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

Lab 8

This lab builds the shared services used by the ASP.NET Core applications. Prior to starting this lab, you must have completed Lab 6 (Lab 7 is an optional lab). The entire lab works in the `AutoLot.Services` project.

Start by renaming the `Class1.cs` file to `GlobalUsings.cs`. Update the code to the following:

```
global using AutoLot.Dal.Repos;
global using AutoLot.Dal.Repos.Interfaces;

global using Microsoft.AspNetCore.Builder;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;

global using Serilog;
global using Serilog.Context;
global using Serilog.Core.Enrichers;
global using Serilog.Events;
global using Serilog.Sinks.MSSqlServer;

global using System.Data;
global using System.Runtime.CompilerServices;
```

Part 1: Add Logging Support

Step 1: Add the Logging Interface

- Add a new folder named `Logging` in the `AutoLot.Services` project. In that folder add a new folder named `Interfaces`, and in that folder, add a new interface file named `IAppLogging.cs`. Update the interface code to the following:

```
namespace AutoLot.Services.Logging.Interfaces;

public interface IAppLogging<T>
{
  void LogAppError(Exception exception,
        string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppError(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppCritical(Exception exception,
        string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppCritical(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppDebug(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppTrace(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppInformation(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
  void LogAppWarning(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0);
}
```

- Add the following into the `GlobalUsings.cs` file:

```
global using AutoLot.Services.Logging;
global using AutoLot.Services.Logging.Interfaces;
```

Step 2: Add the Logging Implementation

- In the `Logging` folder add a class file named `AppLogging.cs`. Make the class public and generic, and implement `IAppLogging`:

```
namespace AutoLot.Services.Logging;

public class AppLogging<T> : IAppLogging<T>
{
  //Implementation goes here
}
```

- Inject into the constructor the framework `ILogger<T>` and create a private variable for it:

```
private readonly ILogger<T> _logger;

public AppLogging(ILogger<T> logger)
{
  _logger = logger;
}
```

- Create two internal methods to push the additional properties into the `SeriLog` context. One works with exception, the other without:

```
internal static void LogWithException(string memberName,string sourceFilePath,
  int sourceLineNumber, Exception ex, string message,
  Action<Exception, string, object[]> logAction)
{
  var list = new List<IDisposable>
  {
    LogContext.PushProperty("MemberName", memberName),
    LogContext.PushProperty("FilePath", sourceFilePath),
    LogContext.PushProperty("LineNumber", sourceLineNumber),
  };
  logAction(ex,message,null);
  foreach (var item in list)
  {
    item.Dispose();
  }
}

internal static void LogWithoutException(string memberName, string sourceFilePath,
    int sourceLineNumber, string message, Action<string, object[]> logAction)
{
  var list = new List<IDisposable>
  {
    LogContext.PushProperty("MemberName", memberName),
    LogContext.PushProperty("FilePath", sourceFilePath),
    LogContext.PushProperty("LineNumber", sourceLineNumber),
  };
  logAction(message, null);
  foreach (var item in list)
  {
    item.Dispose();
  }
}
```

- Implement the logging interface members:

```
public void LogAppError(Exception exception, string message,
  [CallerMemberName] string memberName = "", [CallerFilePath] string sourceFilePath = "",
  [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithException(memberName, sourceFilePath, sourceLineNumber,
    exception, message, _logger.LogError);
}
public void LogAppError(string message, [CallerMemberName] string memberName = "",
  [CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithoutException(memberName, sourceFilePath, sourceLineNumber, message, _logger.LogError);
}
public void LogAppCritical(Exception exception, string message,
  [CallerMemberName] string memberName = "", [CallerFilePath] string sourceFilePath = "",
  [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithException(memberName, sourceFilePath, sourceLineNumber, exception, message,
    _logger.LogCritical);
}
```

```
public void LogAppCritical(string message, [CallerMemberName] string memberName = "",
[CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithoutException(memberName, sourceFilePath, sourceLineNumber, message, _logger.LogCritical);
}
public void LogAppDebug(string message, [CallerMemberName] string memberName = "",
  [CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithoutException(memberName, sourceFilePath, sourceLineNumber, message, _logger.LogDebug);
}
public void LogAppTrace(string message, [CallerMemberName] string memberName = "",
[CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithoutException(memberName, sourceFilePath, sourceLineNumber, message, _logger.LogTrace);
}
public void LogAppInformation(string message, [CallerMemberName] string memberName = "",
  [CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithoutException(memberName, sourceFilePath, sourceLineNumber, message,
    _logger.LogInformation);
}
public void LogAppWarning(string message, [CallerMemberName] string memberName = "",
  [CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber = 0)
{
  LogWithoutException(memberName, sourceFilePath, sourceLineNumber, message, _logger.LogWarning);
}
```

