Cosmo DB: Project [I]

Prepared for:

Date:………., 2020

Prepared by: Shruti Sinhaa

Document Name: Lab Manual

Document Number AZ204_04

Contributor:

Bipeen Sinhaa

Contents

* Introduction:

Lab: Constructing a polyglot data solution

# Student lab manual

# Lab scenario:

You have been assigned the task of updating your company's existing retail web application to use more than one data service in Microsoft Azure. Your company's goal is to take advantage of the best data service for each application component. After conducting thorough research, you decide to migrate your inventory database from Azure SQL Database to Azure Cosmos DB.

# Objectives

After you complete this lab, you will be able to:

- Create instances of various database services by using the Azure portal.

- Write C# code to connect to SQL Database.

- Write C# code to connect to Azure Cosmos DB.

# Lab setup

- Estimated time: **45 minutes**

- Windows
- applications

    - Microsoft Edge

    - File Explorer

    - Visual Studio Code

Exercise 1: Creating database resources in Azure

Task 1: Open the Azure portal

- On the taskbar, select the **Microsoft Edge** icon.

- In the open browser window, browse to the Azure portal (portal.azure.com).

- Enter the email address for your Microsoft account, and then select **Next**.

- Enter the password for your Microsoft account, and then select **Sign in**.

> **Note**: If this is your first time signing in to the Azure portal, you will be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

Task 2: Create an Azure SQL Database server resource

- In the Azure portal's navigation pane, select **All services**.

- From the **All services** blade, select **SQL servers**.

- From the **SQL servers** blade, find your list of SQL server instances.

- From the **SQL servers** blade, select **Add**.

- From the **Create SQL Database Server** blade, observe the tabs from the blade, such as **Basics**, **Networking**, and **Additional settings**.

> **Note**: Each tab represents a step in the workflow to create a new Azure SQL Database server. You can select **Review + Create** at any time to skip the remaining tabs.

- From the **Basics** tab, perform the following actions:

- Leave the **Subscription** drop-down list set to its default value.

- In the **Resource group** section, select **Create new**, enter **PolyglotData**, and then select **OK**.

- In the **Server name** text box, enter **polysqlsrvr*[yourname]***.

- In the **Location** drop-down list, select **(US) East US**.
- In the **Server admin login** text box, enter **testuser**.
    - In the **Password** text box, enter **TestPa55w.rd**.
- In the **Confirm password** text box, enter **TestPa55w.rd** again.
- Select **Next: Networking**.
- From the **Networking** tab, perform the following actions:
- In the **Allow Azure services and resources to access this server** section, select **Yes**.
- Select **Review + Create**.
- From the **Review + Create** tab, review the options that you selected during the previous steps.
- Select **Create** to create the SQL Database server by using your specified configuration.

> **Note**: At this point in the lab, we are only creating the Azure SQL logical server. We will create the Azure SQL database instance later in the lab.

> **Note**: Wait for the creation task to complete before you move forward with this lab.

Task 3: Create an Azure Cosmos DB account resource

- In the Azure portal's navigation pane, select **All services**.
- From the **All services** blade, select **Azure Cosmos DB**.
- From the **Azure Cosmos DB** blade, find your list of Azure Cosmos DB instances.
- From the **Azure Cosmos DB** blade, select **Add**.
- From the **Create Azure Cosmos DB Account** blade, observe the tabs from the blade, such as **Basics**, **Network**, and **Tags**.

> **Note**: Each tab represents a step in the workflow to create a new Azure Cosmos DB account. You can select **Review + Create** at any time to skip the remaining tabs.

- From the **Basics** tab, perform the following actions:
- Leave the **Subscription** list set to its default value.
- In the **Resource group** section, select **PolyglotData** from the list.
- In the **AccountName** text box, enter **polycosmos*[yourname]***.
- In the **API** drop-down list, select **Core (SQL)**.
- In the **Notebooks (Preview)** section, select **Off**.
- In the **Apply Free Tier Discount** section, select **Do Not Apply**.
- In the **Location** drop-down list, select the **(US) East US** region.

- In the **Account Type** section, select **Non-Production**.

- In the **Multi-region Writes** section, select **Disable**.

- Select **Review + Create**.

- From the **Review + Create** tab, review the options that you selected during the previous steps.

- Select **Create** to create the Azure Cosmos DB account by using your specified configuration.

> **Note**: Wait for the creation task to complete before you move forward with this lab.

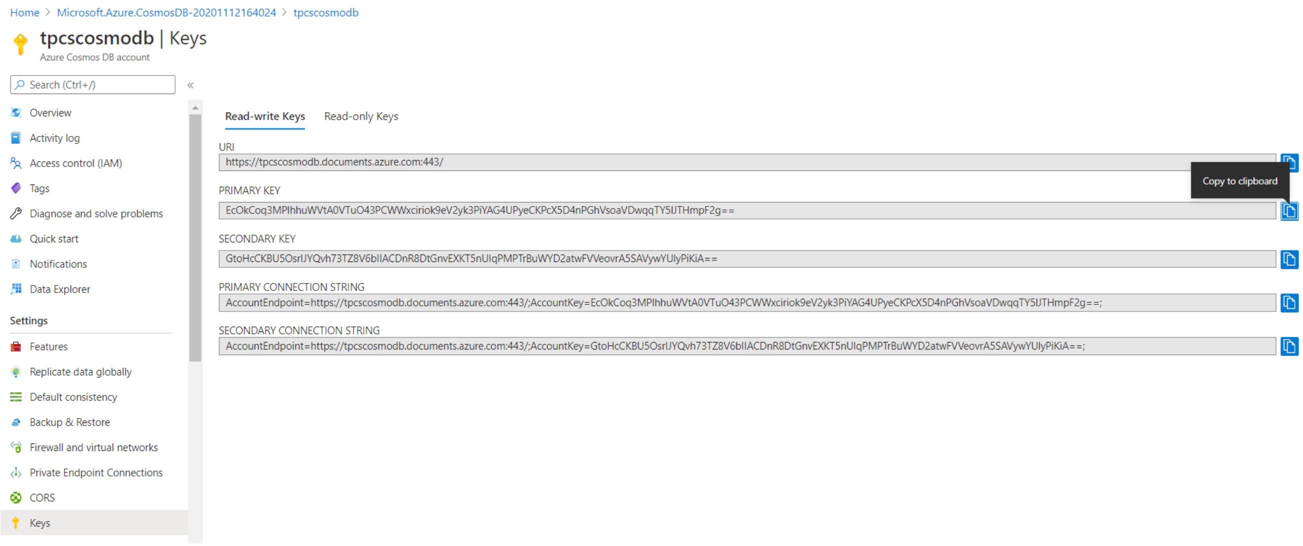
- In the Azure portal's navigation pane, select the **Resource groups** link.

- From the **Resource groups** blade, find and then select the **PolyglotData** resource group that you created earlier in this lab.

- From the **PolyglotData** blade, select the **polycosmos*[yourname]*** Azure Cosmos DB account that you created earlier in this lab.

- From the **Azure Cosmos DB account** blade, find the **Settings** section from the blade, and then select the **Keys** link.

- In the Keys pane, record the value in the **PRIMARY CONNECTION STRING** text box. You'll use this value later in this lab.



URI
https://tpcscosmodb.documents.azure.com:443/

PRIMARY KEY
EcOkCoq3MPlhhuWVtA0VTuO43PCWWxciriok9eV2yk3PiYAG4UPyeCKPcX5D4nPGhVsoaVDwqqTY5IJTHmpF2g==

SECONDARY KEY
GtoHcCKBU5OsrIJYQvh73TZ8V6bIIACDnR8DtGnvEXKT5nUlqPMPTrBuWYD2atwFVVeovrA5SAVywYUlyPiKiA==

PRIMARY CONNECTION STRING
AccountEndpoint=https://tpcscosmodb.documents.azure.com:443/;AccountKey=EcOkCoq3MPlhhuWVtA0VTuO43PCWWxciriok9eV2yk3PiYAG4UPyeCKPcX5D4nPGhVsoaVDwqqTY5IJTHmpF2g==;

SECONDARY CONNECTION STRING
AccountEndpoint=https://tpcscosmodb.documents.azure.com:443/;AccountKey=GtoHcCKBU5OsrIJYQvh73TZ8V6bIIACDnR8DtGnvEXKT5nUlqPMPTrBuWYD2atwFVVeovrA5SAVywYUlyPiKiA==;

Task 4: Create an Azure Storage account resource

- In the Azure portal's navigation pane, select **All services**.

