Build an EF and ASP.NET Core 2.2 App HOL

Lab 1

This lab walks you through creating the projects and adding/updating the NuGet packages. Prior to starting this lab, you must have completed Lab 0, Installing the Prerequisites.

Part 1: Creating the Solution and Projects

Visual Studio (all versions) is capable of creating projects and solutions, but it is much more efficient to use the .NET Core command line. When creating projects using the command line, the names of solutions, projects, and directories are case sensitive.

Step 1: Pin the .NET Core SDK Version

When using the command line, .NET Core will use the most current version of the SDK installed. For this reason, the required version of .NET Core must be pinned with a global.json file.

1. Check current version by typing:

```
dotnet --version
```

2. If the version is not the version you need (e.g. 2.2.401), enter the following command to create a new file named `global.json`:

```
rem VS 2017 15.9
rem dotnet new globaljson --sdk-version 2.2.108
rem VS 2019 16.0
rem dotnet new globaljson --sdk-version 2.2.205
rem VS 2019 16.1
rem dotnet new globaljson --sdk-version 2.2.301
rem VS 2019 16.2
dotnet new globaljson --sdk-version 2.2.401
```

3. This creates the global.json file with the following content (2.2.401 for .NET Core 2.2):

```
{
    "sdk": {
        "version":"2.2.401"
    }
}
```

4. Execute dotnet --version again and the version will return as 2.2.401 (2.2.401 for .NET Core 2.2).

Step 2: Create the Solution

The templates that are installed with .NET Core range from very simple to quite complex. Creating the global.json file is an example of a simple template, as is creating a new solution.

All files copyright Phil Japikse (http://www.skimedic.com/blog)

1. To create a new solution file named `SpyStore.Hol`, enter the following command:

```
dotnet new sln -n SpyStore.Hol
```

Step 3: Create the ASP.NET Core project

1) The `mvc` template is extremely configurable. In addition to setting the name, authentication scheme, and directory location, there are many other options. These can be explored by entering:

```
dotnet new mvc -h
```

2) From the same directory as the solution file, enter the following command to create the ASP.NET Core project using the model-view-controller pattern, set the name, turn off authentication, not require https, and set the directory for the project as `SpyStore.Hol.Mvc`:

```
dotnet new mvc -n SpyStore.Hol.Mvc -au none --no-https  -o .\SpyStore.Hol.Mvc
```

3) Add the project to the solution with the following command:

```
dotnet sln SpyStore.Hol.sln add SpyStore.Hol.Mvc
```

Step 4: Create the Models and Data Access Layer projects

The `classlib` template create .NET Core class libraries or .NET Standard class libraries. Either will work. For this lab, use .NET Core class libraries.

1) The template takes a name and an output directory. Create the `SpyStore.Hol.Dal` and `SpyStore.Hol.Models` projects by entering the following commands:

```
dotnet new classlib -n SpyStore.Hol.Dal -o .\SpyStore.Hol.Dal -f netcoreapp2.2
dotnet new classlib -n SpyStore.Hol.Models -o .\SpyStore.Hol.Models -f netcoreapp2.2
```

2) Add the projects to the solution with the following commands:

```
dotnet sln SpyStore.Hol.sln add SpyStore.Hol.Dal
dotnet sln SpyStore.Hol.sln add SpyStore.Hol.Models
```

Step 5: Create the Unit Test project [Optional]

The xUnit template creates a new .NET Core class library with all of the required packages for unit testing. This include xUnit and the `Microsoft.NET.Test.Sdk`.

1) Create the project with the name of `SpyStore.Hol.Dal.Tests` and the directory `SpyStore.Hol.Dal.Tests` using the following command:

```
dotnet new xunit -n SpyStore.Hol.Dal.Tests -o .\SpyStore.Hol.Dal.Tests
```

2) Add the project to the solution with the following command:

```
dotnet sln SpyStore.Hol.sln add SpyStore.Hol.Dal.Tests
```

Part 2: Add the Project References

When adding project references through the command line, the commands are not case sensitive.

