.NET App Dev Hands-On Lab

Blazor Lab 1 – The Blazor Projects

This lab walks you through creating the solution, `global.json`, `nuget.config`, and the Blazor Web Assembly (WASM) project and then adding/updating the NuGet packages.

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 (you will pin it to version 8.0+ in the next step):

```
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 8.0.100 --roll-forward feature
```

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

```
{
  "sdk": {
    "rollforward":"feature",
    "version":"8.0.100"
  }
}
```

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

```
dotnet new globaljson --sdk-version 8.0.100 --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 any 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 created solution. Each project will be created in a subfolder, added to the solution, and the required NuGet packages will be added.

Step 2: Create the Projects

Note: Non-windows users must adjust the directory separator using the following commands.

Note: PowerShell and bash need quotes around the version monikers.

- Create the Class Library for the entities and add it to the solution:
  NOTE: Using PowerShell, the version intervals must be surrounded by single quotes (like '[17.*,18.0)'). Run the commands as shown here if using a regular command prompt.

```
[Windows]
dotnet new classlib -lang c# -n AutoLot.Blazor.Models -o .\AutoLot.Blazor.Models -f net8.0
dotnet sln AutoLot.sln add AutoLot.Blazor.Models
dotnet add AutoLot.Blazor.Models package Microsoft.VisualStudio.Threading.Analyzers -v [17.*,18.0)
```

- Create the Blazor WebAssembly Standalone App project, add it to the solution, and add a reference to the class library:

```
[Windows]
dotnet new blazorwasm -lang c# -au none -n AutoLot.Blazor -o .\AutoLot.Blazor -f net8.0
dotnet sln AutoLot.sln add AutoLot.Blazor
dotnet add AutoLot.Blazor reference AutoLot.Blazor.Models
```

- Add the required NuGet packages to the project (each on only one line):

```
dotnet add AutoLot.Blazor package Microsoft.Extensions.Options.ConfigurationExtensions -v
[8.0.*,9.0)
dotnet add AutoLot.Blazor package Microsoft.AspNetCore.Components.WebAssembly -v [8.0.*,9.0)
dotnet add AutoLot.Blazor package Microsoft.AspNetCore.Components.WebAssembly.DevServer -v
[8.0.*,9.0)
dotnet add AutoLot.Blazor package Microsoft.Extensions.Http -v [8.0.*,9.0)
dotnet add AutoLot.Blazor package Microsoft.VisualStudio.Threading.Analyzers -v [17.*,18.0)
```

Step 3: Disable Nullable Reference Types

- Open the new project files (*.csproj) and update the PropertyGroup to disable nullable reference types:

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

- Open the NavMenu.Razor file and update the string property not to be nullable:

```
private string NavMenuCssClass => collapseNavMenu ? "collapse" : null;
```

Step 4: (VS) Set AutoLot.Blazor as the startup project

Right-click on `AutoLot.Blazor` and select "Set as Startup Project" from the context menu.

Step 5: Adjust the launchsettings.json file

Open the `launchsettings.json` file (in the `Properties` directory of the project) and move the `HTTPS` profile to the top.

Part 3: Clean up Unnecessary Scaffolded Code

- Delete `Pages\Counter.razor` and `Pages\Weather.razor` files.
- Delete the `wwwroot\sample-data` folder and the JSON file it contains.
- Delete the following from `Layout\NavMenu.razor`:

```
<div class="nav-item px-3">
  <NavLink class="nav-link" href="counter">
    <span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span> Counter
  </NavLink>
</div>
<div class="nav-item px-3">
  <NavLink class="nav-link" href="weather">
    <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Weather
  </NavLink>
</div>
```

Summary

This lab created the Blazor Wasm project and the shared class library for the models.

Next steps

In the next part of this tutorial series, you will start building the Blazor application.