.NET App Dev Hands-On Lab

Razor Pages/MVC/API Lab 2a – Common Services

This lab builds the shared services used by the ASP.NET Core applications. Before starting this lab, you must have completed Razor Pages/MVC/API Lab 1. This 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.Base;
global using AutoLot.Dal.Repos.Interfaces;
global using AutoLot.Dal.Repos.Interfaces.Base;

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

Part 1: Add Logging Support

Step 1: Add the Logging Settings View Model

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

```
namespace AutoLot.Services.Logging.Settings;

public class AppLoggingSettings
{
  public GeneralSettings General { get; set; }
  public FileSettings File { get; set; }
  public SqlServerSettings MSSqlServer { get; set; }

  public class GeneralSettings
  {
    public string RestrictedToMinimumLevel { get; set; }
  }
  public class SqlServerSettings
  {
    public string TableName { get; set; }
    public string Schema { get; set; }
    public string ConnectionStringName { get; set; }
  }

  public class FileSettings
  {
    public string Drive { get; set; }
    public string FilePath { get; set; }
    public string FileName { get; set; }
    public string FullLogPathAndFileName =>
$"{Drive}{Path.VolumeSeparatorChar}{Path.DirectorySeparatorChar}{FilePath}{Path.DirectorySeparator
Char}{FileName}";
  }
}
```

Step 2: Add the Logging Interface

- In the Logging folder, add a new folder named Interfaces; 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 to the GlobalUsings.cs file:

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

Step 3: Add the Logging Implementation

- In the Logging folder, add a class file named AppLogging.cs. Make the class public and generic and implement IAppLogging. Add a default constructor that takes an instance of ILogger<T>:

```
namespace AutoLot.Services.Logging;

public class AppLogging<T>(ILogger<T> logger) : IAppLogging<T>
{
  //implementation goes here
}
```

- 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 4: Add the Logging Configuration Extension Method

- Create a new folder named `Configuration` in the `Logging` folder. Add a new class named `LoggingConfiguration.cs` to the `Configuration` directory. Make the class public and static and add a method to register the `IAppLogging` interface with the ASP.NET Core DI Service Collection:

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

public static class LoggingConfiguration
{
  public static IServiceCollection RegisterLoggingInterfaces(this IServiceCollection services)
  {
    services.AddScoped(typeof(IAppLogging<>), typeof(AppLogging<>));
    return services;
  }
}
```

- Add public static variables to the class 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 extension method to register Serilog as the logging framework for ASP.NET Core:

```csharp
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)
    .MinimumLevel.Override("Microsoft", LogEventLevel.Error)
    .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);
  if (builder.Environment.IsDevelopment())
  {
    Serilog.Debugging.SelfLog.Enable(msg =>
    {
      Debug.Print(msg);
      Debugger.Break();
    });
  }
  builder.Logging.AddSerilog(log.CreateLogger(), false);
}
```

Part 2: Add the String Utility Extension Method

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

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

Part 3: Add the SimpleService Interface and Classes

- Create a new folder named `Simple`, and in that folder, add a new folder named `Interfaces`. In that folder, create a new interface named `ISimpleService.cs` and update the contents to the following:

```
namespace AutoLot.Services.Simple.Interfaces;
public interface ISimpleService
{
  string SayHello();
}
```

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

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

- Create two new classes named `SimpleServiceOne` and `SimpleServiceTwo`, and update them to the following:

```
//SimpleServiceOne
namespace AutoLot.Services.Simple;
public class SimpleServiceOne : ISimpleService
{
  public string SayHello() => "Hello from One";
}
//SimpleServiceTwo
namespace AutoLot.Services.Simple;
public class SimpleServiceTwo : ISimpleService
{
  public string SayHello() => "Hello from Two";
}
```

Part 4: Add the DealerInfo ViewModel

- In the `AutoLot.Services` project, create a new folder named `ViewModels`, and in that folder, create a new file named `DealerInfo.cs` and update the contents to the following:

```
namespace AutoLot.Services.ViewModels;
public class DealerInfo
{
  public string DealerName { get; set; }
  public string City { get; set; }
  public string State { get; set; }
}
```

Summary

This lab created the logging infrastructure, added the string extension method, and added the `SimpleService` to the `AutoLot.Services` project.

Next steps

In the next part of this tutorial series, you will update the configuration settings for the ASP.NET Core application.