- From the **All services** blade, select **Storage Accounts**.

- From the **Storage accounts** blade, find your list of Storage instances.

- From the **Storage accounts** blade, select **Add**.
- From the **Create storage account** blade, observe the tabs from the blade, such as **Basics**, **Advanced**, and **Tags**.

> **Note**: Each tab represents a step in the workflow to create a new Azure Storage account. You can select **Review + Create** at any time to skip the remaining tabs.

- From the **Basics** tab, perform the following actions:
- Leave the **Subscription** list set to its default value.
- In the **Resource group** section, select **PolyglotData** from the list.
- In the **Storage account name** text box, enter **polystor*[yourname]***.
- In the **Location** drop-down list, select the **(US) East US** region.
- In the **Performance** section, select **Standard**.
- In the **Account kind** drop-down list, select **StorageV2 (general purpose v2)**.
- In the **Replication** drop-down list, select **Locally-redundant storage (LRS)**.
- In the **Access tier (default)** section, ensure that **Hot** is selected.
- Select **Review + Create**.
- From the **Review + Create** tab, review the options that you selected during the previous steps.
- Select **Create** to create the storage account by using your specified configuration.

> **Note**: Wait for the creation task to complete before you move forward with this lab.

Review

In this exercise, you created all the Azure resources that you'll need for a polyglot data solution.

Exercise 2: Import and validate data

Task 1: Upload image blobs

- In the Azure portal's navigation pane, select the **Resource groups** link.
- From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
- From the **PolyglotData** blade, select the **polystor*[yourname]*** storage account that you created earlier in this lab.

- From the **Storage account** blade, select the **Containers** link in the **Blob service** section from the blade.

- In the **Containers** section, select **+ Container**.

- In the **New container** following actions:

    - In the **Name** text box, enter **images**.

    - In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**.

    - Select **OK**.

- Back in the **Containers** section, select the newly created **images** container.

- From the **Container** blade, find the **Settings** section from the blade, and then select the **Properties** link.

- In the Properties pane, record the value in the **URL** text box. You'll use this value later in this lab.

- Find and select the **Overview** link from the blade.

- From the blade, select **Upload**.

- In the **Upload blob** pop-up, perform the following actions:

    - In the **Files** section, select the **Folder** icon.

    - In the **File Explorer** window, browse to **Allfiles (F):\\Allfiles\\Labs\\04\\Starter\\Images**, select all 42 individual **.jpg** image files, and then select **Open**.

    - Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

> **Note**: Wait for all the blobs to upload before you continue with this lab.

Task 2: Upload an SQL .bacpac file

- In the Azure portal's navigation pane, select the **Resource groups** link.

- From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.

- From the **PolyglotData** blade, select the **polystor*[yourname]*** storage account that you created earlier in this lab.

- From the **Storage account** blade, select the **Containers** link in the **Blob service** section from the blade.

    - In the **Containers** section, select **+ Container**.

    - In the **New container** pop-up, perform the following actions:

    - In the **Name** text box, enter **databases**.

    - In the **Public access level** drop-down list, select **Private (no anonymous access)**.

    - Select **OK**.

- Back in the **Containers** section, select the newly created **databases** container.

- From the **Container** blade, select **Upload**.

- In the **Upload blob** pop-up, perform the following actions:

    - In the **Files** section, select the **Folder** icon.

    - In the **File Explorer** window, browse to **Allfiles (F):\\Allfiles\\Labs\\04\\Starter**, select the **AdventureWorks.bacpac** file, and then select **Open**.

    - Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

> **Note**: Wait for the blob to upload before you continue with this lab.

Task 3: Import an SQL database

- In the Azure portal's navigation pane, select the **Resource groups** link.

- From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.

- From the **PolyglotData** blade, select the **polysqlsrvr*[yourname]*** SQL server that you created earlier in this lab.

- From the **SQL server** blade, select **Import database**.

- From the **Import database** blade, perform the following actions:

    - Leave the **Subscription** list set to its default value.

    - Select the **Storage** option.

    - From the **Storage accounts** blade, select the **polystor*[yourname]*** storage account that you created earlier in this lab.

    - From the **Containers** blade, select the **databases** container that you created earlier in this lab.

    - From the **Container** blade, select the **AdventureWorks.bacpac** blob that you created earlier in this lab, and then select **Select** to close the blade.

    - Back from the **Import database** blade, leave the **Pricing tier** option set to its default value.

    - In the **Database name** text box, enter **AdventureWorks**.

    - Leave the **Collation** text box set to its default value.

    - In the **Server admin login** text box, enter **testuser**.

    - In the **Password** text box, enter **TestPa55w.rd**.

    - Select **OK**.

> **Note**: Wait for the database to be created before you continue with this lab. If you receive a firewall-related error on the import step, it means you did not correctly configure the **Allow Azure services to access server** setting on your SQL Server earlier in the lab. Review your settings, delete the empty **AdventureWorks** database, and then attempt your import again.

Task 4: Use an imported SQL database

- In the Azure portal's navigation pane, select the **Resource groups** link.

- From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.

- From the **PolyglotData** blade, select the **polysqlsrvr*[yourname]*** SQL server that you created earlier in this lab.

- From the **SQL server** blade, find the **Security** section from the blade, and then select the **Firewalls and virtual networks** link.

- In the Firewalls and virtual networks pane, perform the following actions:

    - Select **Add client IP**

    - Select **Save**.

    - In the **Success!** confirmation dialog, select **OK**.

> **Note**: This step will ensure that your local machine will have access to the databases that are associated with this server.

- In the Azure portal's navigation pane, select the **Resource groups** link.

- From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.

- From the **PolyglotData** blade, select the **AdventureWorks** SQL database that you created earlier in this lab.

- From the **SQL database** blade, find the **Settings** section from the blade, and then select the **Connection strings** link.

- In the Connection strings pane, record the value in the **ADO.NET (SQL Authentication)** text box. You'll use this value later in this lab.

- Update the connection string that you recorded by performing the following actions:

    - Within the connection string, find the *your_username* placeholder and replace it with **testuser**.

    - Within the connection string, find the *your_password* placeholder and replace it with **TestPa55w.rd**.

> **Note**:

For example, if your connection string was originally ``Server=tcp:polysqlsrvrinstructor.database.windows.net,1433;Initial Catalog=AdventureWorks;User ID={your_username};Password={your_password};``,

your updated connection string will be

``Server=tcp:polysqlsrvrinstructor.database.windows.net,1433;Initial Catalog=AdventureWorks;User ID=testuser;Password=TestPa55w.rd;``

- Find and select the **Query editor (preview)** link from the blade.
- In the Query editor pane, perform the following actions:
    - In the **Login** text box, enter **testuser**.
    - In the **Password** text box, enter **TestPa55w.rd**.
    - Select **OK**.
- In the open query editor, enter the following query:

```
SELECT * FROM AdventureWorks.dbo.Models
```

- Select **Run** to run the query, and then observe the results.

> **Note**: This query will return a list of models from the home page of the web application.

- In the query editor, replace the existing query with the following query:

```
SELECT * FROM AdventureWorks.dbo.Products
```

- Select **Run** to run the query, and then observe the results.

> **Note**: This query will return a list of products that are associated with each model.

Review: In this exercise, you imported all the resources that you'll use with your web application.

Exercise 3: Open and configure a .NET web application

Task 1: Open and build the web application

- On the **Start** screen, select the **Visual Studio Code** tile.

- From the **File** menu, select **Open Folder**.

- In the **File Explorer** window that opens, browse to **Allfiles (F):\\Allfiles\\Labs\\04\\Starter\\AdventureWorks**, and then select **Select Folder**.

- In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

- At the open command prompt, enter the following command, and then select Enter to build the .NET web application:

```
dotnet build
```

> **Note**: The **dotnet build** command will automatically restore any missing NuGet packages prior to building all projects in the folder.

- Observe the results of the build printed in the terminal. The build should complete successfully with no errors or warning messages.

- Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

Task 2: Update the SQL connection string

- In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.

- Open the **appsettings.json** file.

