# .NET App Dev Hands-On Workshop

## Lab 1 – Solution and Projects

This lab walks you through creating the projects and adding/updating the NuGet packages. Prior to 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 must be located 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 7.0.100 (make sure to use the version that you have installed):
  **NOTE:** If using .NET 6, replace 7.0.100 with 6.0.100

```
dotnet new globaljson --sdk-version 7.0.100 --roll-forward feature
```

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

```
{
  "sdk": {
    "rollforward":"feature",
    "version":"7.0.100"
  }
}
```

## 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 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 it is much more efficient to use the .NET command line interface (CLI). 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 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
```

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

## Step 2: Create the Class Libraries

There are three class libraries in the solution, AutoLot.Models (for the entities), AutoLot.Dal (for the data access layer code), and AutoLot.Services (to hold common services).

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

The `classlib` template is used to create .NET Core class libraries using C# (-lang c#) and .NET 7.0 (-f net7.0).

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

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Models -o .\AutoLot.Models -f net7.0
```

- Add the project to the solution:

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

- Add the required NuGet packages to the project:

```
dotnet add AutoLot.Models package Microsoft.EntityFrameworkCore
dotnet add AutoLot.Models package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Models package Microsoft.VisualStudio.Threading.Analyzers
dotnet add AutoLot.Models package System.Text.Json
```

 

### Step B: Create the AutoLot.Dal project, add it to the solution, and 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 net7.0
```

- ☐ 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
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.Design
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Dal package Microsoft.EntityFrameworkCore.Tools
dotnet add AutoLot.Dal package Microsoft.VisualStudio.Threading.Analyzers
```

### 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 net7.0
```

- ☐ 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
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 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 net7.0
```

- 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.Extensions.Hosting.Abstractions
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore.Design
dotnet add AutoLot.Dal.Tests package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Dal.Tests package Microsoft.Extensions.Configuration.Json
dotnet add AutoLot.Dal.Tests package Microsoft.NET.Test.Sdk
dotnet add AutoLot.Dal.Tests package Microsoft.VisualStudio.Threading.Analyzers
```

## Step 4: 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
```

- 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 net7.0
```

- Add the project to the solution and project references:

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

- Add the required NuGet packages to the project:

```
dotnet add AutoLot.Mvc package AutoMapper
dotnet add AutoLot.Mvc package Microsoft.VisualStudio.Threading.Analyzers
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.Design
dotnet add AutoLot.Mvc package Microsoft.EntityFrameworkCore.SqlServer
dotnet add AutoLot.Mvc package Microsoft.VisualStudio.Web.CodeGeneration.Design
```

 

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

## Step 1: Disable Nullable Reference Types (AutoLot.Dal, AutoLot.Models, AutoLot.MVC, AutoLot.Services)

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 the all projects and update the `PropertyGroup` to the following (change is in bold):

```
<PropertyGroup>
  <TargetFramework>net7.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>disable</Nullable>
</PropertyGroup>
```

## Step 2: Enable Global Implicit Usings in the AutoLot.Dal.Tests project and Disable Nullable Reference Types

The class library and ASP.NET Core templates all enable global implicit using statements by default. The xUnit test project template does not. To take advantage of this new feature and disable nullable reference types, update the `PropertyGroup` in the AutoLot.Dal.Tests.csproj file to the following (changes is in bold):

```
<PropertyGroup>
  <TargetFramework>net7.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>disable</Nullable>
  <IsPackable>false</IsPackable>
</PropertyGroup>
```

# Part 4: Adjust the launchsettings.json file

## Step 1: Move the https profile to the top in AutoLot.Mvc

If a profile isn't selected, the first profile in the list will be selected by default. Move the `https` profile to the top so it gets selected and not the `http` profile.

# 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 create the `DbContext`, `DesignTimeDbContextFactory`, and run your first migration.