Step 1: Update the SpyStore.Hol.Mvc Project

The `SpyStore.Hol.Mvc` project references the `SpyStore.Hol.Dal` and the `SpyStore.Hol.Models` projects.

1) Enter the following commands to add the references:

```
dotnet add SpyStore.Hol.Mvc reference SpyStore.Hol.Models
dotnet add SpyStore.Hol.Mvc reference SpyStore.Hol.Dal
```

Step 2: Update the SpyStore.Hol.Dal Project

The `SpyStore.Hol.Dal` project references the `SpyStore.Hol.Models` project.

1) Enter the following command to add the reference:

```
dotnet add SpyStore.Hol.Dal reference SpyStore.Hol.Models
```

Step 3: Update the SpyStore.Hol.Dal.Tests Project

The `SpyStore.Hol.Dal.Tests` project references the `SpyStore.Hol.Dal` and the `SpyStore.Hol.Models` projects.

2) Enter the following commands to add the references:

```
dotnet add SpyStore.Hol.Dal.tests reference SpyStore.Hol.Models
dotnet add SpyStore.Hol.Dal.tests reference SpyStore.Hol.Dal
```

Part 3: Add the NuGet packages

.NET Core, ASP.NET Core, and EF Core are composed of a series of NuGet packages. Additional tools are also distributed as NuGet packages. The command line can also be used for adding NuGet packages.

Step 1: Update the SpyStore.Hol.Mvc Project

The `SpyStore.Hol.Mvc` project needs `Automapper`, `Newtonsoft.json`, and `Ligershark.WebOptimizer.Core` added.

1) Enter the following commands to add the packages:

```
dotnet add SpyStore.Hol.Mvc package AutoMapper
dotnet add SpyStore.Hol.Mvc package Newtonsoft.Json
dotnet add SpyStore.Hol.Mvc package LigerShark.WebOptimizer.Core -v 1.0.236
dotnet add SpyStore.Hol.Mvc package LigerShark.WebOptimizer.sass -v 1.0.34-beta
```

Step 2: Update the SpyStore.Hol.Models Project

The `SpyStore.Hol.Models` project needs `AutoMapper` and `Microsoft.EntityFrameworkCore.Abstractions`. The version of EF Core added into the support projects should match the version that is included with `Microsoft.AspNetCore.App` (unless you update all projects to the same version)

1) Add the packages with the following commands:

```
REM .NET Core 2.2
dotnet add SpyStore.Hol.Models package Microsoft.EntityFrameworkCore.Abstractions -v 2.2.0
dotnet add SpyStore.Hol.Models package AutoMapper
dotnet add SpyStore.Hol.Models package Newtonsoft.Json
```

Step 3: Update the SpyStore.Hol.Dal Project

The `SpyStore.Hol.Dal` project needs `NewtonSoft.Json`, `Microsoft.EntityFrameworkCore.SqlServer` (SQL Server provider for EF Core), and `Microsoft.EntityFrameworkCore.Design` (command line tooling). In order to use the EF Core tooling from Package Manager Console, the project also needs `Microsoft.EntityFramewordCore.Tools`.

1) Add the packages with the following commands:

```
REM .NET Core 2.2
dotnet add SpyStore.Hol.Dal package Microsoft.EntityFrameworkCore.SqlServer -v 2.2.0
dotnet add SpyStore.Hol.Dal package Microsoft.EntityFrameworkCore.Design -v 2.2.0
dotnet add SpyStore.Hol.Dal package Microsoft.EntityFrameworkCore.Tools -v 2.2.0
dotnet add SpyStore.Hol.Dal package Newtonsoft.Json
```

Step 4: Update the Unit Test Project

The `SpyStore.Hol.Dal.Tests` project needs the `Microsoft.EntityFrameworkCore.SqlServer` package.

1) Add the package with the following command:

```
dotnet add SpyStore.Hol.Dal.Tests package Microsoft.EntityFrameworkCore.SqlServer -v 2.2.0
```

Summary

This lab created all of the projects for the HOL, added the NuGet packages, and the appropriate references.

Next steps

In the next part of this tutorial series, you will start to build the data access library using Entity Framework Core.