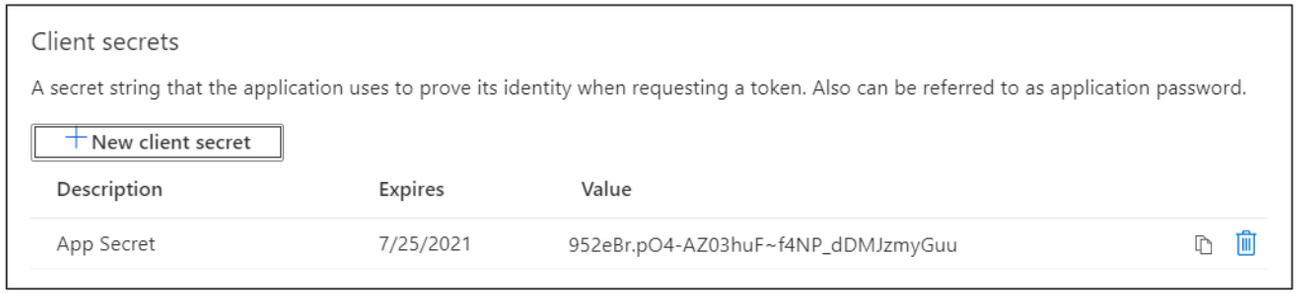
Tutorial Exercise Flow

- Exercise 1: Create a New Azure AD Application
- Exercise 2: Create a .NET 5 Project for a Secure Web Application
- Exercise 3: Call the Power BI Service API
- Exercise 4: Embedding a Report using powerbi.js
- Exercise 5: Adding TypeScript Support to a .NET 5 Project
- Exercise 6: Creating a View Model for App Workspaces

Creating Azure AD Application with PowerShell

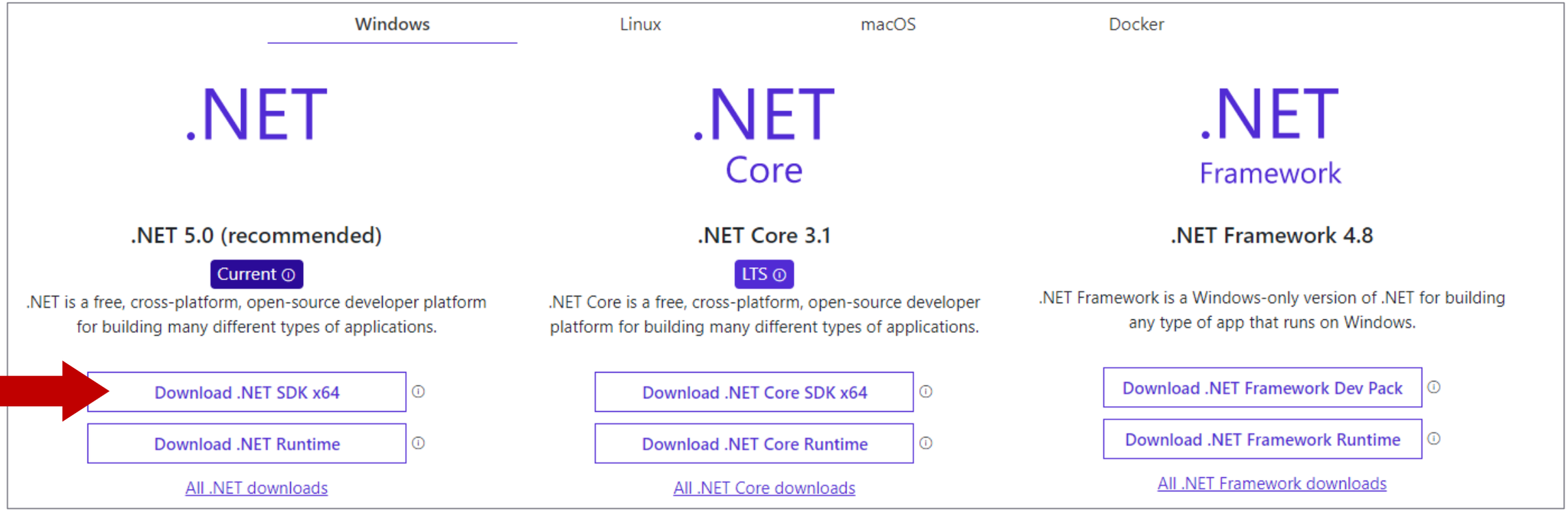
```powershell
$authResult = Connect-AzureAD
$userAccountId = $authResult.Account.Id
$user = Get-AzureADUser -ObjectId $userAccountId

$appDisplayName = "User-Owns-Data Sample App"
$replyUrl = "https://localhost:44300/signin-oidc"

# create app secret
$newGuid = New-Guid
$appSecret = ([System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(($newGuid))))+"="
$startDate = Get-Date
$passwordCredential = New-Object -TypeName Microsoft.Open.AzureAD.Model.PasswordCredential
$passwordCredential.StartDate = $startDate
$passwordCredential.EndDate = $startDate.AddYears(1)
$passwordCredential.KeyId = $newGuid
$passwordCredential.Value = $appSecret

# create Azure AD Application
$aadApplication = New-AzureADApplication `
                        -DisplayName $appDisplayName `
                        -PublicClient $false `
                        -AvailableToOtherTenants $false `
                        -ReplyUrls @($replyUrl) `
                        -Homepage $replyUrl `
                        -PasswordCredentials $passwordCredential

# assign current user as owner
$appId = $aadApplication.AppId
Add-AzureADApplicationOwner -ObjectId $aadApplication.ObjectId -RefObjectId $user.ObjectId
```





