Microsoft

Azure Development

Lab 5
Develop .NET Applications with Databases

DISCLSIMER

The information herein is for informational purposes only and represents the current view of Microsoft Corporation or any Microsoft Group affiliate as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

CONFIDENTIALITY

Microsoft

Contents

Lab 1 – Develop .NET Applications with Databases (90min)

Objective(s)	• To create .NET MVC applications that connects to the backend SQL Database • To use Entity Framework (EF) and easily generate default DbContext • Automatically create controllers and views for managing data • Enable Create, Read, Update and Delete (CRUD) operations with minimal code • Register the DbContext using Dependency Injection • Run the app locally • Deploy the app to Azure
Duration of Lab	• Build .NET NVC App with Azure SQL DB and EF (60min) • Add Enhancements (15min) • Deploy your App to Azure (15min)
Tool(s)	• Azure Portal • Visual Studio 2022 • Azure SQL Database with *AdventureWorks* sample deployed • .NET 8 SDK installed
Exercises	1.
Subscription	[Selected Subscription]
Resource Group	[Selected RG]
Navigation	Throughout this Lab, we will open and use several Browser tabs for easy access. Until the end of the Lab, keep your Browser tabs open.
References	• Introduction to Azure for developers | Microsoft Learn

Naming Convention for Labs

For completing various labs during the workshops, we will use this naming convention. It is slightly different from Microsoft online guidance (Define your naming convention - Cloud Adoption Framework | Microsoft Learn).

The naming convention below is designed to group your Azure resources together for easy access.

[you name/initials]-[short name for Azure service]-[service description]

Exercise 1- Build .NET NVC App with Azure SQL DB and EF (60m +)

Topics	In this exercise, we will cover the following topics. • Create new .NET MVC App • Add Entity Framework Package • Create a Controller and Views • Run and the Test App • (Optional) Scaffold Another Table
Duration	• 60 min + Optional
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Create a New .NET 8 MVC App

1. Open VS2022
2. Click **Create a new project**
3. Select **ASP.NET Core Web App (Model-View-Controller)**
4. Click **Next**, name your project (e.g., AdventureWorksMvcApp)
5. Choose **.NET 8 (LTS)** as the framework
6. Click **Create**

Module 2 – Add Entity Framework (EF) Core Packages

1. Open the **Package Manager Console** (Tools > NuGet Package Manager > Package Manager Console) and run

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
```

2. Still in *Package Manager Console*, use the **Scaffold-DbContext** command to generate models and a DbContext from your AdventureWorks database:

```
Scaffold-DbContext "Your_Connection_String" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -
Tables Person, Product, SalesOrderHeader -Context AdventureWorksContext -f
```

Replace **"Your_Connection_String"** with your actual Azure SQL DB connection string (your *SQL DB -> Overview -> Show database connection strings -> ADO.NET (SWL authentication)*)

3. Register the DbContext in the App. Open Program.cs and add the following inside the builder configuration block:

```
builder.Services.AddDbContext<AdventureWorksContext>(options =>
  options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
```

```
using AdventureWorksMvsApp.Models;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<AdventureWorksContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

var app = builder.Build();

// Configure the HTTP request pipeline.
```

4. Add your connection string under **"ConnectionStrings"** in *appsettings.json*:

```
"ConnectionStrings": {
  "DefaultConnection": "Server=tcp:<your-server>.database.windows.net,1433;Initial Catalog=<your-db>;Persist Security Info=False;User ID=<your-user>;Password=<your-password>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
 },
```

Make to replace <your-server>, <your-user> and <your-password> above

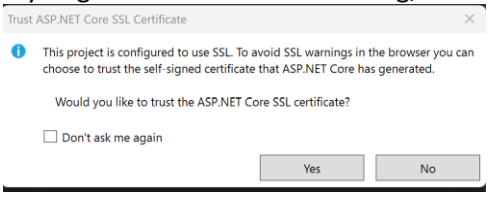
```
{
    "ConnectionStrings": {
        "DefaultConnection": "Server=tcp:<your-server>.databa
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*"
}
```

Module 3 –Create a Controller and Views

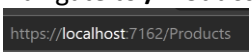
1. Right-click the **Controllers** folder **> Add > Controller**
2. Choose **MVC Controller with views, using Entity Framework**
3. Select:
    a. **Model class:** Product
    b. **Data context class:** AdventureWorksContext
4. Click **Add**
5. This generates:
    a. A controller (e.g., *ProductsController.cs*)
    b. Views for listing, creating, editing, and deleting products

Module 4 –Run and Test the App

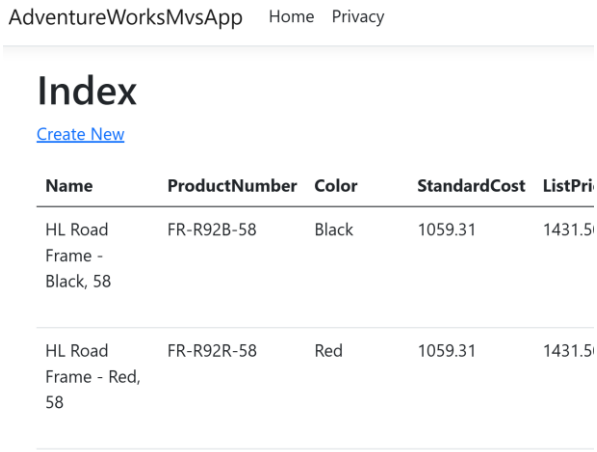
1. Press **F5** to run the app
2. If you get a SSL certificate warning, click Yes



3. Navigate to **/Products** in the address bar at the top



4. You should see a list of products from the AdventureWorkds database



5. Close web browser
6. You have successfully built .NET MVC application connecting to your Azure SQL Database!

Module 5 (Optional) – Scaffold Another Table

1. Scaffold another tale, ***SalesOrderHeader***, like you did for the previous module
2. Run and Test the app

Summary

ACHIEVEMENTS

After you have completed the exercise, you are now able to:

- ✓ Create a structured, maintainable web app using the Model-View-Controller pattern
- ✓ Used Entity Framework Core to scaffold models and interact with data
- ✓ Connect to Aure SQL Database from your web app
- ✓ Automatically create controllers and views for managing data
- ✓ Enable Create, Read, Update and Delete (CRUD) operations with minimal code
- ✓ Register the DbContext using **Dependency Injection**
- ✓ Verify the app works end-to-end by running it locally

Exercise 2- Add Enhancements (15min)

Scenario	You have developed a MVC application with the basic scaffolding structure and CRUD operations to manipulate the data in SQL Database. You will build an enhancement functionality to allow users to **search products by name** in the *Products list* view.
Topics	In this exercise, we will cover the following topics. • Update the Controller
Duration	• 15 min
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Update the controller

1. Open **ProductsController.cs** and update the *Index action from this (below) to the following code*:

Old Index():