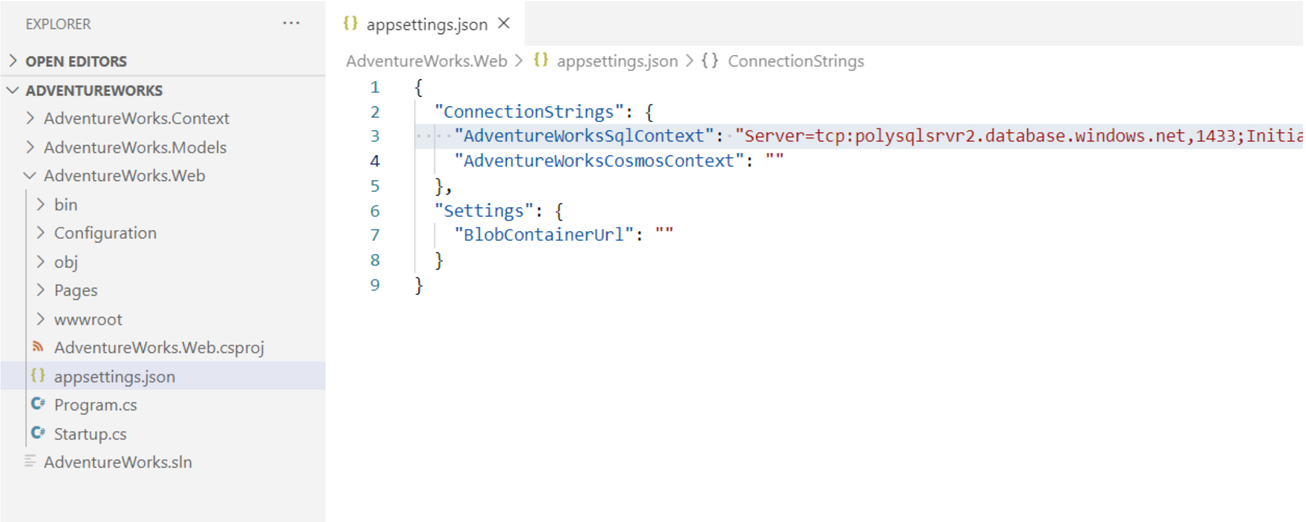
- In the JavaScript Object Notation (JSON) object on line 3, find the **ConnectionStrings.AdventureWorksSqlContext** path. Observe that the current value is empty:

```
"ConnectionStrings": {
        "AdventureWorksSqlContext": "",
        ...
    },
```

-
- Update the value of the **AdventureWorksSqlContext** property by setting its value to
- the **ADO.NET (SQL Authentication) connection string** of the SQL database that you recorded earlier in this lab.

Note: It's important that you use your updated connection string here. The original connection string copied from the portal won't have the username and password necessary to connect to the SQL database.

- Save the **appsettings.json** file.



```
1  {
2    "ConnectionStrings": {
3      "AdventureWorksSqlContext": "Server=tcp:polysqlsrvr2.database.windows.net,1433;Initia
4      "AdventureWorksCosmosContext": ""
5    },
6    "Settings": {
7      "BlobContainerUrl": ""
8    }
9  }
```

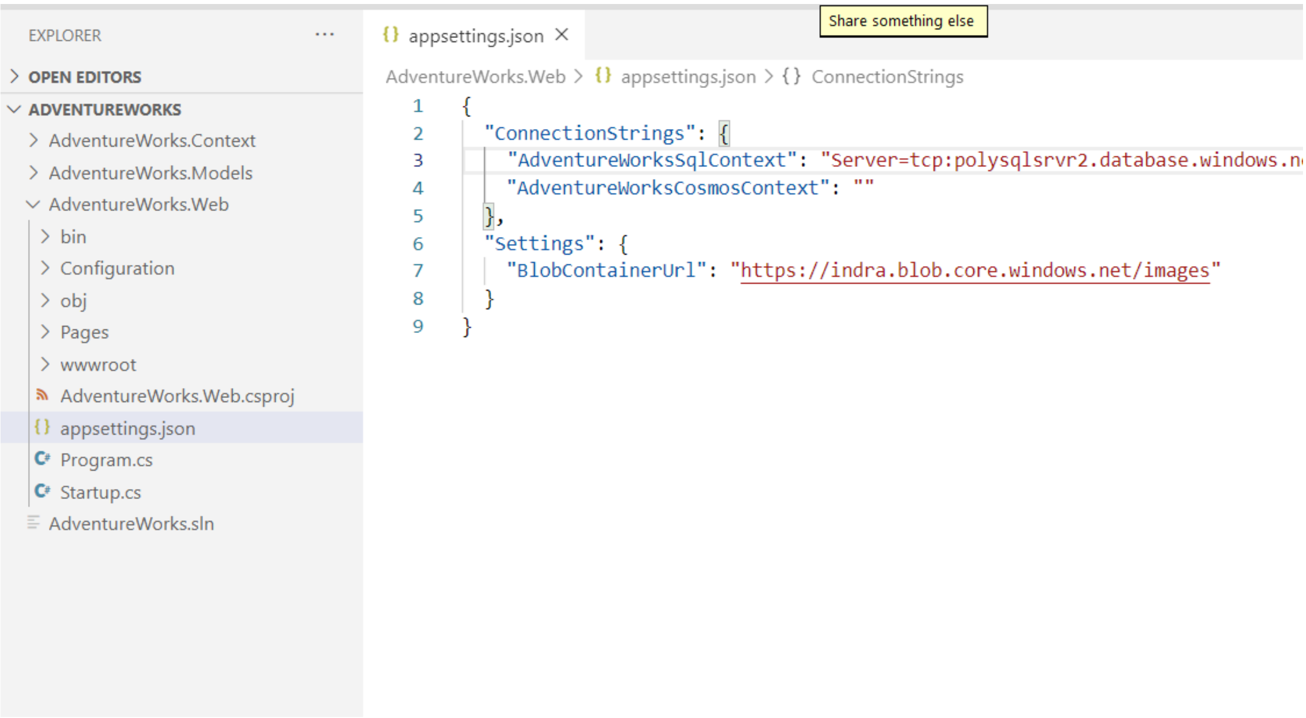
Task 3: Update the blob base URL

- In the JSON object on line 8, find the **Settings.BlobContainerUrl** path. Observe that the current value is empty:

```
"Settings": {
    "BlobContainerUrl": "",
    ...
}
```

- Update the value of the **BlobContainerUrl** property by setting its value to the **URL** property of the Azure Storage blob container named **images** that you recorded earlier in this lab.

- Save the **appsettings.json** file.



```json
{
  "ConnectionStrings": {
    "AdventureWorksSqlContext": "Server=tcp:polysqlsrvr2.database.windows.n
    "AdventureWorksCosmosContext": ""
  },
  "Settings": {
    "BlobContainerUrl": "https://indra.blob.core.windows.net/images"
  }
}
```

Task 4: Validate the web application

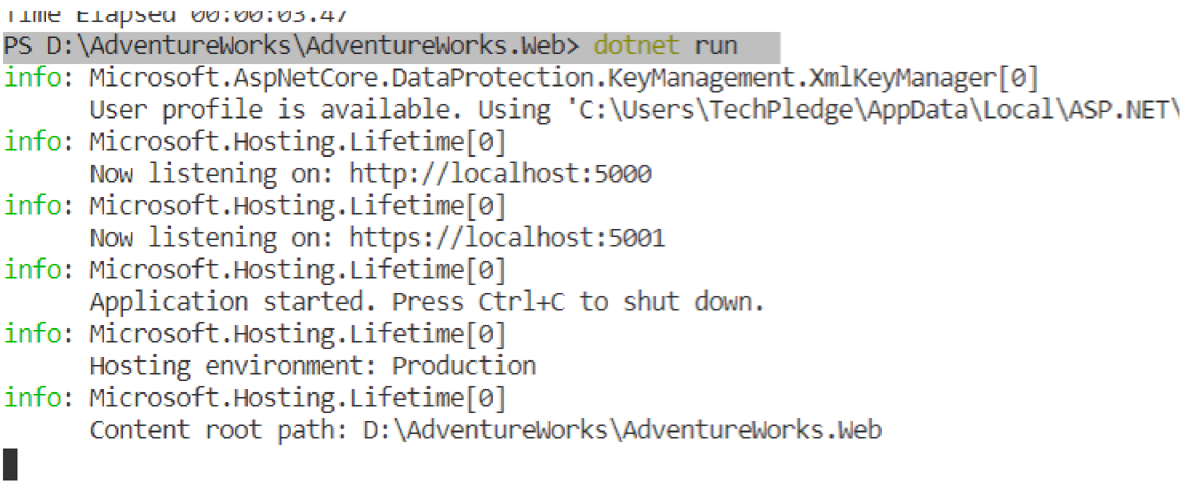
- In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

- At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Web** folder:

```
cd .\AdventureWorks.Web\
```