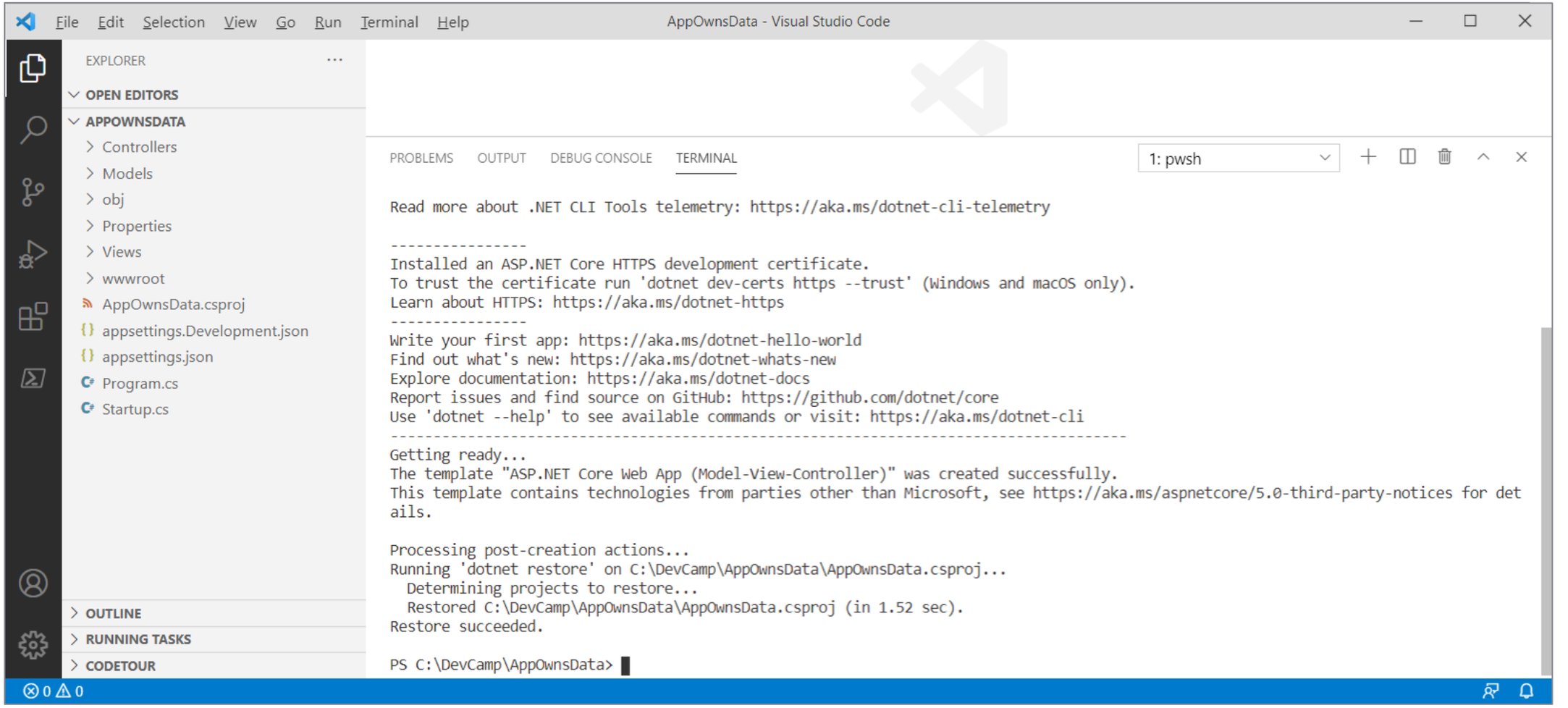
Install .NET 5 SDK

https://dotnet.microsoft.com/download

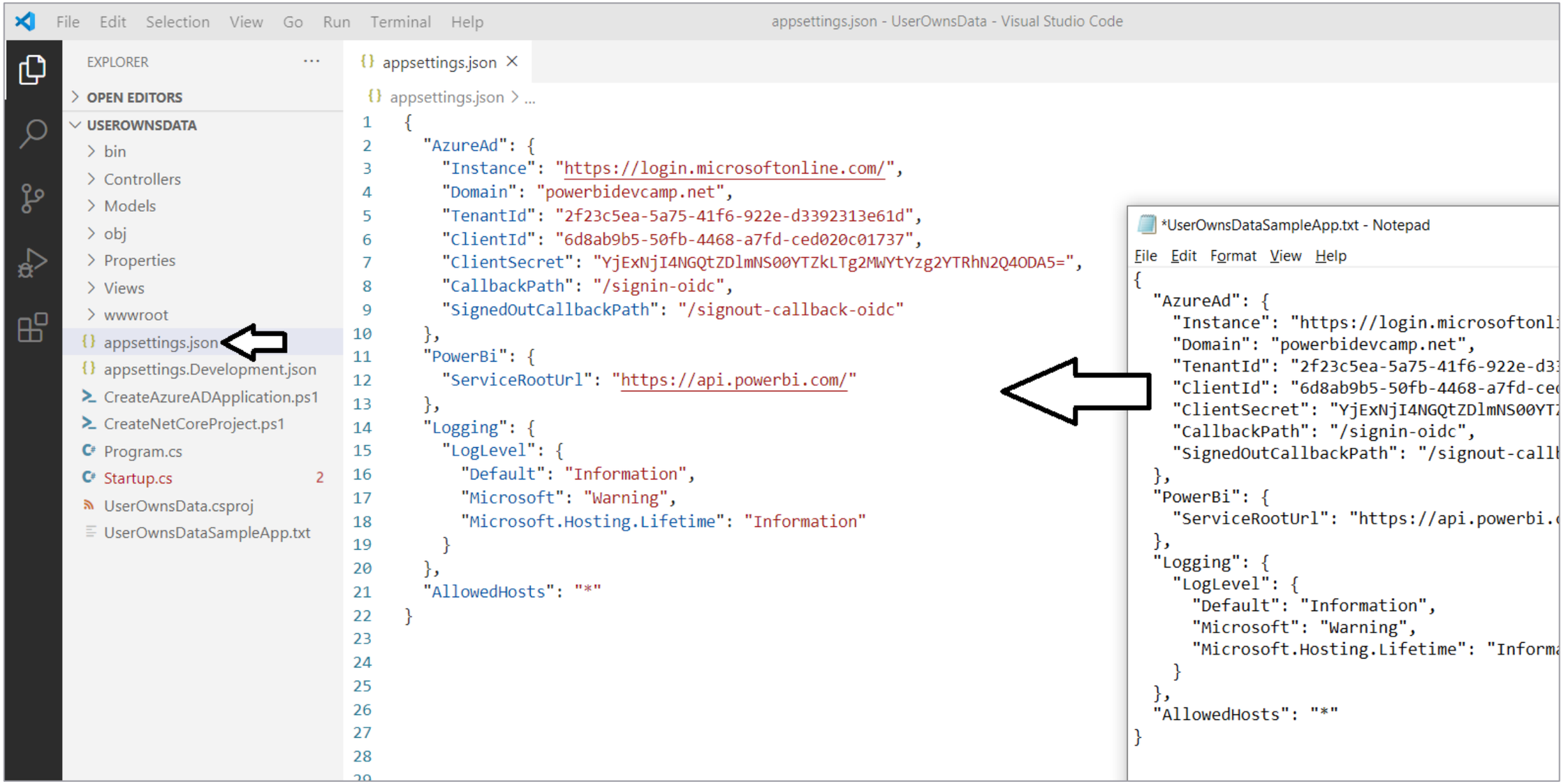


Getting Started with the .NET 5 CLI

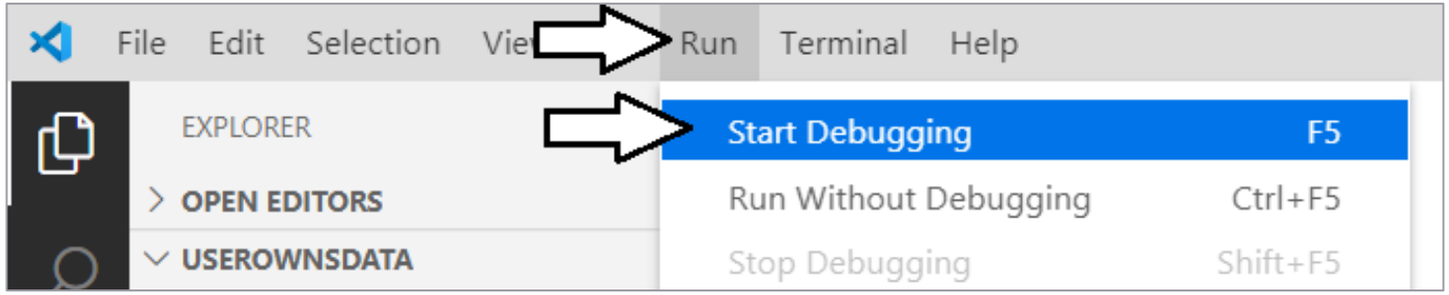
- Command to create a new project> `dotnet new mvc -auth singleOrg`

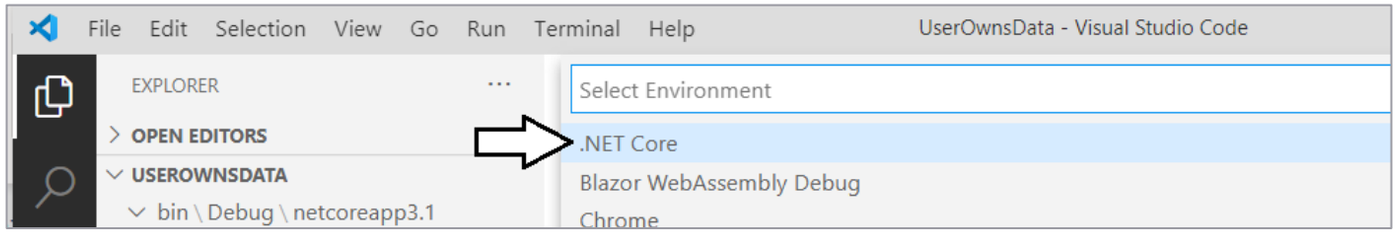


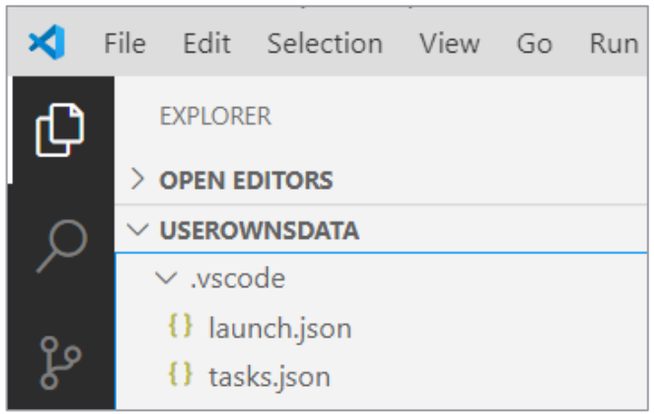
Azure AD Configuration in appsettings.json



Starting a .NET 5 Debugging Session

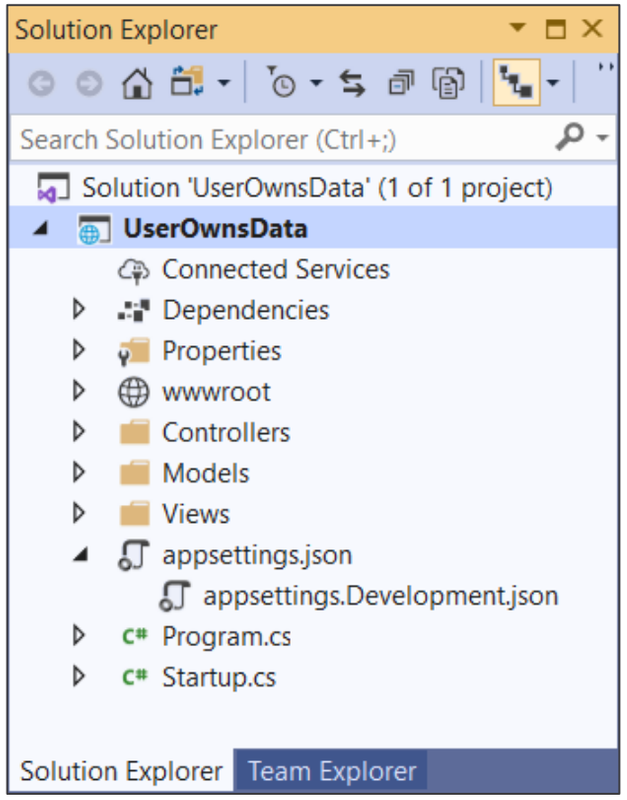


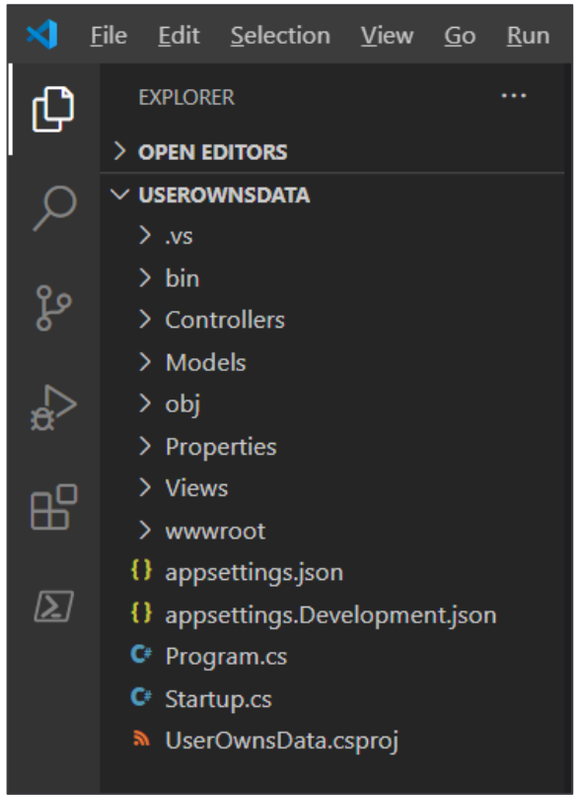




Visual Studio 2019 versus Visual Studio Code

- **Should you use Visual Studio 2019 versus Visual Studio Code?**
  - Yes, either one can be used to develop for .NET 5 3.1 and .NET 5





Agenda

- ✓ Developing for Power BI with .NET 5
- ➢ Introducing `Microsoft.Identity.Web`
- Calling to Power BI as Service Principal
- Programming the Power BI JavaScript API
- Adding TypeScript Support to a .NET 5 Project
- Programming with Multi-Resource Embed Tokens
- Integrating RLS using EffectiveIdentity

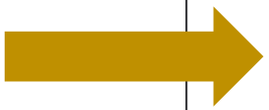
Introducing `Microsoft.Identity.Web`

- **What is `Microsoft.Identity.Web`**
  - Strategic authentication library for Azure AD to assist developers
  - Used to perform authentication in Web applications and Web APIs
  - Used to acquire access tokens
  - Used to implement token caching

- **Scenarios where you should use `Microsoft.Identity.Web`**
  - When developing Web applications and APIs built on .NET 3.1 and .NET 5
  - When authenticating users and acquiring user tokens
  - When authenticating service principals and acquiring app-only tokens

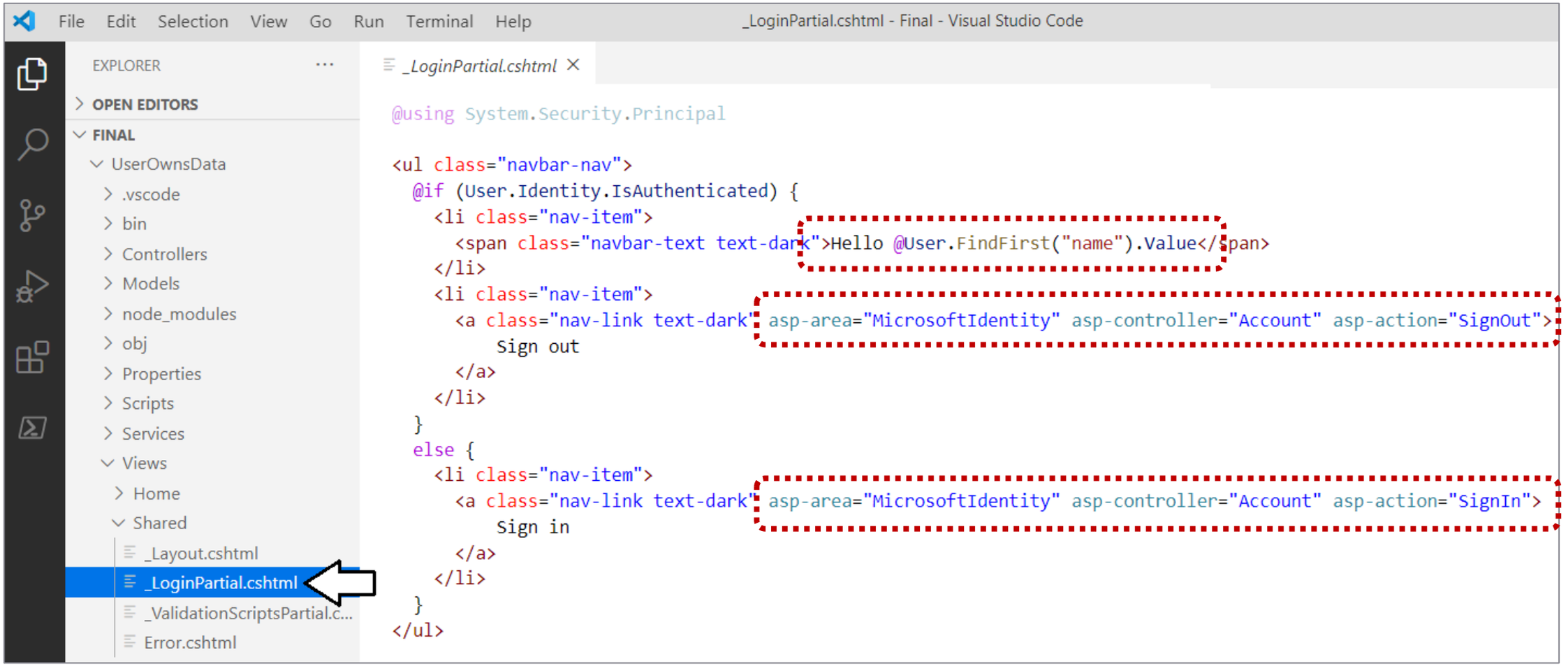
  - More info at https://github.com/AzureAD/microsoft-identity-web/wiki

Enabling Authentication with Microsoft.Identity.Web



```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services) {

  services.AddMicrosoftIdentityWebAppAuthentication(Configuration);

  var mvcBuilder = services.AddControllersWithViews(options => {
    var policy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build();
    options.Filters.Add(new AuthorizeFilter(policy));
  });

  mvcBuilder.AddMicrosoftIdentityUI();

  services.AddRazorPages();
}
```

