.NET 9 App Dev Hands-On Lab

EF Lab 1 –Main Solution and EF Core Projects

This lab walks you through creating the projects and adding/updating the NuGet packages. Before starting this lab, you must have completed 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 on your development machine unless a version is specified in a `global.json` file. The file updates the allowable SDK versions for all directories below its location.

- Check the current version by typing:

```
dotnet --version
```

- Enter the following command to create a new file named `global.json`, pinning the SDK version to 8.0.100 (make sure to use the version that you have installed):

```
dotnet new globaljson --sdk-version 9.0.100 --roll-forward feature
```

- This creates the `global.json` file with the following content (with the version from the previous command):

```
{
  "sdk": {
    "rollforward":"feature",
    "version":"9.0.100"
  }
}
```

- Use `--force` to overwrite an existing file:

```
dotnet new globaljson --sdk-version 9.0.100 --roll-forward feature --force
```

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 all machine sources and adds the standard NuGet feed. This file only applies to the contained directory structure.

- To create the file, enter the following command:

```
dotnet new nugetconfig
```

Part 2: Creating the Solution and Projects

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

Step 1: Create the Solution

The templates installed with the .NET 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
```

The following commands are scripted to run in the same directory as the solution that was created. Each project will be created in a subfolder, added to the solution, and the required NuGet packages added.

Step 2: Create the Class Libraries

The Data Access Layer uses two class libraries: `AutoLot.Models` (for the entities) and `AutoLot.Dal` (for the data access layer code).

Step A: Create the AutoLot.Models project, add it to the solution, and add project references and NuGet Packages

The `classlib` template creates .NET Core class libraries using C# (-lang c#) and .NET 8.0 (-f net8.0).

- Create the `AutoLot.Models` class library:
  NOTE: Windows uses a backslash (\), and non-Windows uses a forward slash (/). Adjust to your OS.

```
dotnet new classlib -lang c# -n AutoLot.Models -o .\AutoLot.Models -f net9.0
```

- Add the project to the solution:

```
dotnet sln AutoLot.sln add AutoLot.Models
```

- Add the required NuGet packages to the project:
  NOTE: If using PowerShell, the version intervals must be surrounded by single quotes (as shown here). Remove the single quotes from the commands if using a regular command prompt.

```
dotnet add AutoLot.Models package Microsoft.EntityFrameworkCore.SqlServer -v '[9.0.*,10.0)'
dotnet add AutoLot.Models package System.Text.Json -v '[9.0.*,10.0)'
dotnet add AutoLot.Models package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,18.0)'
```


Step B: Create the AutoLot.Dal project, add it to the solution, add project references, and NuGet Packages

- Create the `AutoLot.Dal` class library:

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Dal -o .\AutoLot.Dal -f net9.0
```

- Add the project to the solution and project references:

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

- Add the required NuGet packages to the project:

```
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.SqlServer -v '[9.0.*,10.0)'
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.Design -v '[9.0.*,10.0)'
dotnet add AutoLot.Dal package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,18.0)'
```

Step C: Create the AutoLot.Services project, add it to the solution, and add project references and NuGet Packages

- Create the `AutoLot.Services` class library:

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Services -o .\AutoLot.Services -f net9.0
```

- Add the project to the solution and project references:

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

- Add the required NuGet packages to the project:

```
dotnet add AutoLot.Services package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,18.0)'
dotnet add AutoLot.Services package Microsoft.Extensions.Hosting.Abstractions  -v '[9.0.*,10.0)'
dotnet add AutoLot.Services package Microsoft.Extensions.Options  -v '[9.0.*,10.0)'
dotnet add AutoLot.Services package Serilog.AspNetCore -v '[9.0.*,10.0)'
dotnet add AutoLot.Services package Serilog.Enrichers.Environment  -v '[3.0.*,4.0)'
dotnet add AutoLot.Services package Serilog.Settings.Configuration  -v '[9.0.*,10.0)'
dotnet add AutoLot.Services package Serilog.Sinks.Console -v '[6.0.*,7.0)'
dotnet add AutoLot.Services package Serilog.Sinks.File -v '[6.0.*,7.0)'
dotnet add AutoLot.Services package Serilog.Sinks.MSSqlServer -v '[8.*,9.0)'
dotnet add AutoLot.Services package System.Text.Json -v '[9.0.*,10.0)'
```


Step 3: Create the AutoLot.Dal.Tests project, add it to the solution, and add project references and NuGet Packages

- Create the `AutoLot.Dal.Tests` xUnit project:

```
[Windows]
dotnet new xunit -lang c# -n AutoLot.Dal.Tests -o .\AutoLot.Dal.Tests -f net9.0
```

- Add the project to the solution and project references:

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

- Add the required NuGet packages to the project:

```
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore.SqlServer -v '[9.0.*,10.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore.Design -v '[9.0.*,10.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.Extensions.Configuration.Json -v '[9.0.*,10.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.NET.Test.Sdk  -v '[17.*,18.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,18.0)'
dotnet add AutoLot.Dal.Tests package xunit -v '[2.*,3.0)'
dotnet add AutoLot.Dal.Tests package xunit.runner.visualstudio -v '[2.*,3.0)'
dotnet add AutoLot.Dal.Tests package coverlet.collector -v '[6.0.*,7.0)'
```

Part 3: Disable Nullable Reference Types and Enable Global Implicit Using Statements

In .NET 6+, the templates automatically enable nullable reference types. We won't be using that feature in this hands-on lab, so open the project files (*.`csproj`) for all projects and update the `PropertyGroup` to the following (change is in bold):

```
<PropertyGroup>
  <TargetFramework>net9.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>disable</Nullable>
</PropertyGroup>
```

Summary

This lab created the solution and the projects for the hands-on lab and added the NuGet packages and the appropriate references.

Next steps

In the next part of this tutorial series, you will create the `DbContext` and `DesignTimeDbContextFactory` and run your first migration.