AZ-204

# Developing solutions for Microsoft Azure

## Lab 04- Part 3

## Constructing a polyglot data solution

# Table of Contents

# 1 Pre-requisites

## 1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

**Note**: Instructions to connect to the virtual lab environment will be provided by your instructor.

## 1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Azure CLI
- Windows PowerShell
- **Complete Lab-04-Part 2 - Prerequisite**

# 2 Exercise 5: Accessing Azure Cosmos DB by using .NET

## 2.1 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**.
2. 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\
```

3. 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.

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

```
dotnet build
```

## 2.2 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.
2. Access the shortcut menu or right-click or activate the shortcut menu for the **AdventureWorks.Context** folder node, and then select **New File**.
3. At the new file prompt, enter **AdventureWorksCosmosContext.cs**.
4. 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;
```

5. 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;
```

6. 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;
```

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

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

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

```
public class AdventureWorksCosmosContext       {



    }
```

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

```
public class AdventureWorksCosmosContext : IAdventureWorksProductContext
    {



    }
```

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

```
private readonly Container _container;
```

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

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

12. 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);
```

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

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

     }
```

14. 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();
```

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

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

        }
```

16. 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:

```
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;

        }
```

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

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

18. 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:

```
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();

    }
```

19. Save the **AdventureWorksCosmosContext.cs** file.
20. 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\
```

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

```
dotnet build
```

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

## 2.3 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.
2. Open the **appsettings.json** file.
3. In the JSON object on line 4, find the **ConnectionStrings.AdventureWorksCosmosContext** path. Observe that the current value is empty:

```
{
  "ConnectionStrings": {
    "AdventureWorksSqlContext": "jjjjjdfdff",
    "AdventureWorksCosmosContext": ""
  },
```

3. 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.
4. Save the **appsettings.json** file.

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

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.
2. Open the **Startup.cs** file.
3. 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.

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

5. 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:

```
public void ConfigureProductService(IServiceCollection services)

        {
            services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosContext>(
provider =>
                new AdventureWorksCosmosContext(
                        _configuration.GetConnectionString(nameof(AdventureWorksCosmosContex
t))
            )
            );
        }
```

5. Save the **Startup.cs** file.

## 2.5 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**.
2. 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\
```

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

```
dotnet run
```

3. **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.
4. On the taskbar, select the **Microsoft Edge** icon.
5. In the open browser window, browse to the currently running web application (http://localhost:5000).
6. In the web application, observe the list of models displayed from the front page.
7. Find the **Touring-1000** model, and then select **View Details**.
8. 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**.
    2. Find **Add to Cart**, and then observe that the checkout functionality is still disabled.
9. Close the browser window displaying your web application.
10. 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.

## 2.6 Review

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

# 3 Exercise 6: Clean up your subscription

## 3.1 Task 1: Open Azure Cloud Shell

1. In the Azure 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 (_).

2. 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 configure the shell. Select **Bash**, review the selected subscription, 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.

## 3.2 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
   ```

2. Close the Cloud Shell pane in the portal.

## 3.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

## 3.4 Review

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