Adding Sign in / Sign Out Links



ASP.NET Supports Route Authorization

- Routes with `[AllowAnonymous]` accessible to anonymous user
- Routes secured with `[Authorize]` only accessible to authenticated users
- Navigating to secured route automatically prompts for sign in

```
[Authorize]
public class HomeController : Controller {

    public HomeController() {}

    [AllowAnonymous]
    public IActionResult Index() {
        return View();
    }

    public IActionResult Embed() {
        return View();
    }

}
```

Index.cshtml

Embed.cshtml

Index.cshtml

```
@using System.Security.Principal

@if (User.Identity.IsAuthenticated) {
  <div class="jumbotron">
    <h2>Welcome @User.FindFirst("name").Value</h2>
    <p>You have now logged into this application.</p>
  </div>
}
else {
  <div class="jumbotron">
    <h2>Welcome to the User-Owns-Data Tutorial</h2>
    <p>Click the <strong>sign in</strong> link in the upper
  </div>
}
```

Embed.cshtml

```
<h2>TODO: Embed Report Here</h2>
```

Agenda

- ✓ Developing for Power BI with .NET 5
- ✓ Introducing `Microsoft.Identity.Web`
- ➢ Calling to Power BI as Service Principal
- ▪ Programming the Power BI JavaScript API
- ▪ Adding TypeScript Support to a .NET 5 Project
- ▪ Programming with Multi-Resource Embed Tokens
- ▪ Integrating RLS using EffectiveIdentity

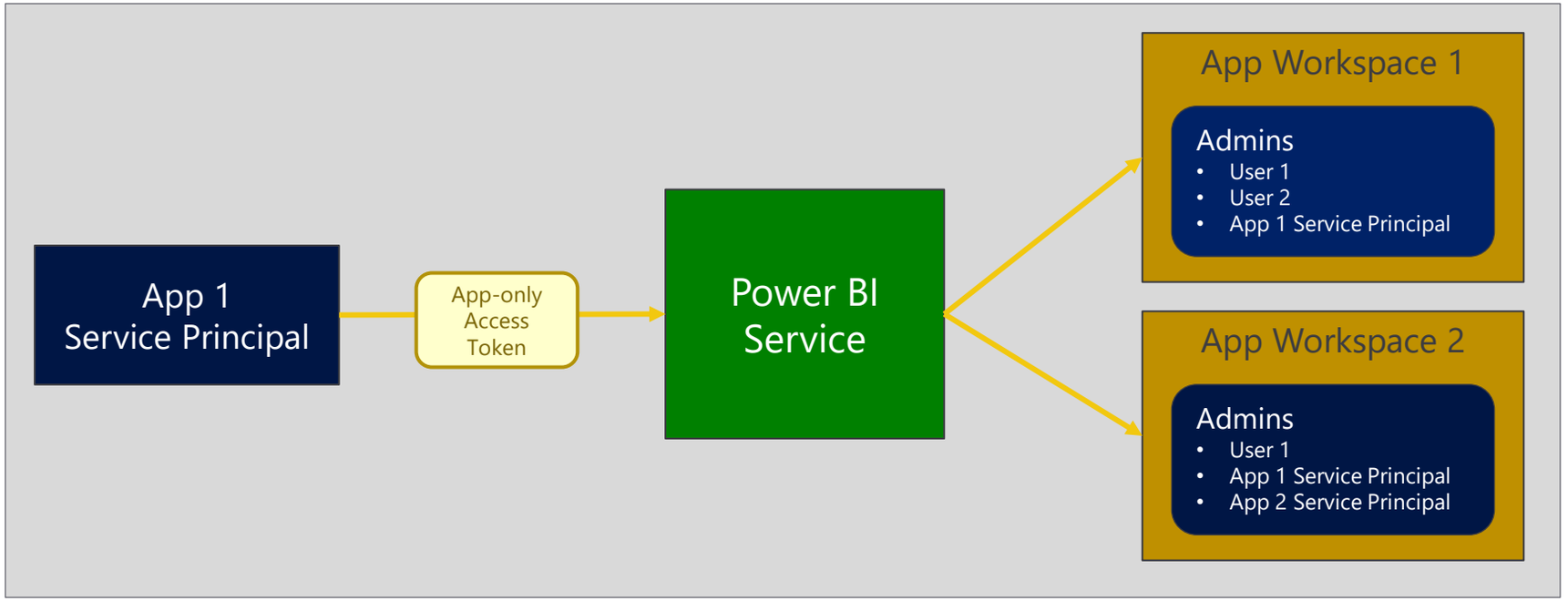
What Is the Power BI Service API?

- **What is the Power BI Service API?**
  - API built on OAuth2, OpenID Connect, REST and ODATA
  - API secured by Azure Active Directory (AAD)
  - API to program with workspaces, datasets, reports & dashboards
  - API also often called “Power BI REST API”

- **What can you do with the Power BI Service API?**
  - Publish PBIX project files
  - Update connection details and datasource credentials
  - Create workspaces and clone content across workspaces
  - Embed Power BI reports and dashboards tiles in web pages
  - Create streaming datasets in order to build real-time dashboards

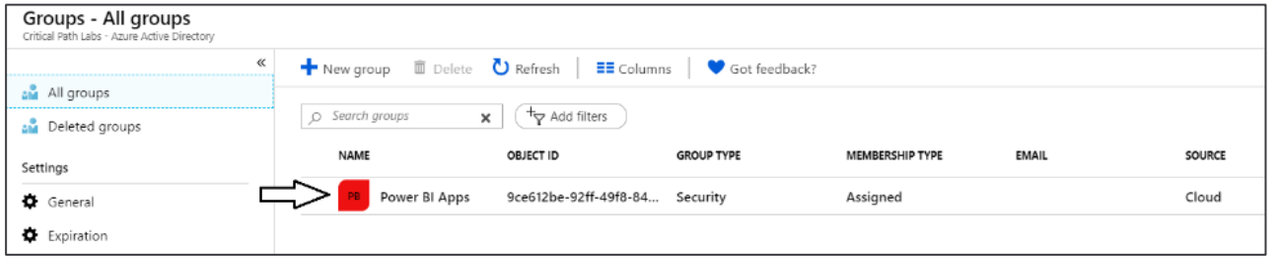
App-only Access Control

- **Service Principal used to configure access control**
  - Requires the use of v2 app workspaces
  - Service principal added to app workspaces as admin
  - Access control NOT based on Azure AD permissions

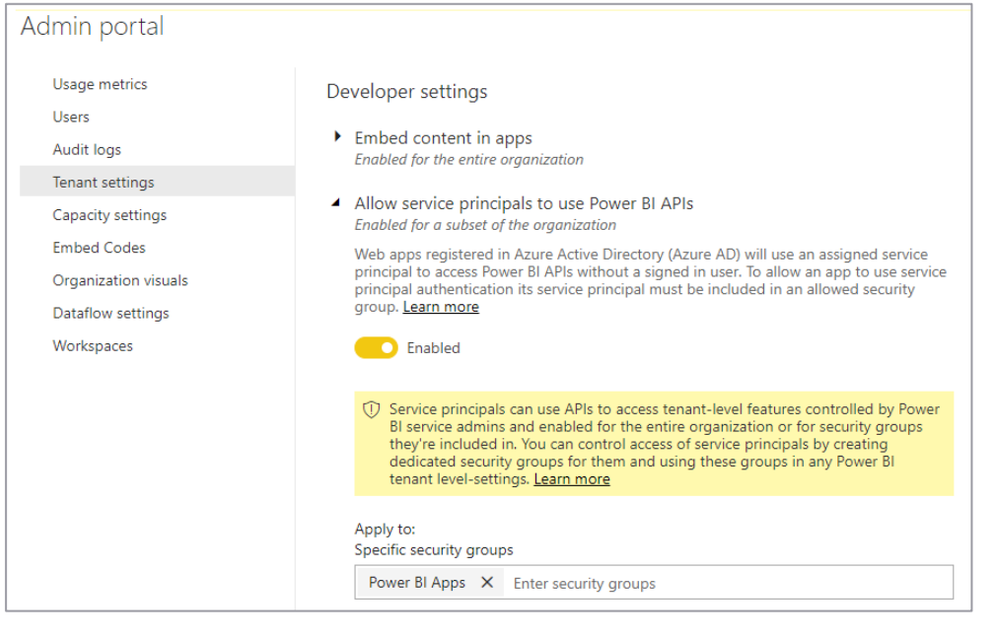


Setting Up for App-Owns-Data – Part 1

- **Enable Service Principal Access to Power BI Service API**
  - Create an Azure AD security group (e.g. Power BI Apps)

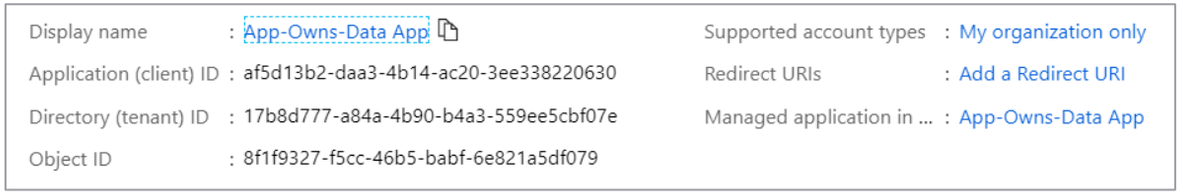


  - Add group to *Power BI Allow service principals to use Power BI APIs*

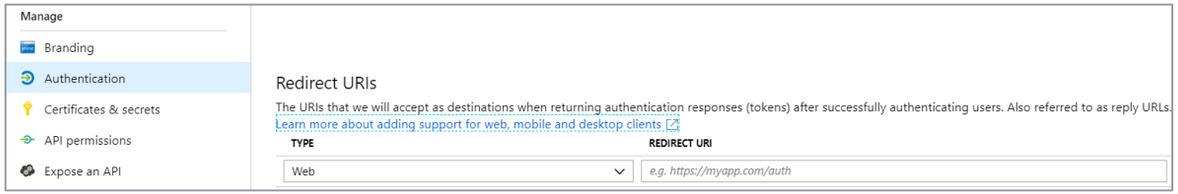


Setting Up for App-Owns-Data – Part 2

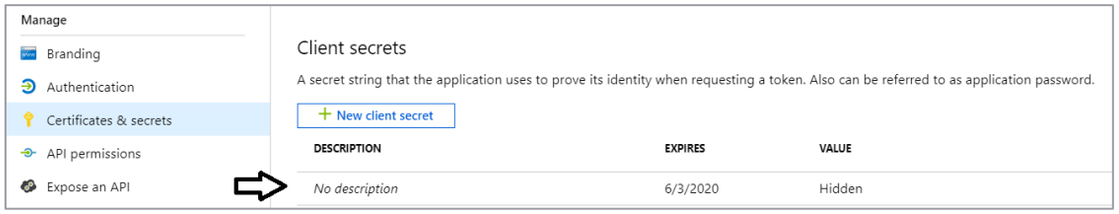
- Create a confidential client in your Azure AD tenant



- Configured as TYPE=Web and no need for a redirect URL



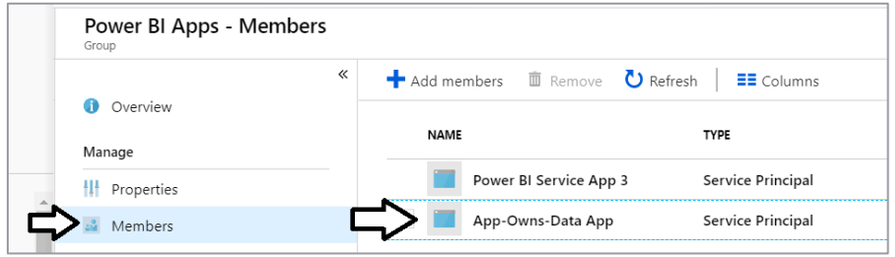
- Add a client secret or a client certificate