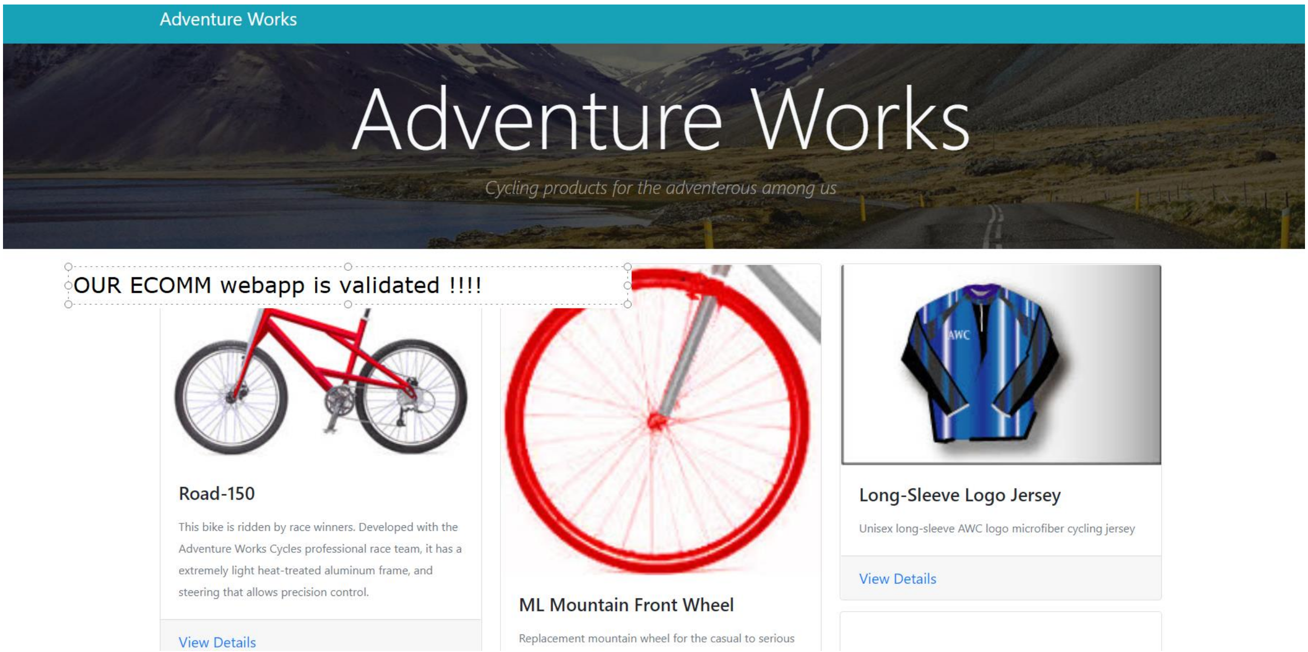
- At the command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```



```
Time Elapsed 00:00:03.47
PS D:\AdventureWorks\AdventureWorks.Web> dotnet run
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\TechPledge\AppData\Local\ASP.NET\
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\AdventureWorks\AdventureWorks.Web
█
```

```

> **Note**: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

1. On the taskbar, select the **Microsoft Edge** icon.

    1. In the open browser window, browse to the currently running web application (<http://localhost:5000>).

1. In the web application, observe the list of models displayed from the front page.

1. Find the **Water Bottle** model, and then select **View Details**.

1. From the **Water Bottle** product detail page, find **Add to Cart**, and then observe that the checkout functionality is currently disabled.

1. Close the browser window displaying your web application.

1. Return to the **Visual Studio Code** window, and then select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Review

In this exercise, you configured your ASP.NET web application to connect to your resources in Azure.
```



Adventure Works

Adventure Works

Cycling products for the adventerous among us

OUR ECOMM webapp is validated !!!!

Road-150

This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has a extremely light heat-treated aluminum frame, and steering that allows precision control.

View Details

ML Mountain Front Wheel

Replacement mountain wheel for the casual to serious

Long-Sleeve Logo Jersey

Unisex long-sleeve AWC logo microfiber cycling jersey

View Details

Exercise 4: Migrating SQL data to Azure Cosmos DB

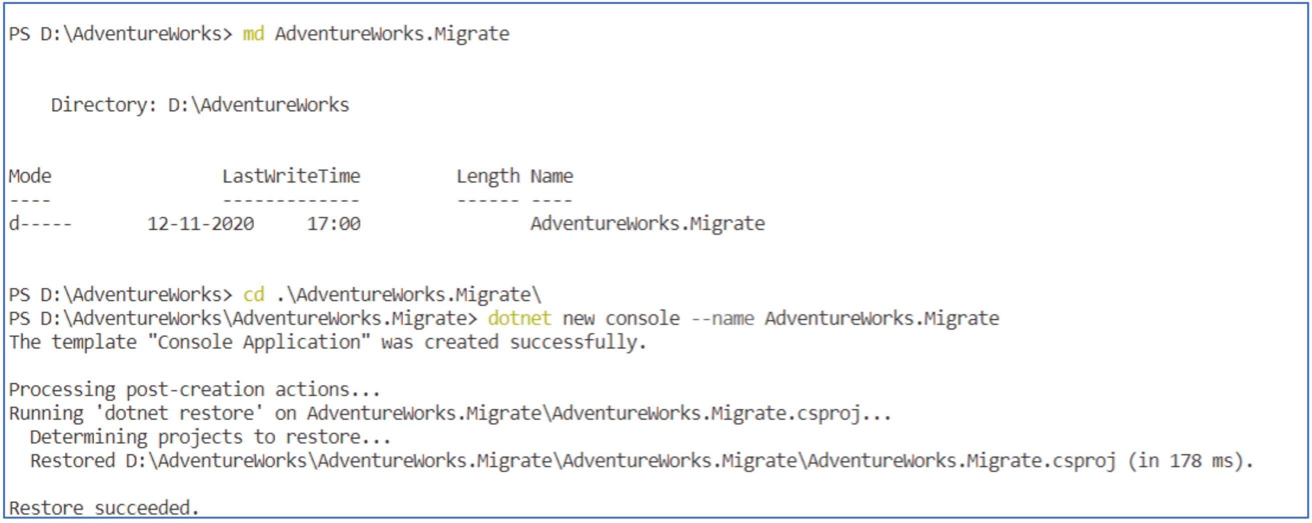
#### Task 1: Create a migration project

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to create a new .NET console project named **AdventureWorks.Migrate** in a folder with the same name:

```
dotnet new console --name AdventureWorks.Migrate
```

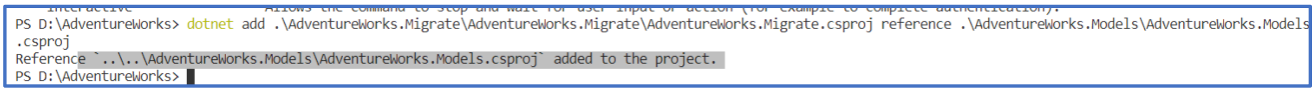
> **Note**: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.



```
PS D:\AdventureWorks> md AdventureWorks.Migrate


    Directory: D:\AdventureWorks


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        12-11-2020     17:00                AdventureWorks.Migrate


PS D:\AdventureWorks> cd .\AdventureWorks.Migrate\
PS D:\AdventureWorks\AdventureWorks.Migrate> dotnet new console --name AdventureWorks.Migrate
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on AdventureWorks.Migrate\AdventureWorks.Migrate.csproj...
  Determining projects to restore...
  Restored D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj (in 178 ms).

Restore succeeded.
```

1. At the command prompt, enter the following command, and then select Enter to add a reference to the existing **AdventureWorks.Models** project:

```
dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Models\AdventureWorks.Models.csproj
```

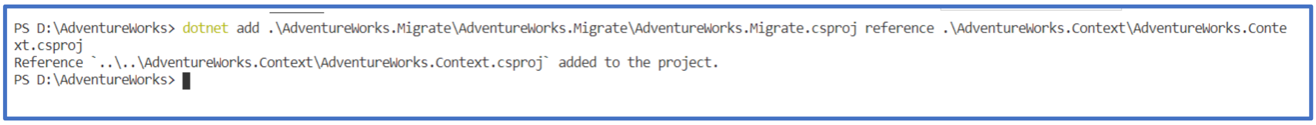


```
  Interactive                Allows the command to stop and wait for user input or action (for example to complete authentication).
PS D:\AdventureWorks> dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Models\AdventureWorks.Models
.csproj
Reference `..\..\AdventureWorks.Models\AdventureWorks.Models.csproj` added to the project.
PS D:\AdventureWorks>
```

> **Note**: The **dotnet add reference** command will add a reference to the model classes contained in the **AdventureWorks.Models** project.

