Build an ASP.NET Core MVC App with EF Core One-Day Hands-On Lab

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.

Create a new directory on your computer and use that as the starting point for all of the commands.

Part 1: Global JSON and NuGet Config files

Step 1: Use a Global JSON file to Pin the .NET Core SDK Version

.NET Core commands use the latest version of the SDK installed unless a version is specified in a global.json file at or above the current working directory.

- Check current version by typing:

```
dotnet --version
```

- Enter the following command to create a new file named `global.json` pinning the SDK version to 5.0.300:

```
dotnet new globaljson --sdk-version 5.0.300
```

- This creates the global.json file with the following content:

```
{
    "sdk": {
        "version":"<version from the chart above>"
    }
}
```

- If you had .NET 6 Preview installed, the global.json file will keep you on the current version.

Step 2: Create a NuGet Config

To prevent corporate or other package sources from interfering with this lab, create a NuGet.config file that clears out any machine sources and adds in the standard NuGet feed. This file only applies to the contained directory structure.

- To create a the file, enter the following command:

```
dotnet new nugetconfig --nuget-source https://api.nuget.org/v3/index.json
```

Part 2: Creating the Solution and Projects

Visual Studio (all versions) can create and manage projects and solutions, but it is much more efficient to use the .NET 5 command line. When creating projects using the command line, the names of solutions, projects, and directories are case sensitive.

Step 1: Create the Solution

The templates that are installed with the .NET 5 SDK range from simple to complex. Creating the global.json and NuGet.config files are examples of simple templates, as is creating a new solution.

- To create a new solution file named `AutoLot`, enter the following command:

```
dotnet new sln -n AutoLot
```

Step 2: Create the Class Library projects

The `classlib` template is used to create .NET Core class libraries.

- Add the three class library projects, using C# (-lang c#) and .NET 5.0 (-f net5.0):

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Models -o .\AutoLot.Models -f net5.0
dotnet new classlib -lang c# -n AutoLot.Dal -o .\AutoLot.Dal -f net5.0
dotnet new classlib -lang c# -n AutoLot.Services -o .\AutoLot.Services -f net5.0
[Non-Windows]
dotnet new classlib -lang c# -n AutoLot.Models -o ./AutoLot.Models -f net5.0
dotnet new classlib -lang c# -n AutoLot.Dal -o ./AutoLot.Dal -f net5.0
dotnet new classlib -lang c# -n AutoLot.Services -o ./AutoLot.Services -f net5.0
```

- Add the projects to the solution with the following commands:

```
dotnet sln AutoLot.sln add AutoLot.Models
dotnet sln AutoLot.sln add AutoLot.Dal
dotnet sln AutoLot.sln add AutoLot.Services
```

Step 3: Create the ASP.NET Core Web Application (MVC) project

- The `mvc` template is extremely configurable. Options can be explored by using -h (help):

```
dotnet new mvc -h
```

- From the same directory as the solution file, enter the following command to create the ASP.NET Core web application project using the model-view-controller pattern:

```
[Windows]
dotnet new mvc -lang c# -n AutoLot.Mvc -au none -o .\AutoLot.Mvc -f net5.0
[Non-Windows]
dotnet new mvc -lang c# -n AutoLot.Mvc -au none -o ./AutoLot.Mvc -f net5.0
```

- Add the project to the solution with the following command:

```
dotnet sln AutoLot.sln add AutoLot.Mvc
```


Step 4: Create the Unit Test project [Optional]

The xUnit template creates a new .NET Core class library with the packages needed for creating tests with the xUnit testing framework. This include xUnit and the `Microsoft.NET.Test.Sdk`.

- Create the project with the names of `AutoLot.Dal.Tests` and `AutoLot.Service.Tests` in their respective directories using the following command:

```
[Windows]
dotnet new xunit -lang c# -n AutoLot.Dal.Tests -o .\AutoLot.Dal.Tests -f net5.0
[Non-Windows]
dotnet new xunit -lang c# -n AutoLot.Dal.Tests -o ./AutoLot.Dal.Tests -f net5.0
```

- Add the project to the solution with the following command:

```
dotnet sln AutoLot.sln add AutoLot.Dal.Tests
```

Part 3: Add the Project References

When adding project references through the command line, the commands are case sensitive on non-Windows operating systems.

Step 1: Update the AutoLot.Dal Project

```
dotnet add AutoLot.Dal reference AutoLot.Models
```

Step 2: Update the AutoLot.Services Project

```
dotnet add AutoLot.Services reference AutoLot.Models
```

Step 3: Update the AutoLot.Mvc Project

```
dotnet add AutoLot.Web reference AutoLot.Models
dotnet add AutoLot.Web reference AutoLot.Dal
dotnet add AutoLot.Web reference AutoLot.Services
```

Step 4: Update the AutoLot.Dal.Tests

```
dotnet add AutoLot.Dal.Tests reference AutoLot.Models
dotnet add AutoLot.Dal.Tests reference AutoLot.Dal
```

Part 4: Add the NuGet packages to the Projects

Step 1: Update the AutoLot.Models Project

```
dotnet add AutoLot.Models package Microsoft.EntityFrameworkCore.Abstractions
dotnet add AutoLot.Models package AutoMapper
dotnet add AutoLot.Models package System.Text.Json
```


Step 2: Update the AutoLot.Dal Project

```
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.Design
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.Tools
```

Step 3: Update the AutoLot.Services Project

```
dotnet add AutoLot.Services package Microsoft.Extensions.Hosting.Abstractions
dotnet add AutoLot.Services package Microsoft.Extensions.Options
dotnet add AutoLot.Services package Serilog.AspNetCore
dotnet add AutoLot.Services package Serilog.Enrichers.Environment
dotnet add AutoLot.Services package Serilog.Settings.Configuration
dotnet add AutoLot.Services package Serilog.Sinks.Console
dotnet add AutoLot.Services package Serilog.Sinks.File
dotnet add AutoLot.Services package Serilog.Sinks.MSSqlServer
dotnet add AutoLot.Services package System.Text.Json
```

Step 5: Update the AutoLot.Mvc Project

```
dotnet add AutoLot.Mvc package AutoMapper
dotnet add AutoLot.Mvc package System.Text.Json
dotnet add AutoLot.Mvc package LigerShark.WebOptimizer.Core
dotnet add AutoLot.Mvc package Microsoft.Web.LibraryManager.Build
dotnet add AutoLot.Mvc package Microsoft.EntityFrameworkCore
dotnet add AutoLot.Mvc package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Mvc package Microsoft.VisualStudio.Web.CodeGeneration.Design
```

Step 6: Update the AutoLot.Data.Tests Projects

```
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Dal.Tests package Microsoft.Extensions.Configuration.Json
dotnet remove AutoLot.Dal.Tests package Microsoft.NET.Test.Sdk
dotnet add AutoLot.Dal.Tests package Microsoft.NET.Test.Sdk
```

Summary

This lab created the solution and the projects for the hands-on lab, 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.