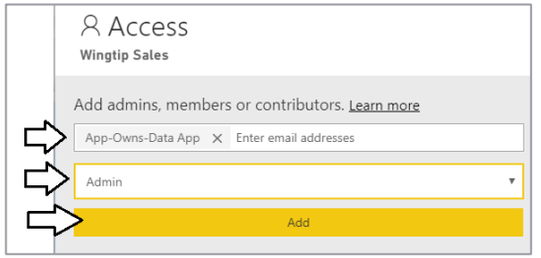
No need to configure any permissions

Setting Up for App-Owns-Data – Part 3

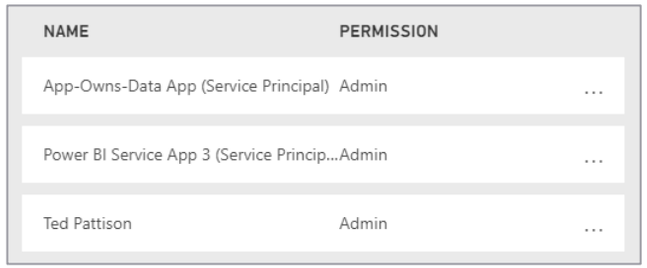
- Add application's service principal in Power BI Apps security group



- Configure application's service principal as workspace admin



- Service principal should now be workspace admin

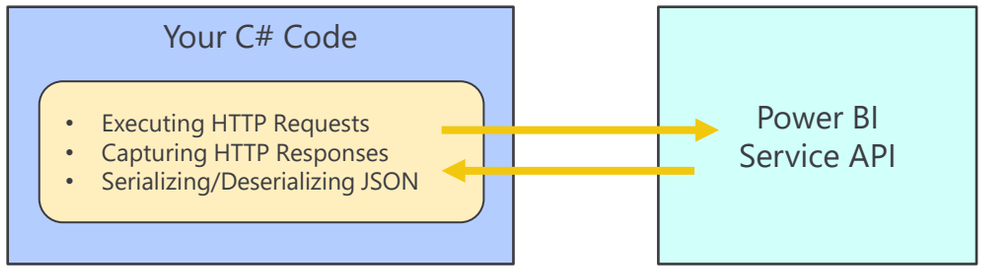


Calling Directly to the Power BI Service API

This is a sample of C# code that does not use the Power BI .NET SDK

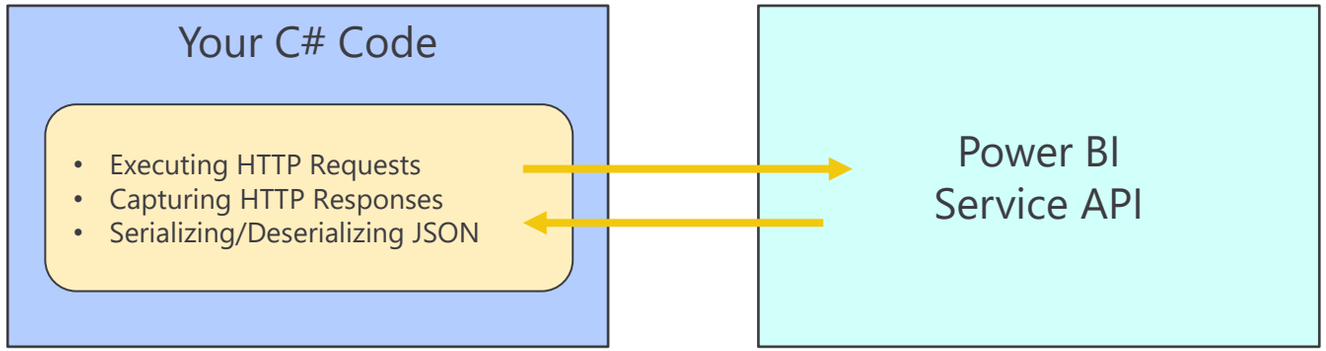
```
static string ExecuteGetRequest(string restUrl) {
  HttpClient client = new HttpClient();
  HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, restUrl);
  request.Headers.Add("Authorization", "Bearer " + GetAccessToken());
  request.Headers.Add("Accept", "application/json;odata.metadata=minimal");
  HttpResponseMessage response = client.SendAsync(request).Result;
  if (response.StatusCode != HttpStatusCode.OK) {
    throw new ApplicationException("Error occured calling the Power BI Servide API");
  }
  return response.Content.ReadAsStringAsync().Result;
}

static void Main() {
  // get report data from app workspace
  string restUrl = "https://api.powerbi.com/v1.0/myorg/groups/" + appWorkspaceId + "/reports/";
  var json = ExecuteGetRequest(restUrl);
  ReportCollection reports = JsonConvert.DeserializeObject<ReportCollection>(json);
  foreach (Report report in reports.value) {
    Console.WriteLine("Report Name: " + report.name);
    Console.WriteLine();
  }
}
```

```
public class Report {
  public string id { get; set; }
  public string name { get; set; }
  public string webUrl { get; set; }
  public string embedUrl { get; set; }
  public bool isOwnedByMe { get; set; }
  public string datasetId { get; set; }
}

public class ReportCollection {
  public List<Report> value { get; set; }
}
```



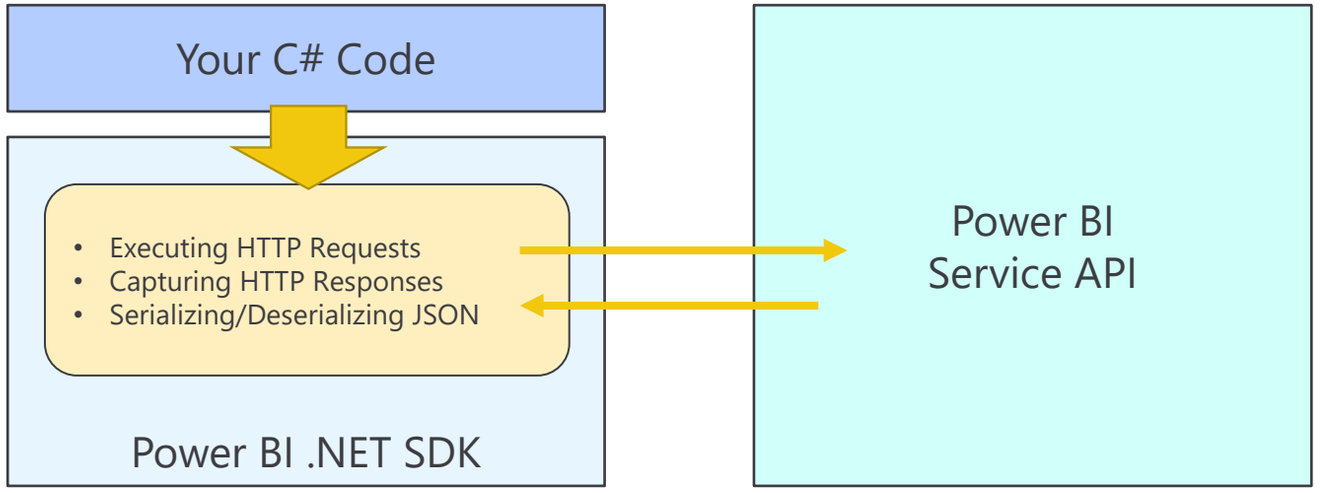
Power BI .NET SDK

- Developing without the Power BI .NET SDK




```
static string ExecuteGetRequest(string restUrl) {
  HttpClient client = new HttpClient();
  HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, restUrl);
  request.Headers.Add("Authorization", "Bearer " + GetAccessToken());
  request.Headers.Add("Accept", "application/json;odata.metadata=minimal");
  HttpResponseMessage response = client.SendAsync(request).Result;
  if (response.StatusCode != HttpStatusCode.OK) {
    throw new ApplicationException("Error occured calling the Power BI Servide API");
  }
  return response.Content.ReadAsStringAsync().Result;
}

static void Main() {
  // get report data from app workspace
  string restUrl = "https://api.powerbi.com/v1.0/myorg/groups/" + appWorkspaceId + "/reports/";
  var json = ExecuteGetRequest(restUrl);
  ReportCollection reports = JsonConvert.DeserializeObject<ReportCollection>(json);
  foreach (Report report in reports.value) {
    Console.WriteLine("Report Name: " + report.name);
    Console.WriteLine();
  }
}
```

- Developing with the Power BI .NET SDK




```
PowerBIClient pbiClient = GetPowerBiClient();

// call to Power BI Service API to get embedding data
var report = await pbiClient.Reports.GetReportInGroupAsync(WorkspaceId, ReportId);
```

Exercise 3

Call the Power BI Service API

1. Add a new class named `PowerBiServiceApi`
2. Add support to acquire access tokens using `Microsoft.Identity.Web`
3. Call Power BI Service API to get embedding data for a report
4. Pass the embedding data for a report to the browser

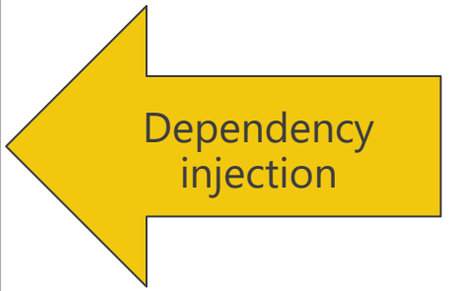
Adding Support for Token Acquisition

- Modify ConfigureService in Startup.cs

```
public void ConfigureServices (IServiceCollection services) {

  services
    .AddMicrosoftIdentityWebAppAuthentication(Configuration)
    .EnableTokenAcquisitionToCallDownstreamApi(PowerBiServiceApi.RequiredScopes)
    .AddInMemoryTokenCaches();

  services.AddScoped (typeof (PowerBiServiceApi));
```

- Add a new C# Class named PowerBiServiceAPi

```
public class PowerBiServiceApi {

    private ITokenAcquisition tokenAcquisition { get; }
    private string urlPowerBiServiceApiRoot { get; }

    public PowerBiServiceApi(IConfiguration configuration, ITokenAcquisition tokenAcquisition)
        this.urlPowerBiServiceApiRoot = configuration["PowerBi:ServiceRootUrl"];
        this.tokenAcquisition = tokenAcquisition;
    }
```



Authenticating with Azure AD as Service Principal

```
public class PowerBiServiceApi {

  private ITokenAcquisition tokenAcquisition { get; }
  private string urlPowerBiServiceApiRoot { get; }

  public PowerBiServiceApi(IConfiguration configuration, ITokenAcquisition tokenAcquisition) {
    this.urlPowerBiServiceApiRoot = configuration["PowerBi:ServiceRootUrl"];
    this.tokenAcquisition = tokenAcquisition;
  }

  public const string powerbiApiDefaultScope = "https://analysis.windows.net/powerbi/api/.default";

  public string GetAccessToken() {
    return this.tokenAcquisition.GetAccessTokenForAppAsync(powerbiApiDefaultScope).Result;
  }

  public PowerBIClient GetPowerBiClient() {
    var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");
    return new PowerBIClient(new Uri(urlPowerBiServiceApiRoot), tokenCredentials);
  }
```

Accessing PowerBiServiceApi from a Controller

```
[Authorize]
public class HomeController : Controller {

    private PowerBiServiceApi powerBiServiceApi;

    public HomeController(PowerBiServiceApi powerBiServiceApi) {
        this.powerBiServiceApi = powerBiServiceApi;
    }

    public async Task<IActionResult> Embed() {
        Guid workspaceId = new Guid("912f2b34-7daa-4589-83df-35c75944d864");
        Guid reportId = new Guid("cd496c1c-8df0-48e7-8b92-e2932298743e");

        // call to PowerBiServiceApi
        var viewModel = await powerBiServiceApi.GetReport(workspaceId, reportId);
        return View(viewModel);
    }
```

Dependency injection

Call GetReportInGroupAsync to Get Embedding Data

```
public class EmbeddedReportViewModel {
    public string Id;
    public string Name;
    public string EmbedUrl;
    public string Token;
}
```

```
public async Task<EmbeddedReportViewModel> GetReport(Guid WorkspaceId, Guid ReportId) {

    PowerBIClient pbiClient = GetPowerBiClient();

    // call to Power BI Service API to get embedding data
    var report = await pbiClient.Reports.GetReportInGroupAsync(WorkspaceId, ReportId);

    // return report embedding data to caller
    return new EmbeddedReportViewModel {
        Id = report.Id.ToString(),
        EmbedUrl = report.EmbedUrl,
        Name = report.Name,
        Token = GetAccessToken()
    };
}
```

Generating Embed Tokens

- **App-Owns-Data embedding requires developer to generate embed tokens**
  - Developer controls what permissions are extended to user for embedding purposes
  - Generating embed tokens requires dedicated capacity for any production scenarios