1. At the command prompt, enter the following command, and then select Enter to add a reference to the existing **AdventureWorks.Context** project:

```
dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Context\AdventureWorks.Context.csproj
```



```
PS D:\AdventureWorks> dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Context\AdventureWorks.Conte
xt.csproj
Reference `..\..\AdventureWorks.Context\AdventureWorks.Context.csproj` added to the project.
PS D:\AdventureWorks>
```

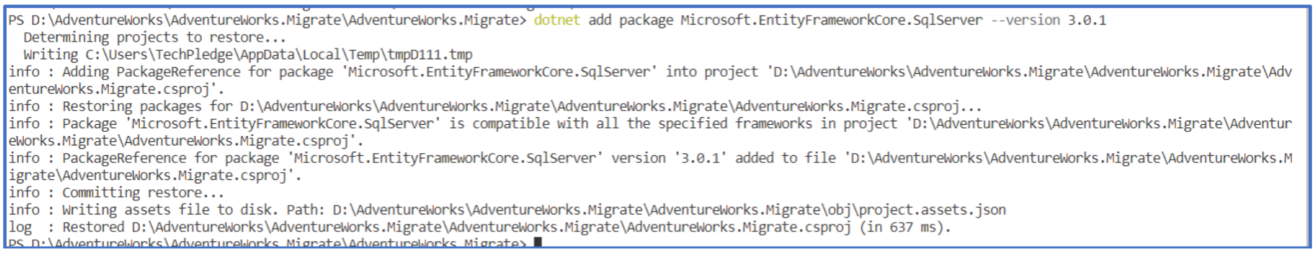
> **Note**: The **dotnet add reference** command will add a reference to the context classes contained in the **AdventureWorks.Context** project.

1. At the command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

1. At the command prompt, enter the following command, and then select Enter to import version 2.2.6 of **Microsoft.EntityFrameworkCore.SqlServer** from NuGet:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 3.0.1
```

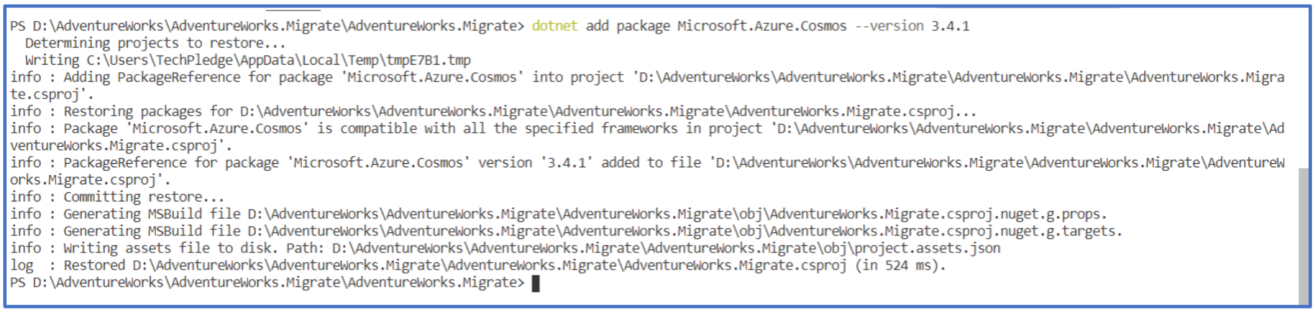


```
PS D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate> dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 3.0.1
  Determining projects to restore...
  Writing C:\Users\TechPledge\AppData\Local\Temp\tmpD111.tmp
info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore.SqlServer' into project 'D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\Adv
entureWorks.Migrate.csproj'.
info : Restoring packages for D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj...
info : Package 'Microsoft.EntityFrameworkCore.SqlServer' is compatible with all the specified frameworks in project 'D:\AdventureWorks\AdventureWorks.Migrate\Adventur
eWorks.Migrate\AdventureWorks.Migrate.csproj'.
info : PackageReference for package 'Microsoft.EntityFrameworkCore.SqlServer' version '3.0.1' added to file 'D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.M
igrate\AdventureWorks.Migrate.csproj'.
info : Committing restore...
info : Writing assets file to disk. Path: D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\obj\project.assets.json
log  : Restored D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj (in 637 ms).
PS D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate>
```

> **Note**: The **dotnet add package** command will add the **Microsoft.EntityFrameworkCore.SqlServer** package from **NuGet**. For more information, go to: [Microsoft.EntityFrameworkCore.SqlServer](https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer/3.0.1).

1. At the command prompt, enter the following command, and then select Enter to import version 3.4.1 of **Microsoft.Azure.Cosmos** from NuGet:

```
dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
```



```
PS D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate> dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
  Determining projects to restore...
  Writing C:\Users\TechPledge\AppData\Local\Temp\tmpE7B1.tmp
info : Adding PackageReference for package 'Microsoft.Azure.Cosmos' into project 'D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migra
te.csproj'.
info : Restoring packages for D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj...
info : Package 'Microsoft.Azure.Cosmos' is compatible with all the specified frameworks in project 'D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\Ad
ventureWorks.Migrate.csproj'.
info : PackageReference for package 'Microsoft.Azure.Cosmos' version '3.4.1' added to file 'D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureW
orks.Migrate.csproj'.
info : Committing restore...
info : Generating MSBuild file D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\obj\AdventureWorks.Migrate.csproj.nuget.g.props.
info : Generating MSBuild file D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\obj\AdventureWorks.Migrate.csproj.nuget.g.targets.
info : Writing assets file to disk. Path: D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\obj\project.assets.json
log  : Restored D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj (in 524 ms).
PS D:\AdventureWorks\AdventureWorks.Migrate\AdventureWorks.Migrate>
```

> **Note**: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, go to: [Microsoft.Azure.Cosmos](https://www.nuget.org/packages/Microsoft.Azure.Cosmos/3.4.1).

1. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

```
dotnet build
```

1. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Task 2: Create a .NET class

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Migrate** project.

1. Open the **Program.cs** file.

1. From the code editor tab for the **Program.cs** file, delete all the code in the existing file.

1. Add the following lines of code to import the **AdventureWorks.Models** and **AdventureWorks.Context** namespaces from the referenced **AdventureWorks.Models** and **AdventureWorks.Context** projects:

```
using AdventureWorks.Context;
using AdventureWorks.Models;
```

1. Add the following line of code to import the **Microsoft.Azure.Cosmos** namespace from the **Microsoft.Azure.Cosmos** package imported from NuGet:

```
using Microsoft.Azure.Cosmos;
```

1. Add the following line of code to import the **Microsoft.EntityFrameworkCore** namespace from the **Microsoft.EntityFrameworkCore.SqlServer** package imported from NuGet:

```
using Microsoft.EntityFrameworkCore;
```

1. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

1. Enter the following code to create a new **Program** class:

```
public class Program
{
}
```

1. Within the **Program** class, enter the following line of code to create a new string constant named **sqlDBConnectionString**:

```
private const string sqlDBConnectionString = "";
```

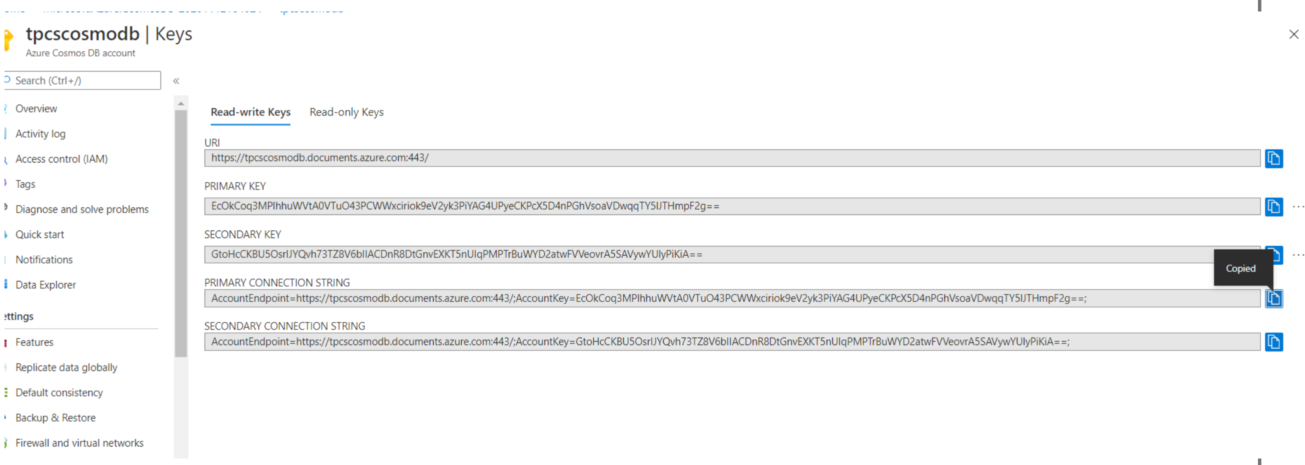
1. Update the **sqlDBConnectionString** string constant by setting its value to the **ADO.NET (SQL Authentication) connection string** of the SQL database that you recorded earlier in this lab.

> **Note**: It's important that you use your updated connection string here. The original connection string copied from the portal won't have the username and password necessary to connect to the SQL database.

1. Within the **Program** class, enter the following line of code to create a new string constant named **cosmosDBConnectionString**:

```
private const string cosmosDBConnectionString = "";
```

1. Update the **cosmosDBConnectionString** string constant by setting its value to the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.



URI
https://tpcscosmodb.documents.azure.com:443/

PRIMARY KEY
EcOkCoq3MPlhhuWVtA0VTuO43PCWWxciriok9eV2yk3PiYAG4UPyeCKPcX5D4nPGhVsoaVDwqqTY5IJTHmpF2g==

SECONDARY KEY
GtoHcCKBU5OsrlJYQvh73TZ8V6bIIACDnR8DtGnvEXKT5nUIqPMPTrBuWYD2atwFVVeovrA5SAVywYUIyPiKiA==

PRIMARY CONNECTION STRING
AccountEndpoint=https://tpcscosmodb.documents.azure.com:443/;AccountKey=EcOkCoq3MPlhhuWVtA0VTuO43PCWWxciriok9eV2yk3PiYAG4UPyeCKPcX5D4nPGhVsoaVDwqqTY5IJTHmpF2g==;

SECONDARY CONNECTION STRING
AccountEndpoint=https://tpcscosmodb.documents.azure.com:443/;AccountKey=GtoHcCKBU5OsrlJYQvh73TZ8V6bIIACDnR8DtGnvEXKT5nUIqPMPTrBuWYD2atwFVVeovrA5SAVywYUIyPiKiA==;

2.

1. Within the **Program** class, enter the following code to create a new asynchronous **Main** method:

    ```
    public static async Task Main(string[] args)
    {
    }
    ```

1. Within the **Main** method, add the following line of code to print an introductory message to the console:

    ```
    await Console.Out.WriteLineAsync("Start Migration");
    ```

1. Save the **Program.cs** file.

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

    ```
    cd .\AdventureWorks.Migrate\
    ```

1. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

    ```
    dotnet build
    ```

1. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Task 3: Get SQL database records by using Entity Framework

1. Within the **Main** method of the **Program** class within the **Program.cs** file, add the following line of code to create a new instance of the **AdventureWorksSqlContext** class, passing in the *sqlDBConnectionString* variable as the connection string value:

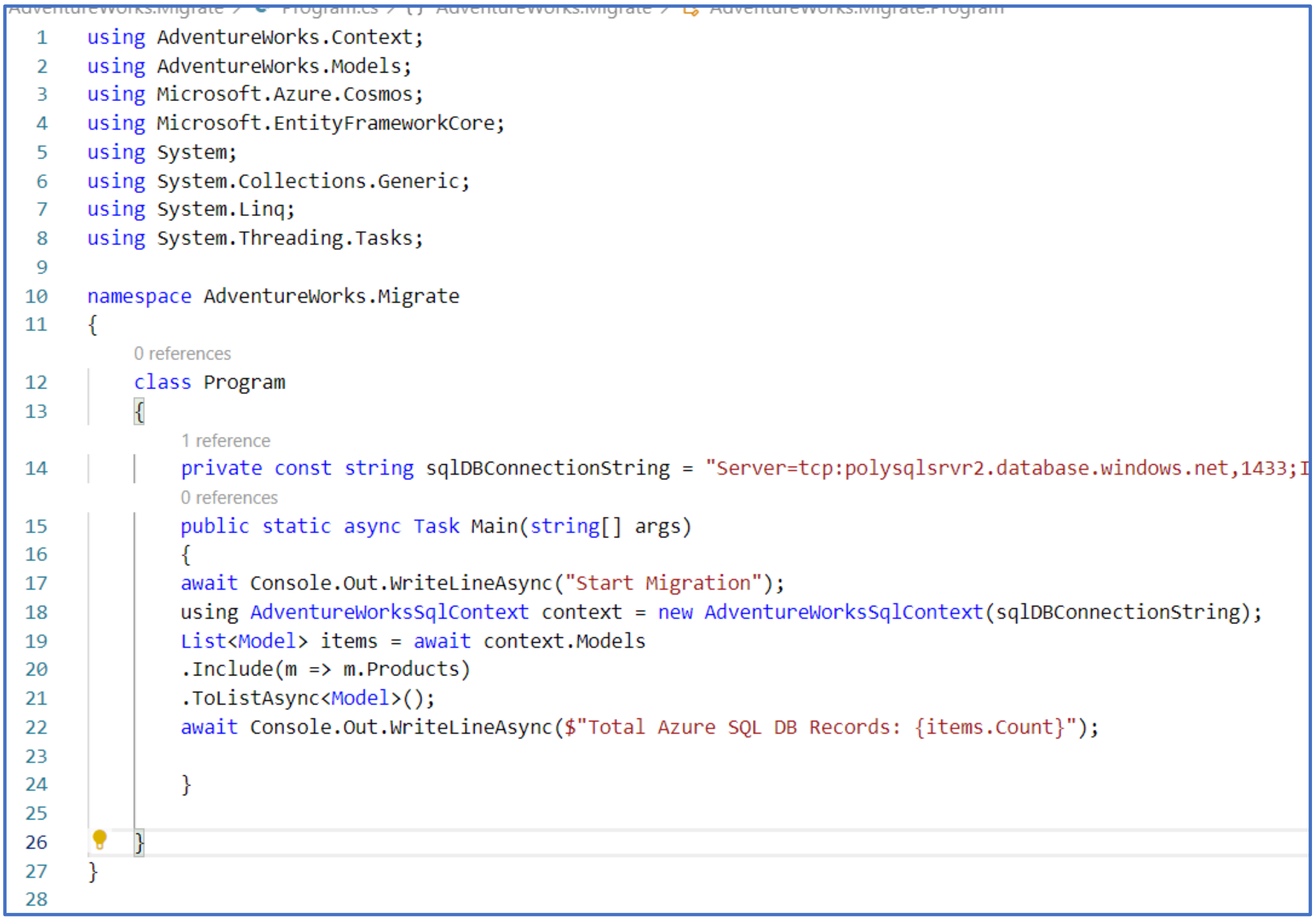
```
using AdventureWorksSqlContext context = new AdventureWorksSqlContext(sqlDBConnectionString);
```

1. Within the **Main** method, add the following block of code to issue a language-integrated query (LINQ) to get all **Models** and child **Products** from the database and store them in an in-memory **List<>** collection:

```
List<Model> items = await context.Models
    .Include(m => m.Products)
    .ToListAsync<Model>();
```

1. Within the **Main** method, add the following line of code to print the number of records imported from SQL Database:

```
await Console.Out.WriteLineAsync($"Total Azure SQL DB Records: {items.Count}");
```


AdventureWorks.Migrate > Program.cs > {} AdventureWorks.Migrate > AdventureWorks.Migrate.Program
0 references
1 reference
0 references

```
using AdventureWorks.Context;
using AdventureWorks.Models;
using Microsoft.Azure.Cosmos;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AdventureWorks.Migrate
{
    class Program
    {
        private const string sqlDBConnectionString = "Server=tcp:polysqlsrvr2.database.windows.net,1433;I
        public static async Task Main(string[] args)
        {
        await Console.Out.WriteLineAsync("Start Migration");
        using AdventureWorksSqlContext context = new AdventureWorksSqlContext(sqlDBConnectionString);
        List<Model> items = await context.Models
        .Include(m => m.Products)
        .ToListAsync<Model>();
        await Console.Out.WriteLineAsync($"Total Azure SQL DB Records: {items.Count}");

        }

    }
}

```

1. Save the **Program.cs** file.

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

1. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

```
dotnet build
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\\Allfiles\\Labs\\04\\Solution\\AdventureWorks\\AdventureWorks.Migrate** folder.

1. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Task 4: Insert items into Azure Cosmos DB

1. Within the **Main** method of the **Program** class within the **Program.cs** file, add the following line of code to create a new instance of the **CosmosClient** class, passing in the *cosmosDBConnectionString* variable as the connection string value:

```
using CosmosClient client = new CosmosClient(cosmosDBConnectionString);
```

1. Within the **Main** method, add the following line of code to create a new **database** named **Retail** if it doesn't already exist in the Azure Cosmos DB account:

```
Database database = await client.CreateDatabaseIfNotExistsAsync("Retail");
```

1. Within the **Main** method, add the following block of code to create a new **container** named **Online** if it doesn't already exist in the Azure Cosmos DB account with a partition key path of **/Category** and a throughput of **1000** Request Units:

```
Container container = await database.CreateContainerIfNotExistsAsync("Online",
    partitionKeyPath: $"/{nameof(Model.Category)}",
    throughput: 1000
);
```

1. Within the **Main** method, add the following line of code to create an *int* variable named **count**:

```
int count = 0;
```

1. Within the **Main** method, add the following block of code to create a **foreach** loop that iterates over the objects in the **items** collection:

```
foreach (var item in items)
{
}
```

```

1.  Within the **foreach** loop in the **Main** method, add the following line of code to **upsert** the object into the Azure Cosmos DB collection and save the result in a variable of type *ItemResponse<>* named **document**:

   ```
   ItemResponse<Model> document = await container.UpsertItemAsync<Model>(item);
   ```

1.  Within the **foreach** loop contained in the **Main** method, add the following line of code to print the activity ID of each upsert operation:

   ```
   await Console.Out.WriteLineAsync($"Upserted document #{++count:000} [Activity Id: {document.ActivityId}]");
   ```

1. Back within the **Main** method (outside of the **foreach** loop), add the following line of code to print the number of documents exported to Azure Cosmos DB:

   ```
   await Console.Out.WriteLineAsync($"Total Azure Cosmos DB Documents: {count}");
   ```

1. Save the **Program.cs** file.

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

   ```
   cd .\AdventureWorks.Migrate\
   ```

1. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

   ```
   dotnet build
   ```

   > **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\\Allfiles\\Labs\\04\\Solution\\AdventureWorks\\AdventureWorks.Migrate** folder.

```
using System;
using AdventureWorks.Context;
using AdventureWorks.Models;
using Microsoft.Azure.Cosmos;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```
```

```
namespace AdventureWorks.Migrate
{
    public class Program
    {
        private const string sqlDBConnectionString = "Server=tcp:polysqlsrvrshruti
.database.windows.net,1433;Initial Catalog=AdventureWorks;Persist Security Info=Fa
lse;User ID=Techpledge;Password=Password@123;MultipleActiveResultSets=False;Encryp
t=True;TrustServerCertificate=False;Connection Timeout=30;";
        private const string cosmosDBConnectionString = "AccountEndpoint=https://t
pcscosmodb.documents.azure.com:443/;AccountKey=EcOkCoq3MPlhhuWVtA0VTuO43PCWWxcirio
k9eV2yk3PiYAG4UPyeCKPcX5D4nPGhVsoaVDwqqTY5lJTHmpF2g==;";

            public static async Task Main(string[] args)
    {
        await Console.Out.WriteLineAsync("Start Migration");
        using AdventureWorksSqlContext context = new AdventureWorksSqlContext(sqlD
BConnectionString);
        List<Model> items = await context.Models
        .Include(m => m.Products)
        .ToListAsync<Model>();
        await Console.Out.WriteLineAsync($"Total Azure SQL DB Records: {items.Coun
t}");
        using CosmosClient client = new CosmosClient(cosmosDBConnectionString);
        Database database = await client.CreateDatabaseIfNotExistsAsync("Retail");
        Container container = await database.CreateContainerIfNotExistsAsync("Onli
ne",
        partitionKeyPath: $"/{nameof(Model.Category)}",
        throughput: 1000
        );
        int count = 0;
        foreach (var item in items)
    {
        ItemResponse<Model> document = await container.UpsertItemAsync<Model>(item
);
        await Console.Out.WriteLineAsync($"Upserted document #{++count:000} [Activ
ity Id: {document.ActivityId}]");
        await Console.Out.WriteLineAsync($"Total Azure Cosmos DB Documents: {count
}");

    }

    }



    }
}
```

#### Task 5: Perform a migration

1. At the open command prompt, enter the following command, and then select Enter to run the .NET console application:

    ```
    dotnet run
    ```

    > **Note**: The **dotnet run** command will start the console application.

1. Observe the various data that prints to the screen, including initial SQL record count, individual upsert activity identifiers, and final Azure Cosmos DB document count.

1. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Task 6: Validate the migration

1. Return to the **Microsoft Edge** browser window with the Azure portal.

1. In the Azure portal's navigation pane, select the **Resource groups** link.

1. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.

1. From the **PolyglotData** blade, select the **polycosmos*[yourname]*** Azure Cosmos DB account that you created earlier in this lab.

1. From the **Azure Cosmos DB account** blade, find and select the **Data Explorer** link from the blade.

1. In the Data Explorer pane, expand the **Retail** database node.

1. Expand the **Online** container node, and then select **New SQL Query**.

    > **Note**: The label for this option might be hidden. You can get labels by hovering over the icons in the Data Explorer pane.

1. From the query tab, enter the following text:

    ```
    SELECT * FROM models
    ```

1. Select **Execute Query**, and then observe the list of JSON models that the query returns.

1. Back in the query editor, replace the existing text with the following text:

```
SELECT VALUE COUNT(1) FROM models
```

1. Select **Execute Query**, and then observe the result of the **COUNT** aggregate operation.

1. Return to the **Visual Studio Code** window.

#### Review

In this exercise, you used Entity Framework and the .NET SDK for Azure Cosmos DB to migrate data from SQL Database to Azure Cosmos DB.

Exercise 5: Accessing Azure Cosmos DB by using .NET

#### Task 1: Update library with the Cosmos SDK and references

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Context** folder:

```
cd .\AdventureWorks.Context\
```

1. At the command prompt, enter the following command, and then select Enter to import **Microsoft.Azure.Cosmos** from NuGet:

```
dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
```

> **Note**: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, go to: [Microsoft.Azure.Cosmos](https://www.nuget.org/packages/Microsoft.Azure.Cosmos/3.4.1).

1. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

```
dotnet build
```

1. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Task 2: Write .NET code to connect to Azure Cosmos DB

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Context** project.

1. Access the shortcut menu or right-click or activate the shortcut menu for the **AdventureWorks.Context** folder node, and then select **New File**.

1. At the new file prompt, enter **AdventureWorksCosmosContext.cs**.

1. From the code editor tab for the **AdventureWorksCosmosContext.cs** file, add the following lines of code to import the **AdventureWorks.Models** namespace from the referenced **AdventureWorks.Models** project:

    ```
    using AdventureWorks.Models;
    ```

1. Add the following lines of code to import the **Microsoft.Azure.Cosmos** and **Microsoft.Azure.Cosmos.Linq** namespaces from the **Microsoft.Azure.Cosmos** package imported from NuGet:

    ```
    using Microsoft.Azure.Cosmos;
    using Microsoft.Azure.Cosmos.Linq;
    ```

1. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

    ```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;
    ```

1. Enter the following code to add an **AdventureWorks.Context** namespace block:

    ```
    namespace AdventureWorks.Context
    {
    }
    ```

1. Within the **AdventureWorks.Context** namespace, enter the following code to create a new **AdventureWorksCosmosContext** class:

    ```
    public class AdventureWorksCosmosContext
    {
    }
    ```

1. Update the declaration of the **AdventureWorksCosmosContext** class by adding a specification indicating that this class will implement the **IAdventureWorksProductContext** interface:

    ```
    public class AdventureWorksCosmosContext : IAdventureWorksProductContext
    {
    ```

```
}
```

1. Within the **AdventureWorksCosmosContext** class, enter the following line of code to create a new read-only *Container* variable named **_container**:

```
private readonly Container _container;
```

1. Within the **AdventureWorksCosmosContext** class, add a new constructor with the following signature:

```
public AdventureWorksCosmosContext(string connectionString, string database = "Retail", string container = "Online")
{
}
```

1. Within the constructor, add the following block of code to create a new instance of the **CosmosClient** class and then obtain both a **Database** and **Container** instance from the client:

```
_container = new CosmosClient(connectionString)
    .GetDatabase(database)
    .GetContainer(container);
```

1. Within the **AdventureWorksCosmosContext** class, add a new **FindModelAsync** method with the following signature:

```
public async Task<Model> FindModelAsync(Guid id)
{
}
```

1. Within the **FindModelAsync** method, add the following blocks of code to create a LINQ query, transform it into an iterator, iterate over the result set, and then return the single item in the result set:

```
var iterator = _container.GetItemLinqQueryable<Model>()
    .Where(m => m.id == id)
    .ToFeedIterator<Model>();

List<Model> matches = new List<Model>();
while (iterator.HasMoreResults)
{
    var next = await iterator.ReadNextAsync();
    matches.AddRange(next);
}

return matches.SingleOrDefault();
```

1. Within the **AdventureWorksCosmosContext** class, add a new **GetModelsAsync** method with the following signature:

```
public async Task<List<Model>> GetModelsAsync()
{
}
```

1. Within the **GetModelsAsync** method, add the following blocks of code to run an SQL query, get the query result iterator, iterate over the result set, and then return the union of all results:

```
string query = $@"SELECT * FROM items";

var iterator = _container.GetItemQueryIterator<Model>(query);

List<Model> matches = new List<Model>();
while (iterator.HasMoreResults)
{
    var next = await iterator.ReadNextAsync();
    matches.AddRange(next);
}

return matches;
```

1. Within the **AdventureWorksCosmosContext** class, add a new **FindProductAsync** method with the following signature:

```
public async Task<Product> FindProductAsync(Guid id)
{
}
```

1. Within the **FindProductAsync** method, add the following blocks of code to run an SQL query, get the query result iterator, iterate over the result set, and then return the single item in the result set:

```
string query = $@"SELECT VALUE products
                    FROM models
                    JOIN products in models.Products
                    WHERE products.id = '{id}'";

var iterator = _container.GetItemQueryIterator<Product>(query);

List<Product> matches = new List<Product>();
while (iterator.HasMoreResults)
{
    var next = await iterator.ReadNextAsync();
    matches.AddRange(next);
}

return matches.SingleOrDefault();
```

1. Save the **AdventureWorksCosmosContext.cs** file.

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Context** folder:

    ```
    cd .\AdventureWorks.Context\
    ```

1. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

    ```
    dotnet build

```
using AdventureWorks.Models;
using Microsoft.Azure.Cosmos;
using Microsoft.Azure.Cosmos.Linq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace AdventureWorks.Context
    {
        public class AdventureWorksCosmosContext:IAdventureWorksProductContext
        {
            private readonly Container _container;
            public AdventureWorksCosmosContext(string connectionString, string database = "Retail", string container = "Online")
            {
            _container = new CosmosClient(connectionString).GetDatabase(database).GetContainer(container);
            }

        public async Task<Model> FindModelAsync(Guid id)
    {
        var iterator = _container.GetItemLinqQueryable<Model>().Where(m => m.id == id).ToFeedIterator<Model>();

    List<Model> matches = new List<Model>();
    while (iterator.HasMoreResults)
    {
        var next = await iterator.ReadNextAsync();
        matches.AddRange(next);
    }

    return matches.SingleOrDefault();

    }
    public async Task<List<Model>> GetModelsAsync()
```
```

```
    {
        string query = $@"SELECT * FROM items";

    var iterator = _container.GetItemQueryIterator<Model>(query);

    List<Model> matches = new List<Model>();
    while (iterator.HasMoreResults)
    {
        var next = await iterator.ReadNextAsync();
        matches.AddRange(next);
    }

    return matches;

    }
    public async Task<Product> FindProductAsync(Guid id)
    {
        string query = $@"SELECT VALUE products
                        FROM models
                        JOIN products in models.Products
                        WHERE products.id = '{id}'";

    var iterator = _container.GetItemQueryIterator<Product>(query);

    List<Product> matches = new List<Product>();
    while (iterator.HasMoreResults)
    {
        var next = await iterator.ReadNextAsync();
        matches.AddRange(next);
    }

    return matches.SingleOrDefault();

    }

    }

    }
```

```

> **Note**: If there are any build errors, review the **AdventureWorksCosmosContext.cs** file in the **Allfiles (F):\\Allfiles\\Labs\\04\\Solution\\AdventureWorks\\AdventureWorks.Context** folder.

1.  Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Task 3: Update the Azure Cosmos DB connection string
```

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.

1. Open the **appsettings.json** file.

1. In the JSON object on line 4, find the **ConnectionStrings.AdventureWorksCosmosContext** path. Observe that the current value is empty:

    ```
    "ConnectionStrings": {
        ...
        "AdventureWorksCosmosContext": "",
        ...
    },
    ```

1. Update the value of the **AdventureWorksCosmosContext** property by setting its value to the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.

1. Save the **appsettings.json** file.

#### Task 4: Update .NET application startup logic

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.

1. Open the **Startup.cs** file.

1. In the **Startup** class, find the existing **ConfigureProductService** method:

    ```
    public void ConfigureProductService(IServiceCollection services)
    {
        services.AddScoped<IAdventureWorksProductContext, AdventureWorksSqlContext>(provider =>
            new AdventureWorksSqlContext(
                _configuration.GetConnectionString(nameof(AdventureWorksSqlContext))
            )
        );
    }
    ```

    > **Note**: The current product service uses SQL as its database.

1. Within the **ConfigureProductService** method, delete all existing lines of code:

    ```
    public void ConfigureProductService(IServiceCollection services)
    {
    }
    ```

1. Within the **ConfigureProductService** method, add the following block of code to change the products provider to the **AdventureWorksCosmosContext** implementation that you created earlier in this lab:

    ```
    services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosContext>(provider =>
        new AdventureWorksCosmosContext(
            _configuration.GetConnectionString(nameof(AdventureWorksCosmosContext))
    ```

~~~
   )
 );
```
~~~

```
using AdventureWorks.Context;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace AdventureWorks.Web
{
    public class Startup
    {
        private IConfiguration _configuration { get; }

        public Startup(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();

            services.AddRouting(options => options.LowercaseUrls = true);

            services.Configure<Settings>(
                _configuration.GetSection(nameof(Settings))
            );

            ConfigureProductService(services);
        }

        public void ConfigureProductService(IServiceCollection services)
        {

           services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosC
ontext>(provider => new AdventureWorksCosmosContext(
            _configuration.GetConnectionString(nameof(AdventureWorksCosmosContext)
)
        )
    );


        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
        {
            app.UseDeveloperExceptionPage();
            app.UseStaticFiles();
            app.UseRouting();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapRazorPages();
            });
        }
    }
}
```

1. Save the **Startup.cs** file.

#### Task 5: Validate that the .NET application successfully connects to Azure Cosmos DB

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

1. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Web** folder:

    ```
    cd .\AdventureWorks.Web\
    ```

1. At the command prompt, enter the following command, and then select Enter to run the ASP.NET web application:

    ```
    dotnet run
    ```

    > **Note**: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

1. On the taskbar, select the **Microsoft Edge** icon.

1. In the open browser window, browse to the currently running web application (<http://localhost:5000>).

1. In the web application, observe the list of models displayed from the front page.

1. Find the **Touring-1000** model, and then select **View Details**.

1. From the **Touring-1000** product detail page, perform the following actions:

    1. In the **Select options** list, select **Touring-1000 Yellow, 50, $2,384.07**.

    1. Find **Add to Cart**, and then observe that the checkout functionality is still disabled.

1. Close the browser window displaying your web application.

1. Return to the **Visual Studio Code** window, and then select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

#### Review

In this exercise, you wrote C# code to query an Azure Cosmos DB collection by using the .NET SDK.

Exercise 6: Clean up your subscription

#### Task 1: Open Azure Cloud Shell

1. In the portal, select the **Cloud Shell** icon to open a new shell instance.

    > **Note**: The **Cloud Shell** icon is represented by a greater than sign (\>) and underscore character (_).

1. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

    1. A dialog box prompts you to create a new storage account to begin using the shell. Accept the default settings, and then select **Create storage**.

    > **Note**: Wait for the Cloud Shell to finish its initial setup procedures before moving forward with the lab.If you don't notice the Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

#### Task 2: Delete resource groups

1. At the command prompt, enter the following command, and then select Enter to delete the **PolyglotData** resource group:

    ```
    az group delete --name PolyglotData --no-wait --yes
    ```

1. Close the Cloud Shell pane in the portal.

#### Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.

1. Close the currently running Visual Studio Code application.

#### Review

In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.