Build an ASP.NET Core Service, and App with Core 2.2 Two-Day Hand-On Lab

Lab 6

This lab is the first in a series that builds the SpyStore RESTful service. Prior to starting this lab, you must have completed Lab 5.

Part 1: Configure the Application

Step 1: Add the connection string to the development settings

1) Update the `appsettings.Development.json` in the `SpyStore.Hol.Service` project to the following (adjusted for your machine's setup):

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "ConnectionStrings": {
    //SQL Server 2016 LocalDb
    //"SpyStore":
"Server=(localdb)\\MSSQLLocalDB;Database=SpyStore21;Trusted_Connection=True;MultipleActiveResultSe
ts=true"
    //Docker-Compose
    //"SpyStore": "Server=db;Database=SpyStore21;User
Id=sa;Password=P@ssw0rd;MultipleActiveResultSets=true"
    //Docker
    "SpyStore": "Server=.,6433;Database=SpyStore21;User
ID=sa;Password=P@ssw0rd;MultipleActiveResultSets=true;"
  }
}
```

2) [SQL 2017 LocalDb] Update the `appsettings.Development.json` to the following (adjusted for your machine's setup). **NOTE:** It's best to **not** use SQL Server 2017 LocalDb, but use SQL Server 2017 Developer Edition:

```
  "ConnectionStrings": {
    "SpyStore": "Data
Source=(localdb)\\mssqllocaldb;Database=SpyStoreHOL;AttachDbFileName={path}\\SpyStoreHOL.mdf;Trust
ed_Connection=True;MultipleActiveResultSets=true",
  }
}
```

Step 2: Add the Production Settings File

1) Add a new JSON file to the `SpyStore.Hol.Service` project named `appsettings.Production.json`. Update the file to the following (this will cause the app to fail in production since the connection string is invalid):

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "System": "Warning",
      "Microsoft": "Warning"
    }
  },
  "ConnectionStrings": {
    "SpyStore": "N/A"
  }
}
```

Part 2: Update the Startup.cs class

Step 1: Update the using statements

1) Add the following using statements to the top of the Statup.cs class:

```
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;
using SpyStore.Hol.Dal.EfStructures;
using SpyStore.Hol.Dal.Initialization;
using SpyStore.Hol.Dal.Repos;
using SpyStore.Hol.Dal.Repos.Interfaces;
```

Step 2: Add a Class Level Variable for the Environment

1) Open the `Startup.cs` file and navigate to the constructor. The constructor by default takes in an instance of `IConfiguration`, but it can also take in `IHostingEnvironment` and `ILoggerFactory` instances. Update the constructor to take an instance of `IHosingEnvironment`, and assign that injected instance to a class level variable.

```
private readonly IHostingEnvironment _env;
public Startup(IConfiguration configuration, IHostingEnvironment env)
{
  Configuration = configuration;
  _env = env;
}
```

Step 3: Add the Required Services to the Dependency Injection Container

1) Open the `Startup.cs` file and navigate to the `ConfigureServices` method
2) Use the `IConfiguration` instance to get the connection string:

```
var connectionString = Configuration.GetConnectionString("SpyStore");
```

a) [SQL 2017 LocalDb] Use the `IConfiguration` instance to get the connection string and modify based on the APPDATA user directory (only if the environment is `Development`):

```
var connectionString = Configuration.GetConnectionString("SpyStore");
if (_env.IsDevelopment())
{
  var path = Environment.GetEnvironmentVariable("APPDATA");
  connectionString = connectionString.Replace("{path}", path);
}
```

3) EF Core support is added to the ASP.NET Core DI Container using the built-in `AddDbContextPool` method. This method constructs the derived `DbContext` using the constructor that takes an instance of the `StoreContext` class. Add the following code into the `ConfigureService` method:

```
services.AddDbContextPool<StoreContext>(
  options =>options.UseSqlServer(connectionString));
```

4) Next add all of the repos into the DI container by adding these lines into the `ConfigureServices` method:

```
services.AddScoped<ICategoryRepo, CategoryRepo>();
services.AddScoped<IProductRepo, ProductRepo>();
services.AddScoped<ICustomerRepo, CustomerRepo>();
services.AddScoped<IShoppingCartRepo, ShoppingCartRepo>();
services.AddScoped<IOrderRepo, OrderRepo>();
services.AddScoped<IOrderDetailRepo, OrderDetailRepo>();
```

5) Change the JSON formatting to Pascal casing. Add the following code **after** the first call to services.AddMvc:

```
services.AddMvcCore()
  .AddJsonFormatters(j =>
  {
    j.ContractResolver = new DefaultContractResolver();
    j.Formatting = Formatting.Indented;
  });
```

Step 4: Configure CORS

This policy lets any application call the methods on the service. NOTE: Production applications need to be more locked down.

1) Add the CORS policy in the `ConfigureService` method. Add the following code to the Configure method:

```
services.AddCors(options =>
{
  options.AddPolicy("AllowAll", builder =>
  {
    builder.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin().AllowCredentials();
  });
});
```

2) Add CORS support to the Application (using the policy created in `ConfigureServices` method) in the `Configure` method. Add the following code to the Configure method **before** the app.UseMvc call:

```
app.UseCors("AllowAll");
```

Step 5: Call the Data Initializer in the Configure method

1) Navigate to the Configure method and update the code block in the `IsDevelopment` if block:

```
if (env.IsDevelopment() || env.IsEnvironment("Local"))
{
  app.UseDeveloperExceptionPage();
  using (var serviceScope = app.ApplicationServices
                               .GetRequiredService<IServiceScopeFactory>().CreateScope())
  {
    SampleDataInitializer.InitializeData(
      serviceScope.ServiceProvider.GetRequiredService<StoreContext>());
  }
}
```

Part 3: Update the Service Project Change Hosting Model

A change in ASP.NET Core 2.2 is defaulting projects to run InProcess. In earlier versions of ASP.NET Core, IIS serves as a reverse proxy. In 2.2, the ASP.NET Core Module can boot the CoreCLR and host an app inside the IIS worker process (*w3wp.exe*). In-process hosting provides performance and diagnostic gains when running with IIS.

Step 1: Change the SpyStore.Service.csproj file

1) Remove the following line from the Project file (accessed in VS 2019 by double clicking the project, or right clicking the project and selecting Edit SpyStore.Service.csproj):

```
<AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
```

Summary

This lab configured the DI container and the HTTP Pipeline.

Next steps

In the next part of this tutorial series, you will add Swagger support to the services project.