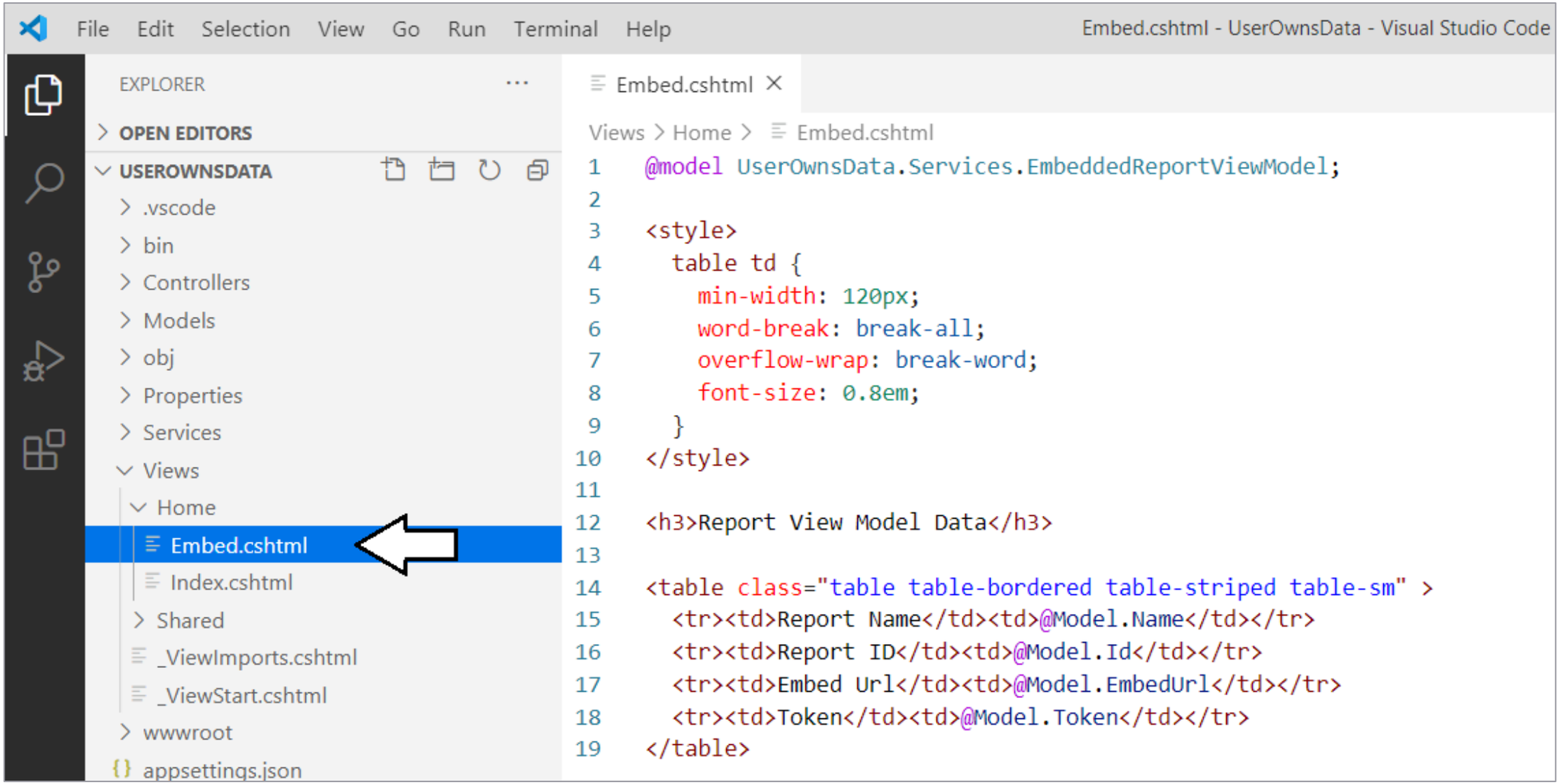
- **You generate embed tokens with the Power BI Service API**
  - Embed Token V1 API used to create single resource embed tokens (see example below)
  - Embed Token V2 API used to create multi-resource embed tokens (see example in later slide)
  - Embed token provides restrictions on whether user can view, edit or create
  - Embed token can be generated to support row-level security (RLS)

```
Report report = reports.Where(r => r.Id == reportId).First();
var generateTokenRequestParameters = new GenerateTokenRequest(accessLevel: "edit");
var token = client.Reports.GenerateTokenInGroupAsync(appWorkspaceId,
                                                     report.Id,
                                                     generateTokenRequestParameters).Result;
```

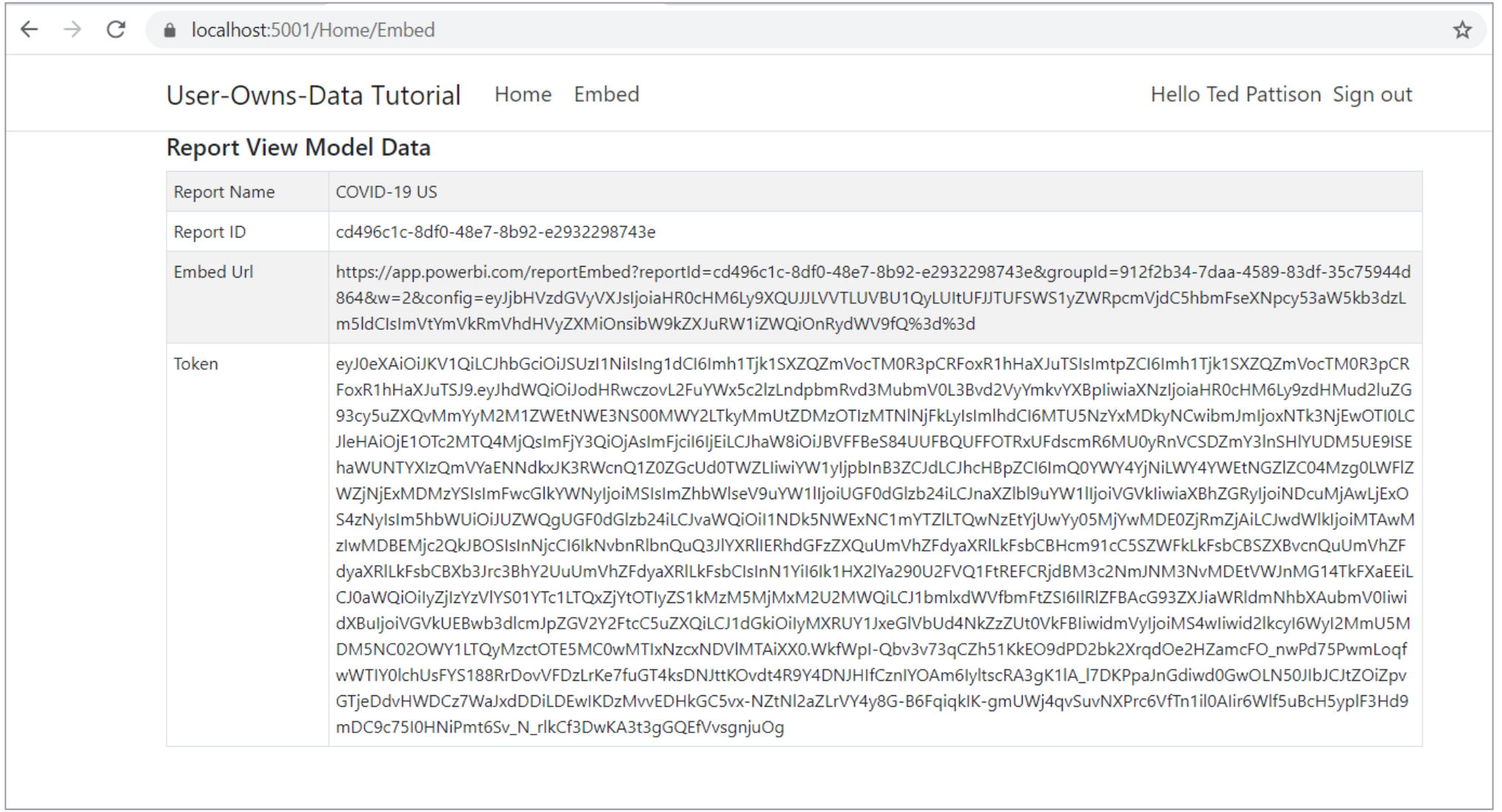
Getting the Data for Report Embedding

```
public async Task<EmbeddedReportViewModel> GetReport(Guid WorkspaceId, Guid ReportId) {

    PowerBIClient pbiClient = GetPowerBiClient();

    // call to Power BI Service API to get embedding data
    var report = await pbiClient.Reports.GetReportInGroupAsync(WorkspaceId, ReportId);

    // generate read-only embed token for the report
    var datasetId = report.DatasetId;
    var tokenRequest = new GenerateTokenRequest(TokenAccessLevel.View, datasetId);
    var embedTokenResponse = await pbiClient.Reports.GenerateTokenAsync(WorkspaceId, ReportId, tokenRequest);
    var embedToken = embedTokenResponse.Token;

    // return report embedding data to caller
    return new EmbeddedReportViewModel {
        Id = report.Id.ToString(),
        EmbedUrl = report.EmbedUrl,
        Name = report.Name,
        Token = embedToken
    };
}
```

Displaying Data from the View Model



Exercise 3 Solution



Agenda

- ✓ Developing for Power BI with .NET 5
- ✓ Introducing `Microsoft.Identity.Web`
- ✓ Calling to Power BI as Service Principal
- ➢ **Programming the Power BI JavaScript API**
  - ▪ Adding TypeScript Support to a .NET 5 Project
  - ▪ Programming with Multi-Resource Embed Tokens
  - ▪ Integrating RLS using EffectiveIdentity

Exercise 4

Embedding a Report using powerbi.js

1. Add a script link to powerbi.js
2. Create view model to pass embedding data to browser
3. Write JavaScript code to embed a report

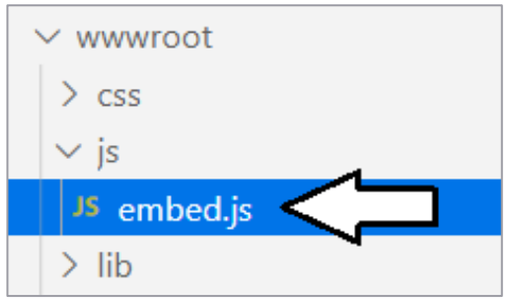
Creating a Simple View To Embed a Report

1. First script link loads Power BI JavaScript API
2. Second script link adds view model to web page with report embedding data
3. Third script link loads a custom Javascript file named embed.js that you will write

Embed.cshtml

```
@model UserOwnsData.Services.EmbeddedReportViewModel;

<div id="embed-container" style="height:800px;"></div>

@section Scripts {
  <script src="https://cdn.jsdelivr.net/npm/powerbi-client@2.13.3/dist/powerbi.min.js"></script>
  <script>
    var viewModel = {
      reportId: "@Model.Id",
      embedUrl: "@Model.EmbedUrl",
      token: "@Model.Token"
    };
  </script>
  <script src="~/js/embed.js"></script>
}
```