Step 3: Add the Logging Configuration Extension Method

- Create a new folder named `Configuration` to the `Logging` folder. Add a new class named `LoggingConfiguration.cs` to the `Configuration` directory. Make the class public and static:

```
namespace AutoLot.Services.Logging.Configuration;

public static class LoggingConfiguration
{
  //implementation goes here
}
```

- Add variables to hold the output template (for text file logging) and the `ColumnOptions` (for SQL Server logging):

```
private static readonly string OutputTemplate =
  @"[{Timestamp:yy-MM-dd HH:mm:ss}
{Level}]{ApplicationName}:{SourceContext}{NewLine}Message:{Message}{NewLine}in method {MemberName}
at {FilePath}:{LineNumber}{NewLine}{Exception}{NewLine}";
```

```
private static readonly ColumnOptions ColumnOptions = new()
{
  AdditionalColumns = new List<SqlColumn>
  {
    new() { DataType = SqlDbType.VarChar, ColumnName = "ApplicationName" },
    new() { DataType = SqlDbType.VarChar, ColumnName = "MachineName" },
    new() { DataType = SqlDbType.VarChar, ColumnName = "MemberName" },
    new() { DataType = SqlDbType.VarChar, ColumnName = "FilePath" },
    new() { DataType = SqlDbType.Int, ColumnName = "LineNumber" },
    new() { DataType = SqlDbType.VarChar, ColumnName = "SourceContext" },
    new() { DataType = SqlDbType.VarChar, ColumnName = "RequestPath" },
    new() { DataType = SqlDbType.VarChar, ColumnName = "ActionName" }
  }
};
```

- Add the `WebApplicationBuilder` extension method to register `Serilog` as the logging framework for ASP.NET Core:

```
public static void ConfigureSerilog(this WebApplicationBuilder builder)
{
  builder.Logging.ClearProviders();
  var config = builder.Configuration;
  var settings = config.GetSection(nameof(AppLoggingSettings)).Get<AppLoggingSettings>();
  var connectionStringName = settings.MSSqlServer.ConnectionStringName;
  var connectionString = config.GetConnectionString(connectionStringName);
  var tableName = settings.MSSqlServer.TableName;
  var schema = settings.MSSqlServer.Schema;
  string restrictedToMinimumLevel = settings.General.RestrictedToMinimumLevel;
  if (!Enum.TryParse<LogEventLevel>(restrictedToMinimumLevel, out var logLevel))
  {
    logLevel = LogEventLevel.Debug;
  }

  var sqlOptions = new MSSqlServerSinkOptions
  {
    AutoCreateSqlTable = false,
    SchemaName = schema,
    TableName = tableName,
  };
  if (builder.Environment.IsDevelopment())
  {
    sqlOptions.BatchPeriod = new TimeSpan(0, 0, 0, 1);
    sqlOptions.BatchPostingLimit = 1;
  }
  var log = new LoggerConfiguration()
    .MinimumLevel.Is(logLevel)
    .Enrich.FromLogContext()
    .Enrich.With(new PropertyEnricher(
       "ApplicationName", config.GetValue<string>("ApplicationName")))
    .Enrich.WithMachineName()
    .WriteTo.File(
       path: builder.Environment.IsDevelopment()
         ? settings.File.FileName
         : settings.File.FullLogPathAndFileName, // "ErrorLog.txt",
       rollingInterval: RollingInterval.Day,
       restrictedToMinimumLevel: logLevel,
       outputTemplate: OutputTemplate)
    .WriteTo.Console(restrictedToMinimumLevel: logLevel)
    .WriteTo.MSSqlServer(
       connectionString: connectionString,
       sqlOptions,
       restrictedToMinimumLevel: logLevel,
       columnOptions: ColumnOptions);
  builder.Logging.AddSerilog(log.CreateLogger(), false);
}
```

Part 2: Add the String Utility Extension Method

- Add a new folder named `Utilities` and, in that folder, add a new class file named `StringExtensions.cs`. Update the code to match the following:

```
using System;
namespace AutoLot.Services.Utilities;

public static class StringExtensions
{
  public static string RemoveController(this string original)
      => original.Replace("Controller", "", StringComparison.OrdinalIgnoreCase);
}
```

Summary

This lab created the Services project used by the ASP.NET Core projects.

Next steps

In the next part of this tutorial series, you will start working with ASP.NET Core.