```
// GET: Products
3 references
public async Task<IActionResult> Index()
{
    return View(await _context.Products.ToListAsync());
}
```

Update the method to:

```
public async Task<IActionResult> Index(string searchString)
{
   var products = from p in _context.Products
              select p;

   if (!string.IsNullOrEmpty(searchString))
   {
      products = products.Where(p => p.Name.Contains(searchString));
   }

   return View(await products.ToListAsync());
}
```

```
// GET: Products
3 references
public async Task<IActionResult> Index(string searchString)
{
    var products = from p in _context.Products
                   select p;

    if (!string.IsNullOrEmpty(searchString))
    {
        products = products.Where(p => p.Name.Contains(searchString));
    }

    return View(await products.ToListAsync());
}
```

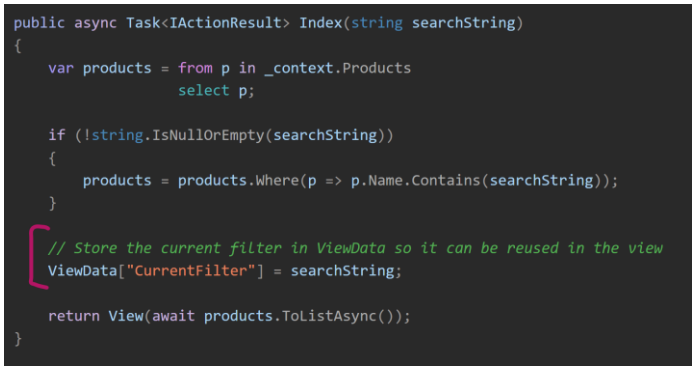
2. Open **Views/Products/Index.cshtml** and add a search form above the table:

```
<form asp-controller="Products" asp-action="Index" method="get">
```

```
    <p>
        Search by name: <input type="text" name="searchString" value="@ViewData["CurrentFilter"]" />
        <input type="submit" value="Search" />
    </p>
</form>
```

3. Add the following line inside the *Index action* of your *ProductsController.cs*, just before returning the view

```
ViewData["CurrentFilter"] = searchString;
```



4. This ensures that when the user submits a search, the input box retains the search term after the page reloads
5. Press F5 and run the app
6. Navigate to */Products* page and test your search functionality
7. You now have a working search box that filters products by name!

Summary

ACHIEVEMENTS

After you have completed the exercise, you are now able to:

- ✓ Add search functionality to filter the product by name
- ✓ Update relevant Controller and View in code

Exercise 3- Deploy your App to Azure (15min)

Scenario	You have a working instance of the MVC app locally. It's time to deploy the app to Azure!
Topics	In this exercise, we will cover the following topics. • Publish Your App in Azure • (Optionally) Secure your App Settings
Duration	• 15 min + Optional
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Publish Your App in Azure

1. In Vs, *publish* the app like you did in the previous lab (right-lick the project -> Publish, Windwos)
2. Create a new App Service for your app
3. In the Publish profile, go to **Settings > Connection Strings**
4. Add a new connection string:
    a. **Name**: DefaultConnection
    b. **Value:** Your Azure SQL DB connection string
    c. **Type:** SQL Server
5. Make sure the connection string matches the one in *appsettings.json*
6. Click **Publish**
7. After deployment, your browser will open the lie site
8. Navigate to /Products and test the app and search functionality

Module 2 (Optional) – Secure Connection Strings

1. Remove the hardcoded Connection Strings for SQL DB connection from App Settings in your App Service in Azure, like you did in the previous exercise
2. Store the SQL DB connection string in Key Vault
3. Update the App Setting value to use Key Vault Reference, instead