Retrieving Data from the Report View Model

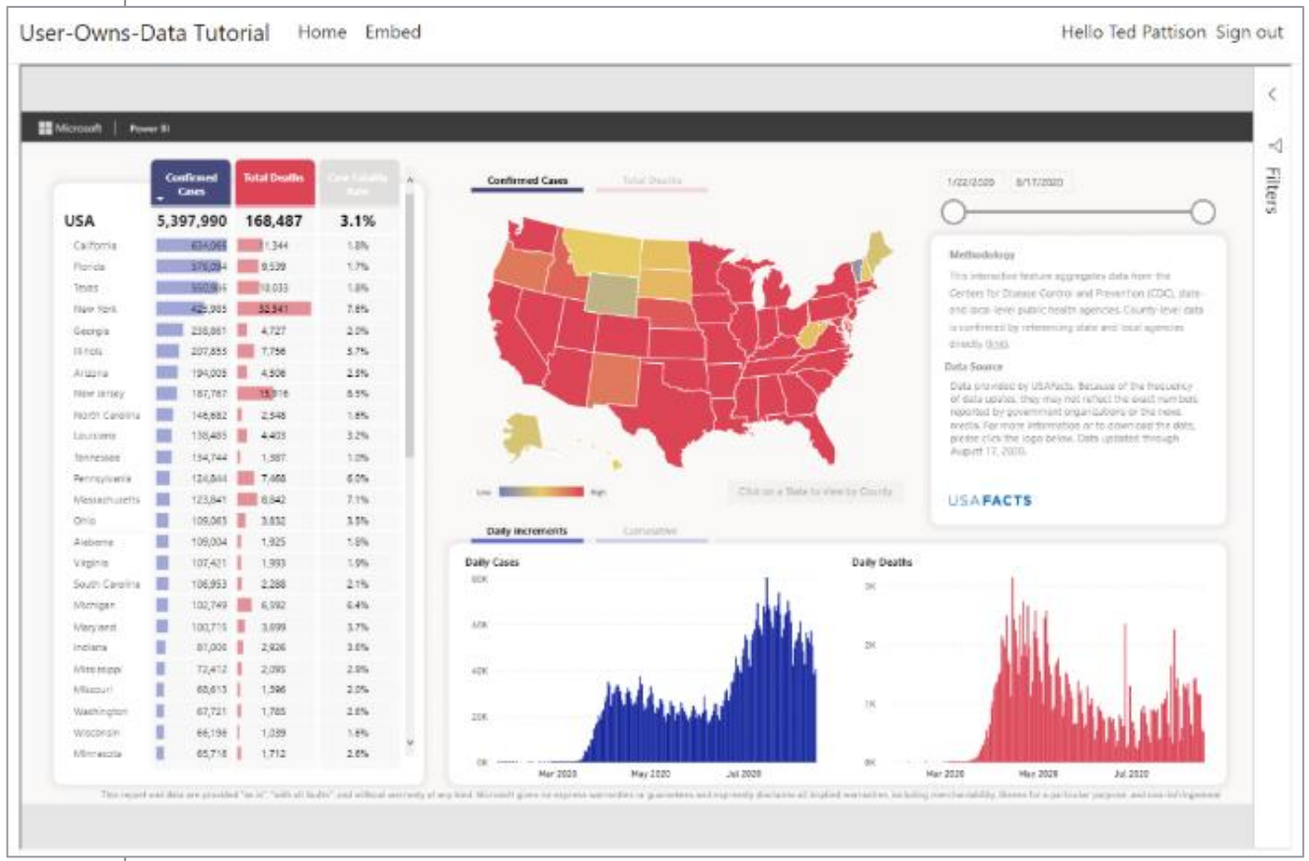


JS embed.js

```
$(function () {

  // 1 - get DOM object for div that is report container
  var reportContainer = document.getElementById("embed-container");

  // 2 - get report embedding data from view model
  var reportId = window.viewModel.reportId;
  var embedUrl = window.viewModel.embedUrl;
  var token = window.viewModel.token

  // 3 - embed report using the Power BI JavaScript API.

  // 4 - add logic to resize embed container on window resize event

});
```

Embedding a Report using powerbi.js

```
// 2 - get report embedding data from view model
var reportId = window.viewModel.reportId;
var embedUrl = window.viewModel.embedUrl;
var token = window.viewModel.token

// 3 - embed report using the Power BI JavaScript API.
var models = window['powerbi-client'].models;

var config = {
  type: 'report',
  id: reportId,
  embedUrl: embedUrl,
  accessToken: token,
  permissions: models.Permissions.All,
  tokenType: models.TokenType.Aad,
  viewMode: models.ViewMode.View,
  settings: {
    panes: {
      filters: { expanded: false, visible: true },
      pageNavigation: { visible: false }
    }
  }
};

// Embed the report and display it within the div container.
var report = powerbi.embed(reportContainer, config);
```



Agenda

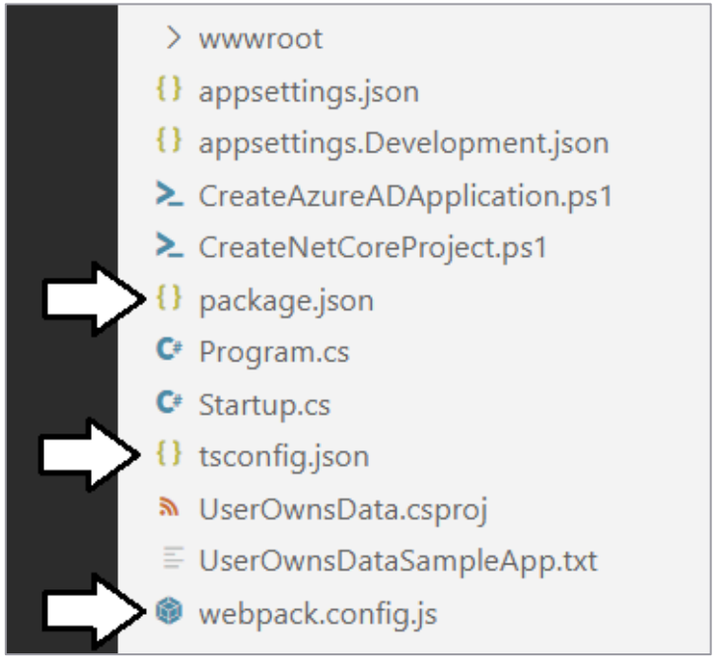
- ✓ Developing for Power BI with .NET 5
- ✓ Introducing `Microsoft.Identity.Web`
- ✓ Calling to Power BI as Service Principal
- ✓ Programming the Power BI JavaScript API
- ➢ Adding TypeScript Support to a .NET 5 Project
- ▪ Programming with Multi-Resource Embed Tokens
- ▪ Integrating RLS using EffectiveIdentity

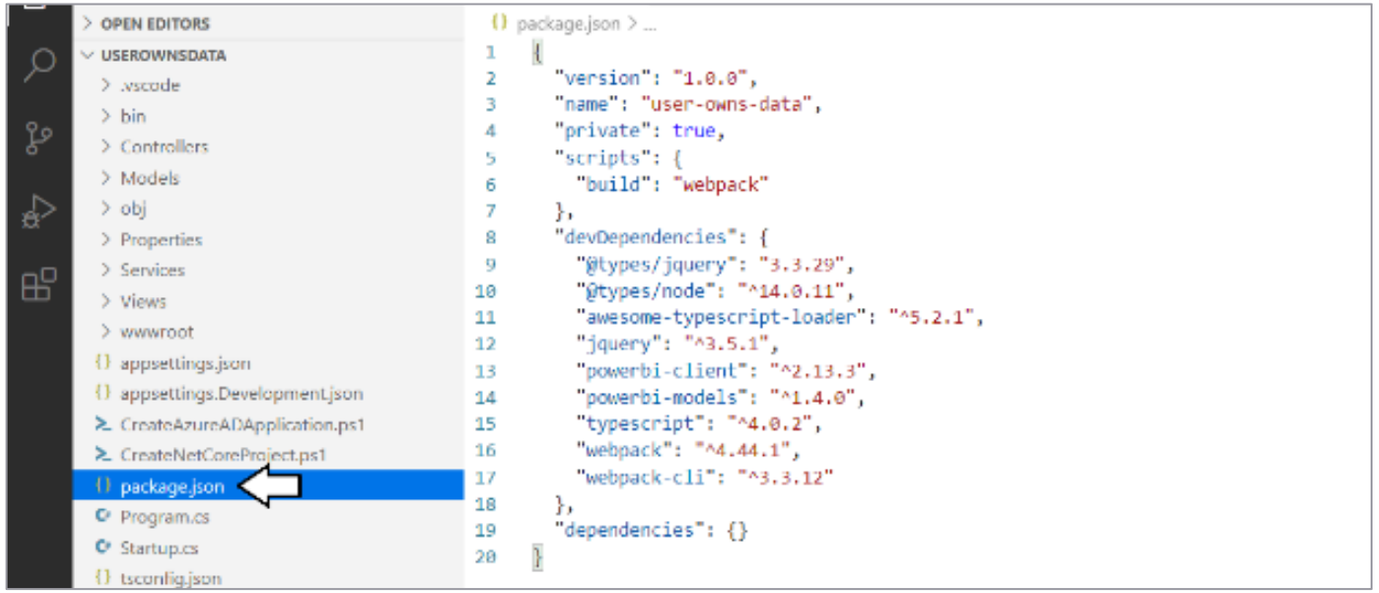
Exercise 5

Adding TypeScript/Webpack Support to a .NET 5 Project

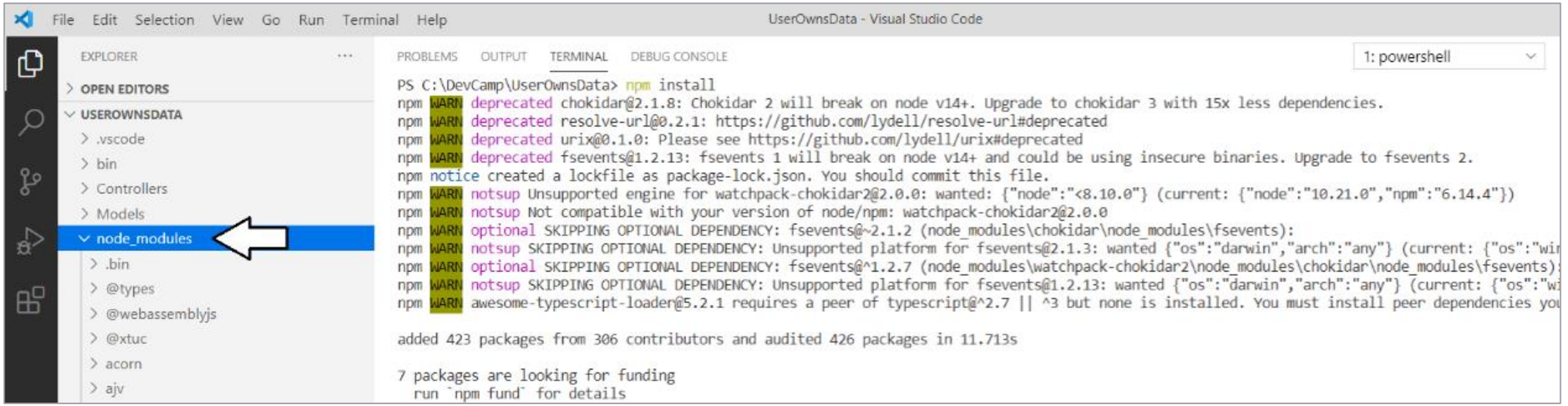
1. Copy `package.json` file to `UserOwnsData` project root folder
2. Execute `npm install` to install Node.js packages
3. Copy `tsconfig.json` and `webpack.config.js` to root folder
4. Add new TypeScript file named `embed.ts`
5. Execute `npm run build` to compile `embed.ts` to `embed.js`
6. Update `UserOwnsData.csproj` with `npm run build` command

