.NET 10 App Dev Hands-On Lab

EF Lab 1 –Create the Solution and Initial Projects

This lab guides you through creating the solution, initial projects, and adding or updating NuGet packages. Before starting this lab, you must have installed the prerequisites.

Create a new directory on your computer and use that as the starting point for all commands.
NOTE: Do NOT use the `CompletedLabs` folder of the GitHub Repository that you cloned.

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 will 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 10.0.100 (make sure to use the version that you have installed):

```
dotnet new globaljson --sdk-version 10.0.100 --roll-forward feature
```

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

```
{
  "sdk": {
    "rollforward":"feature",
    "version":"10.0.100"
  }
}
```

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

```
dotnet new globaljson --sdk-version 10.0.100 --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, solution, project, and directory names are case-sensitive.

Step 1: 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 created solution. Each project will be created in a subfolder, added to the solution, and the required NuGet packages will be added.

Step 2: 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: The AutoLot.Models Project

The `classlib` template creates .NET Core class libraries using C# (-lang c#) and .NET 10.0 (-f net10.0).

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

```
dotnet new classlib -lang c# -n AutoLot.Models -o .\AutoLot.Models -f net10.0
```

- 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 in the example below). Remove the single quotes from the commands if using a regular command prompt.

```
dotnet add AutoLot.Models package Microsoft.EntityFrameworkCore.SqlServer -v '[10.*,11.0)'
dotnet add AutoLot.Models package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,19.0)'
```

Step B: The AutoLot.Dal Project

- Create the `AutoLot.Dal` class library:

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Dal -o .\AutoLot.Dal -f net10.0
```

- 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 '[10.*,11.0)'
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.Design -v '[10.*,11.0)'
dotnet add AutoLot.Dal package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,19.0)'
```

Step C: The AutoLot.Services Project

- Create the `AutoLot.Services` class library:

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Services -o .\AutoLot.Services -f net10.0
```

- 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.AspNetCore.Hosting.Abstractions
dotnet add AutoLot.Services package Microsoft.AspNetCore.Http.Abstractions
dotnet add AutoLot.Services package Microsoft.AspNetCore.Mvc.Abstractions
dotnet add AutoLot.Services package Microsoft.Extensions.DependencyInjection.Abstractions
dotnet add AutoLot.Services package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,19.0)'
dotnet add AutoLot.Services package Serilog -v '[4.*,5.0)'
dotnet add AutoLot.Services package Serilog.AspNetCore -v '[9.*,10.0)'
dotnet add AutoLot.Services package Serilog.Enrichers.Environment  -v '[3.*,4.0)'
dotnet add AutoLot.Services package Serilog.Sinks.Console -v '[6.*,7.0)'
dotnet add AutoLot.Services package Serilog.Sinks.File -v '[7.*,8.0)'
dotnet add AutoLot.Services package Serilog.Sinks.MSSqlServer -v '[9.*,10.0)'
```

Step 3: The AutoLot.Dal.Tests Project

- If you don't have the xunit3 templates, execute the following command:

```
dotnet new install xunit.v3.templates
```

- If you already have them, make sure to get the latest:

```
dotnet new update
```


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

```
[Windows]
dotnet new xunit3 -lang c# -n AutoLot.Dal.Tests -f net10.0 -o .\AutoLot.Dal.Tests
```

- 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 '[10.*,11.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore.Design -v '[10.*,11.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.Extensions.Configuration.Json -v '[10.*,11.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.NET.Test.Sdk -v '[18.*,19.0)'
dotnet add AutoLot.Dal.Tests package Microsoft.VisualStudio.Threading.Analyzers -v '[17.*,19.0)'
dotnet add AutoLot.Dal.Tests package xunit.v3 -v '[3.*,4.0)'
dotnet add AutoLot.Dal.Tests package xunit.runner.visualstudio -v '[3.*,4.0)'
```

Part 3: Disable Nullable Reference Types

In .NET 6+, the templates automatically enable nullable reference types. We won't be using that feature in this hands-on lab.

Manual

Open the project files (*.`csproj`) for all projects and update the Nullable node of the `PropertyGroup` to the following (change is in bold):

```
<Nullable>disable</Nullable>
```

Coplot Agent Mode

```
Prompt: update all projects to set nullable reference types to false
```

Summary

This lab created the solution and projects for the hands-on lab, added the NuGet packages, and included 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.

