Build an EF and ASP.NET Core App HOL

Welcome to the Build an Entity Framework Core and ASP.NET Core Application in a Day Hands On Lab. This lab walks you through configuring the pipeline, setting up configuration, and dependency injection.

Prior to starting this lab, you must have completed Lab 3.

All labs and files are available at https://github.com/skimedic/dotnetcore_hol.

Part 1: Update the WebHostBuilder in Program.cs

1) Examine the port in the launchSettings.json for the SpyStore_HOL.MVC setting block.
2) Open the Program.cs file and update the WebHostBuilder call as follows:

```
var host = new WebHostBuilder()
  .UseKestrel()
  //Use the port from launchSettings.json
  .UseUrls("http://*:55876/")
  .UseContentRoot(Directory.GetCurrentDirectory())
  .UseIISIntegration()
  .UseStartup<Startup>()
  .UseApplicationInsights()
  .Build();
```

Part 2: Update the Startup.cs class

Step 1: Update the using statements

1) Update the using statements to the following:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.EF.Initialization;
using SpyStore_HOL.DAL.Repos;
using SpyStore_HOL.DAL.Repos.Interfaces;
```

Step 2: Configure the Dependency Injection Container

1) Open the Startup.cs file and navigate to the ConfigureServces method
2) Add the following code after the services.MVC() call:

```
services.AddSingleton(_ => Configuration);
services.AddDbContext<StoreContext>(options =>
```

```
    options.UseSqlServer(Configuration.GetConnectionString("SpyStore")));
services.AddScoped<ICategoryRepo, CategoryRepo>();
services.AddScoped<IProductRepo, ProductRepo>();
services.AddScoped<ICustomerRepo, CustomerRepo>();
services.AddScoped<IShoppingCartRepo, ShoppingCartRepo>();
services.AddScoped<IOrderRepo, OrderRepo>();
services.AddScoped<IOrderDetailRepo, OrderDetailRepo>();
```

Step 2: Call the Data Initializer in the Configure method

1) Navigate to the Configure method.
2) Update the code block in the IsDevelopment if block:

```
app.UseDeveloperExceptionPage();
//app.UseBrowserLink();
using (var serviceScope = app.ApplicationServices.GetRequiredService<IServiceScopeFactory>()
  .CreateScope())
{
  StoreDataInitializer.InitializeData(app.ApplicationServices);
}
```

Step 3: Change the error handler

1) Update the UseExceptionHandler method to the following:

```
app.UseExceptionHandler("/Products/Error");
```

Step 4: Change the Default Route

1) Update the default route to the following:

```
routes.MapRoute(
  name: "default",
  template: "{controller=Products}/{action=Index}/{id?}");
```

Part 3: Configure the Application

Step 1: Add the connection string to the development settings

1) Update the appsettings.Development.json to the following:

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "ConnectionStrings": {
```

```
    "SpyStore":
"Server=(localdb)\\mssqllocaldb;Database=SpyStore_HOL;Trusted_Connection=True;MultipleActiveResultSets=
true"
  }
}
```

Step 2: Add the connection string to the production settings

1) Add a new json file named appsettings.Production.json.
2) Update the file to the following:

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "None"
    }
  },
  "ConnectionStrings": {
    "SpyStore": "Production connection string"
  }
}
```

Summary

The lab configured the WebHostBuilder, the Startup class, the HTTP pipeline, and application configuration.

Next steps

In the next part of this tutorial series, you will create a custom validation attribute.