Adding Node.js Support for TypeScript Compilation





Running `npm install`

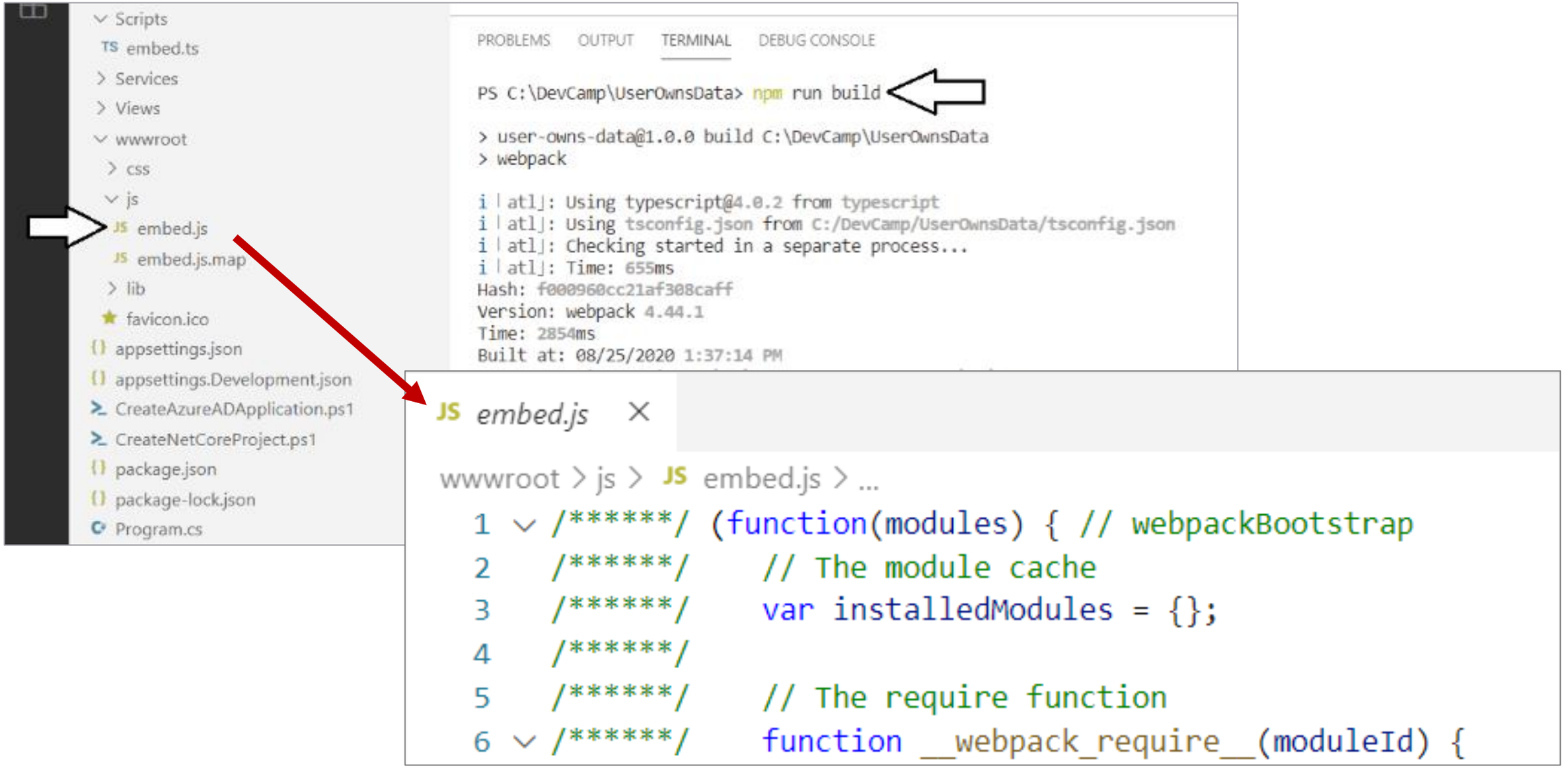


Understanding Webpack and Dynamic Module Loading

TS embed.ts

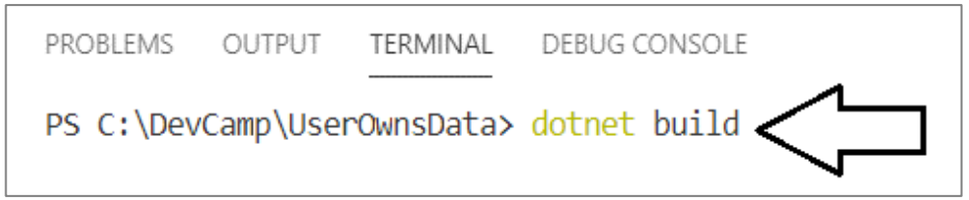
```
import * as $ from 'jquery';

import * as powerbi from "powerbi-client";
import * as models from "powerbi-models";

// ensure Power BI JavaScript API has loaded
require('powerbi-models');
require('powerbi-client');
```

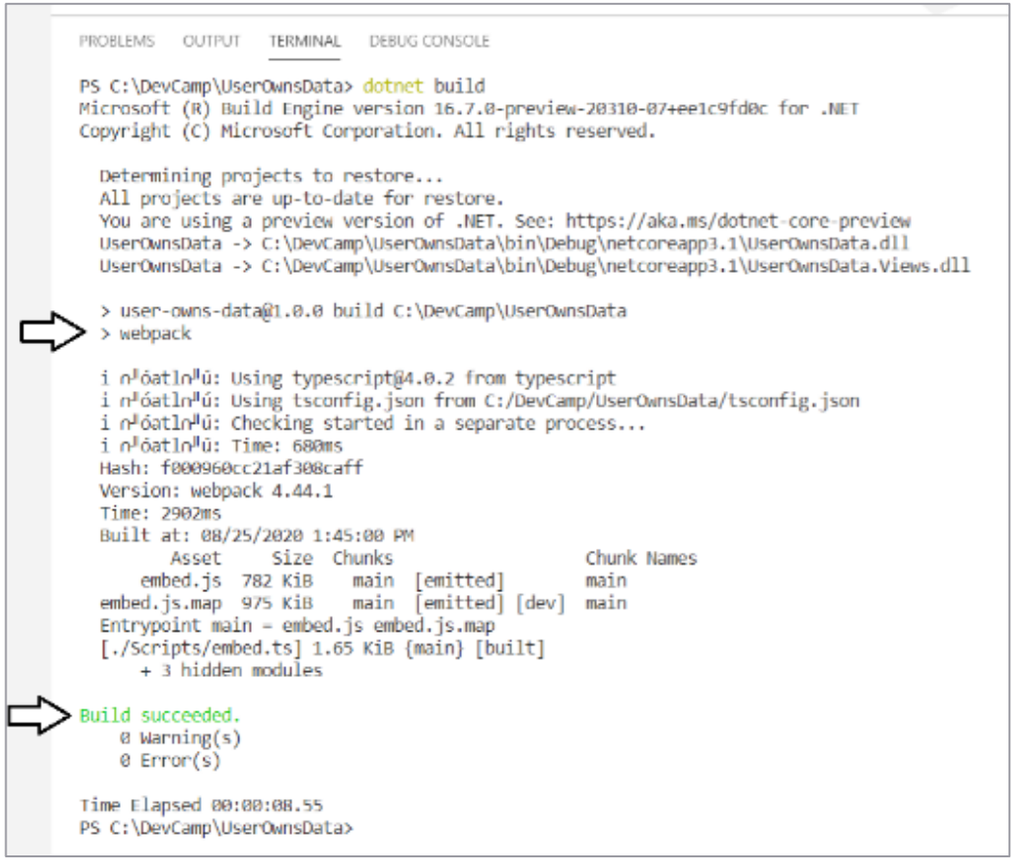
Running npm build



Updating UserOwnsData.csproj

Running dotnet build





Programming the PBI JS API using TypeScript

```typescript
// get DOM object div for report container
var reportContainer: HTMLElement = document.getElementById("embed-container");

var viewModel: viewModel = window["viewModel"];

var config: powerbi.IEmbedConfiguration = {
    type: "report",
    id: viewModel.reportId,
    embedUrl: viewModel.embedUrl,
    accessToken: viewModel.token,
    permissions: models.Permissions.All,
    tokenType: models.TokenType.Aad,
    viewMode: models.ViewMode.View,
    settings: {
        panes: {
            filters: { expanded: false, visible: true },
            pageNavigation: { visible: true }
        },
        persistentFiltersEnabled: true
    }
};

// Embed the report and display it within the div container.
var report: powerbi.Report = <powerbi.Report>window.powerbi.embed(reportContainer, config);
```

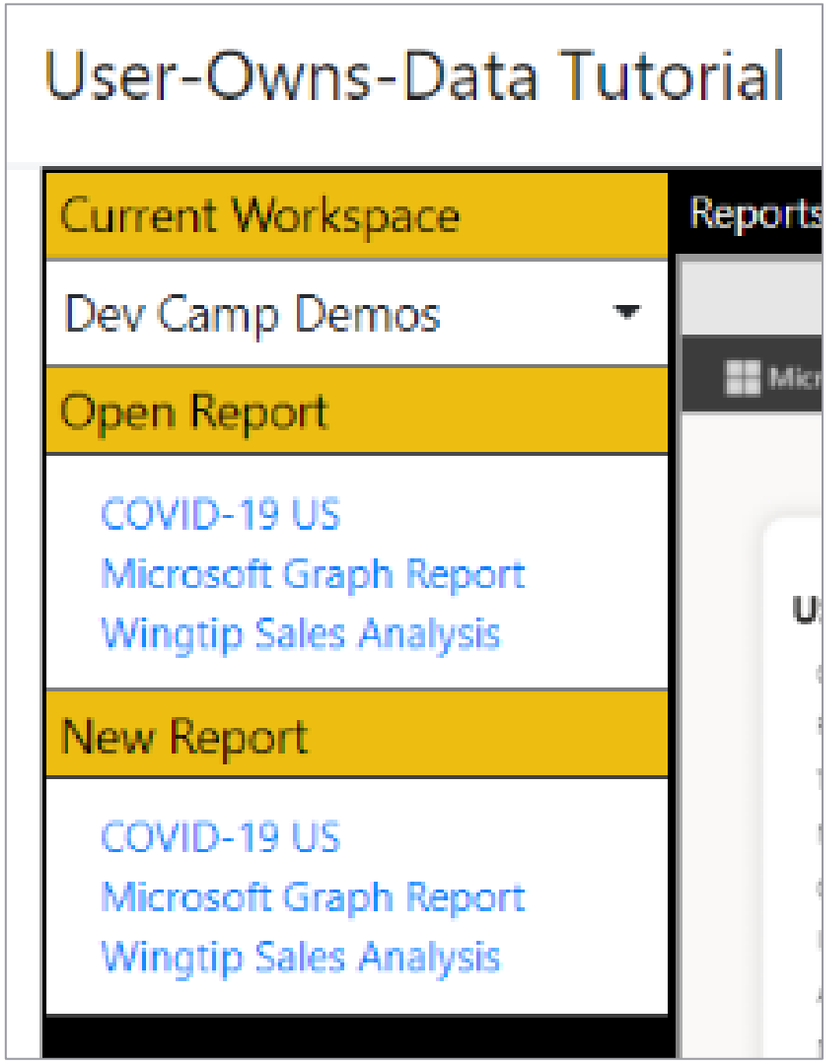
Exercise 6
Creating a View Model for App Workspaces

```
public async Task<string> GetEmbeddedViewModel(string appWorkspaceId = "") {

  var accessToken = this.tokenAcquisition.GetAccessTokenForUserAsync(RequiredScopes).Result;
  var tokenCredentials = new TokenCredentials(accessToken, "Bearer");
  PowerBIClient pbiClient = new PowerBIClient(new Uri(urlPowerBiServiceApiRoot), tokenCredentials);

  Object viewModel;
  if (string.IsNullOrEmpty(appWorkspaceId)) {
    viewModel = new {
      currentWorkspace = "My Workspace",
      workspaces = (await pbiClient.Groups.GetGroupsAsync()).Value,
      datasets = (await pbiClient.Datasets.GetDatasetsAsync()).Value,
      reports = (await pbiClient.Reports.GetReportsAsync()).Value,
      token = accessToken
    };
  }
  else {
    Guid workspaceId = new Guid(appWorkspaceId);
    var workspaces = (await pbiClient.Groups.GetGroupsAsync()).Value;
    var currentWorkspace = workspaces.First((workspace) => workspace.Id == workspaceId);
    viewModel = new {
      workspaces = workspaces,
      currentWorkspace = currentWorkspace.Name,
      currentWorkspaceIsReadOnly = currentWorkspace.IsReadOnly,
      datasets = (await pbiClient.Datasets.GetDatasetsInGroupAsync(workspaceId)).Value,
      reports = (await pbiClient.Reports.GetReportsInGroupAsync(workspaceId)).Value,
      token = accessToken
    };
  }
  return JsonConvert.SerializeObject(viewModel);
}
```


```
{
  "workspaces": [
    { "id": "6679bd47-5b5f-4be0-ac6d-7a7ab1ba16f8", "name": "All Company", "isReadOn
    { "id": "912f2b34-7daa-4589-83df-35c75944d864", "name": "Dev Camp Demos", "isRea
  ],
  "currentWorkspace": "Dev Camp Demos",
  "currentWorkspaceIsReadOnly": false,
  "datasets": [
    { "id": "94e863ea-9117-445c-ac9e-db9373bdb8ea", "name": "COVID-19 US" },
    { "id": "89795189-0f43-4057-9af7-ab4838082a3b", "name": "Microsoft Graph Report"
    { "id": "bfa46105-5d6e-4270-88d8-eb6be5bf57ad", "name": "Wingtip Sales Analysis"
  ],
  "reports": [
    { "id": "cd496c1c-8df0-48e7-8b92-e2932298743e", "name": "COVID-19 US", "webUrl":
    { "id": "23d56559-af3c-4c51-9175-2904ee76bbae", "name": "Microsoft Graph Report"
    { "id": "fa89b3ae-c6a6-4e0c-8e4b-be7057cf583c", "name": "Wingtip Sales Analysis"
  ],
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImppYk5iaOZTU2JteFBZck45Q0Zx
};
```

Creating the User Experience


```
{
  "workspaces": [
    { "id": "6679bd47-5b5f-4be0-ac6d-7a7ab1ba16f8", "name": "All Company", "isReadOn
    { "id": "912f2b34-7daa-4589-83df-35c75944d864", "name": "Dev Camp Demos", "isRea
  ],
  "currentWorkspace": "Dev Camp Demos",
  "currentWorkspaceIsReadOnly": false,
  "datasets": [
    { "id": "94e863ea-9117-445c-ac9e-db9373bdb8ea", "name": "COVID-19 US" },
    { "id": "89795189-0f43-4057-9af7-ab4838082a3b", "name": "Microsoft Graph Report"
    { "id": "bfa46105-5d6e-4270-88d8-eb6be5bf57ad", "name": "Wingtip Sales Analysis"
  ],
  "reports": [
    { "id": "cd496c1c-8df0-48e7-8b92-e2932298743e", "name": "COVID-19 US", "webUrl":
    { "id": "23d56559-af3c-4c51-9175-2904ee76bbae", "name": "Microsoft Graph Report"
    { "id": "fa89b3ae-c6a6-4e0c-8e4b-be7057cf583c", "name": "Wingtip Sales Analysis"
  ],
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImppYk5iaOZTU2JteFBZck45Q0Zx
};
```

Agenda

- ✓ Developing for Power BI with .NET 5
- ✓ Introducing `Microsoft.Identity.Web`
- ✓ Calling to Power BI as Service Principal
- ✓ Programming the Power BI JavaScript API
- ✓ Adding TypeScript Support to a .NET 5 Project
- ➢ **Programming with Multi-Resource Embed Tokens**
- ▪ **Integrating RLS using EffectiveIdentity**

Programming with Multi-Resource Embed Tokens

- Used in App-Owns-Data embedding

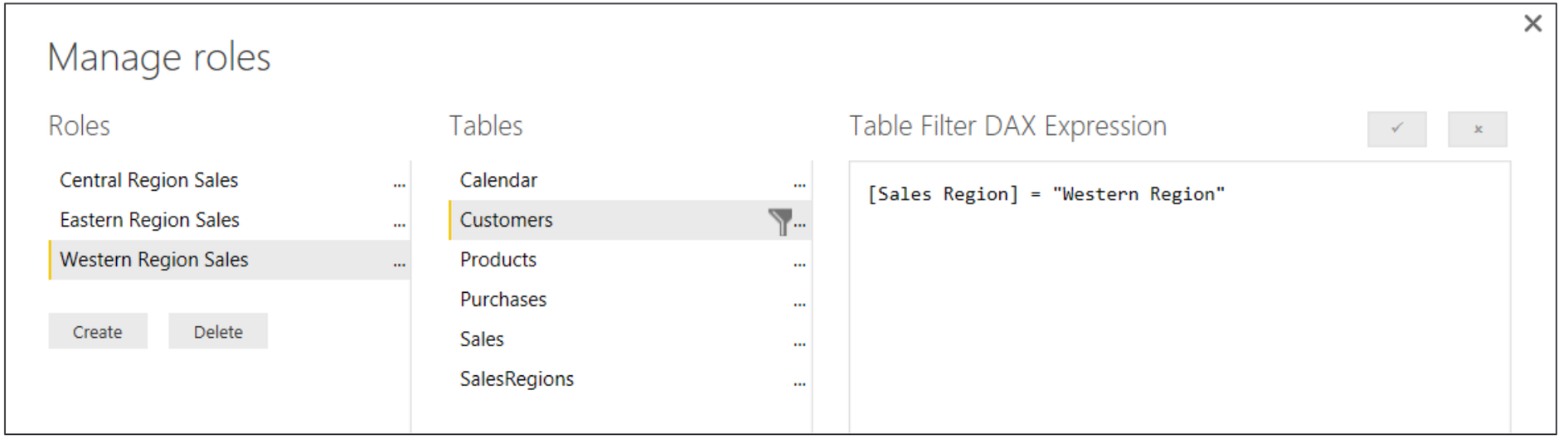
```
Guid workspaceId = new Guid(appWorkspaceId);
var workspaces = (await pbiClient.Groups.GetGroupsAsync()).Value;
var currentWorkspace = workspaces.First((workspace) => workspace.Id == workspaceId);
var datasets = (await pbiClient.Datasets.GetDatasetsInGroupAsync(workspaceId)).Value;
var reports = (await pbiClient.Reports.GetReportsInGroupAsync(workspaceId)).Value;

IList<GenerateTokenRequestV2Dataset> datasetRequests = new List<GenerateTokenRequestV2Dataset>();
foreach (var dataset in datasets) {
  datasetRequests.Add(new GenerateTokenRequestV2Dataset(dataset.Id));
};

IList<GenerateTokenRequestV2Report> reportRequests = new List<GenerateTokenRequestV2Report>();
foreach (var report in reports) {
  reportRequests.Add(new GenerateTokenRequestV2Report(report.Id, allowEdit: true));
};

IList<GenerateTokenRequestV2TargetWorkspace> workspaceRequests =
  new GenerateTokenRequestV2TargetWorkspace[] {
    new GenerateTokenRequestV2TargetWorkspace(workspaceId)
};

GenerateTokenRequestV2 tokenRequest =
  new GenerateTokenRequestV2(datasets: datasetRequests,
                             reports: reportRequests,
                             targetWorkspaces: workspaceRequests);

// call to Power BI Service API and pass GenerateTokenRequest object to generate embed token
string embedToken = pbiClient.EmbedToken.GenerateToken(tokenRequest).Token;
```

Agenda

- ✓ Developing for Power BI with .NET 5
- ✓ Introducing `Microsoft.Identity.Web`
- ✓ Calling to Power BI as Service Principal
- ✓ Programming the Power BI JavaScript API
- ✓ Adding TypeScript Support to a .NET 5 Project
- ✓ Programming with Multi-Resource Embed Tokens
- ➢ **Integrating RLS using EffectiveIdentity**

What Is Row-level Security (RLS)?

- **Security Scheme for Power BI Datasets based on Named Roles**
  - Roles are defined using Power BI Desktop or Tabular Object Model (TOM)
  - Each role is scoped to the dataset
- **Role defined using one or more DAX expressions**
  - DAX expressions restrict which rows are accessible



Embedding RLS-enabled Reports

- **Programming with EffectiveIdentity**
  - Trying to embed an RLS-enabled report without EffectiveIdentity will fail
  - EffectiveIdentity containing 1 or more roles must be added to embed token
  - Roles added into EffectiveIdentity as string array

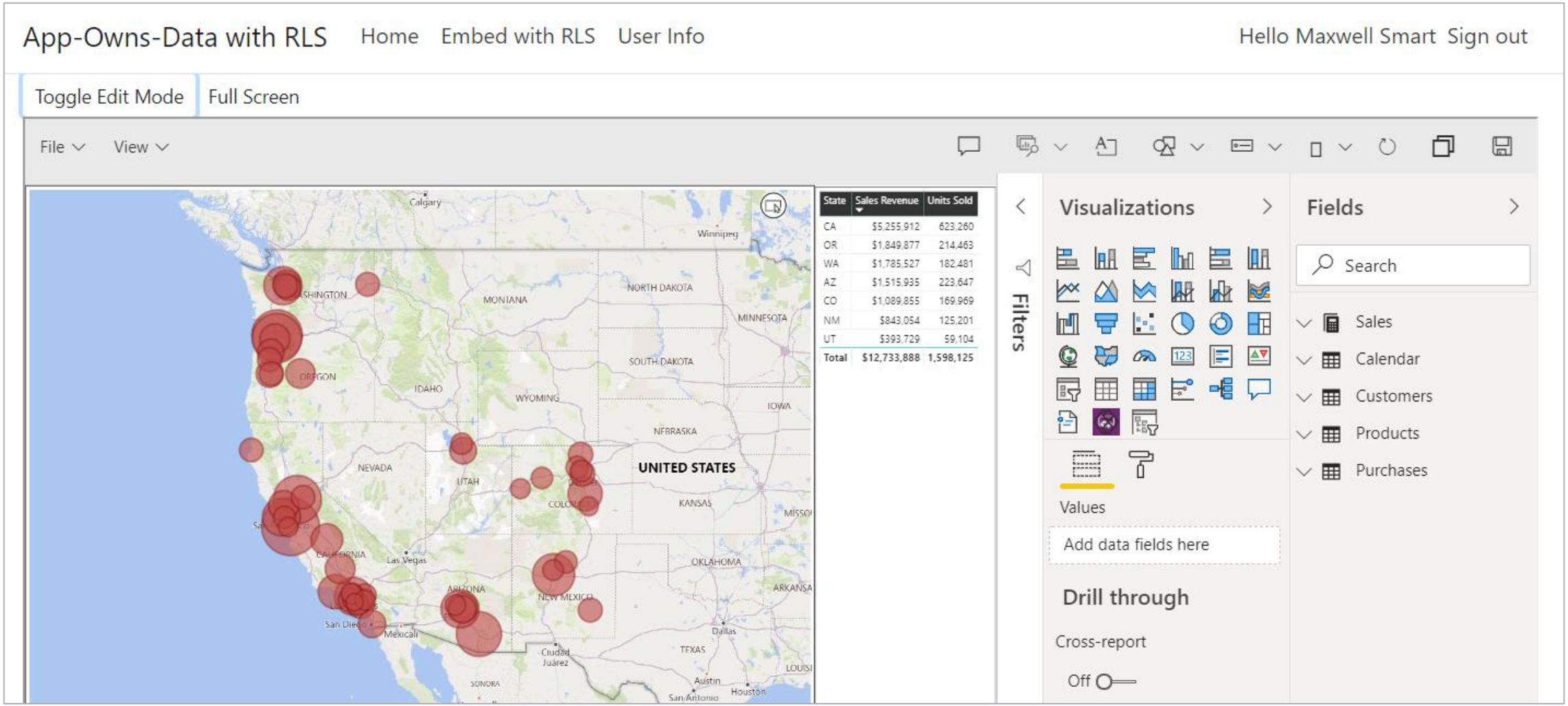
```
GenerateTokenRequest generateTokenRequestParameters =
 new GenerateTokenRequest(accessLevel: accessLevel,
                          identities: new List<EffectiveIdentity> {
                             new EffectiveIdentity(username: currentUser.UserName,
                                                   datasets: new List<string> { datasetId  },
                                                   roles: roles) });

string embedToken =
       (await pbiClient.Reports.GenerateTokenInGroupAsync(workspaceId,
                                                          report.Id,
                                                          generateTokenRequestParameters)).Token;
```

Generating Embed Tokens with EffectiveIdentity

```
public async Task<EmbeddedReportViewModel> GetReportWithRls(string UserName, string[] Roles, bool CustomizationEnabled) {

  PowerBIClient pbiClient = GetPowerBiClient();

  // call to Power BI Service API to get embedding data
  var report = await pbiClient.Reports.GetReportInGroupAsync(WorkspaceId, RlsReportId);

  // generate read-only embed token for the report
  var datasetId = report.DatasetId;
  var datasetList = new List<string>() { report.DatasetId };

  // create EffectiveIdentity object
  var effectiveIdentity = new EffectiveIdentity(UserName, datasetList, Roles);

  // generate embed token
  TokenAccessLevel tokenAccessLevel = CustomizationEnabled ? TokenAccessLevel.Edit : TokenAccessLevel.View;
  var tokenRequest = new GenerateTokenRequest(tokenAccessLevel, datasetId, effectiveIdentity);
  var embedTokenResponse = await pbiClient.Reports.GenerateTokenAsync(WorkspaceId, RlsReportId, tokenRequest);
  var embedToken = embedTokenResponse.Token;

  // return report embedding data to caller
  return new EmbeddedReportViewModel {
    Id = report.Id.ToString(),
    EmbedUrl = report.EmbedUrl,
    Name = report.Name,
    Token = embedToken,
    CustomizationEnabled = CustomizationEnabled
  };
}
```

Sample Application with App-Owns-Data and RLS

https://github.com/PowerBiDevCamp/AppOwnsDataWithRLS



Summary

- ✓ Developing for Power BI with .NET 5
- ✓ Introducing `Microsoft.Identity.Web`
- ✓ Calling to Power BI as Service Principal
- ✓ Programming the Power BI JavaScript API
- ✓ Adding TypeScript Support to a .NET 5 Project
- ✓ Programming with Multi-Resource Embed Tokens
- ✓ Integrating RLS using EffectiveIdentity

Questions