

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.

Part 1: Add Logging Support

- Delete the generated `Class1.cs` file.

Step 1: Add the Logging Interface

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

```
using System;
using System.Runtime.CompilerServices;

namespace AutoLot.Services.Logging
{
    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);
}
}

```

Step 2: Add the Logging Implementation

- In the Logging folder add a class file named AppLogging.cs. Add the following using statements to the class:

```

using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using Serilog.Context;

```

- Make the class public and generic, and implement IAppLogging:

```

namespace AutoLot.Services.Logging
{
    public class AppLogging<T> : IAppLogging<T>
    {
    }
}

```

- Inject into the constructor the framework ILogger<T> and an instance of IConfiguration and create private variables for them. Also get the application name from the configuration:

```

private readonly ILogger<T> _logger;
private readonly IConfiguration _config;
private readonly string _applicationName;

public AppLogging(ILogger<T> logger, IConfiguration config)
{
    _logger = logger;
    _config = config;
    _applicationName = config.GetValue<string>("ApplicationName");
}

```

- Create an internal method to push the additional properties into the SeriLog context:

```
internal List<IDisposable> PushProperties(
    string memberName,
    string sourceFilePath,
    int sourceLineNumber)
{
    List<IDisposable> list = new List<IDisposable>
    {
        LogContext.PushProperty("MemberName", memberName),
        LogContext.PushProperty("FilePath", sourceFilePath),
        LogContext.PushProperty("LineNumber", sourceLineNumber),
        LogContext.PushProperty("ApplicationName", _applicationName)
    };
    return list;
}
```

- Implement the logging interface members:

```
public void LogAppError(Exception exception, string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogError(exception, message);
    foreach (var item in list) { item.Dispose(); }
}

public void LogAppError(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogError(message);
    foreach (var item in list) { item.Dispose(); }
}

public void LogAppCritical(Exception exception, string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogCritical(exception, message);
    foreach (var item in list) { item.Dispose(); }
}

public void LogAppCritical(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogCritical(message);
    foreach (var item in list) { item.Dispose(); }
}
```

```

public void LogAppDebug(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogDebug(message);
    foreach (var item in list) { item.Dispose(); }
}

public void LogAppTrace(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogTrace(message);
    foreach (var item in list) { item.Dispose(); }
}

public void LogAppInformation(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogInformation(message);
    foreach (var item in list) { item.Dispose(); }
}

public void LogAppWarning(string message,
    [CallerMemberName] string memberName = "",
    [CallerFilePath] string sourceFilePath = "",
    [CallerLineNumber] int sourceLineNumber = 0)
{
    var list = PushProperties(memberName, sourceFilePath, sourceLineNumber);
    _logger.LogWarning(message);
    foreach (var item in list) { item.Dispose(); }
}

```

Step 3: Add the Logging Configuration Extension Method

- Add a new class file named `LoggingConfiguration.cs` to the `Logging` directory. Add the following using statements to the top of the file:

```

using System;
using System.Collections.Generic;
using System.Data;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Serilog;
using Serilog.Events;
using Serilog.Sinks.MSSqlServer;

```

- Make the class public and static:

```
namespace AutoLot.Services.Logging
{
    public static class LoggingConfiguration
    {
    }
}
```

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

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

```
public static IHostBuilder ConfigureSerilog(this IHostBuilder builder)
{
    builder
        .ConfigureLogging((context, logging) => { logging.ClearProviders(); })
        .UseSerilog((hostingContext, loggerConfiguration) =>
        {
            var config = hostingContext.Configuration;
            var connectionString = config.GetConnectionString("AutoLot").ToString();
            var tableName = config["Logging:MSSqlServer:tableName"].ToString();
            var schema = config["Logging:MSSqlServer:schema"].ToString();
            string restrictedToMinimumLevel =
            config["Logging:MSSqlServer:restrictedToMinimumLevel"].ToString();
            //LogEventLevel logLevel = log;
            if (!Enum.TryParse<LogEventLevel>(restrictedToMinimumLevel, out var logLevel))
            {
                logLevel = LogEventLevel.Debug;
            }
            LogEventLevel level =
            (LogEventLevel)Enum.Parse(typeof(LogEventLevel), restrictedToMinimumLevel);
            loggerConfiguration
                .Enrich.FromLogContext()
                .Enrich.WithMachineName()
                .ReadFrom.Configuration(hostingContext.Configuration)
                .WriteTo.File(
                    path:"ErrorLog.txt",
                    rollingInterval:RollingInterval.Day,
                    restrictedToMinimumLevel:logLevel,
                    outputTemplate:OutputTemplate)
                .WriteTo.Console(restrictedToMinimumLevel:logLevel)
                .WriteTo.MSSqlServer(
                    connectionString: connectionString,
                    new MSSqlServerSinkOptions
                    {
                        AutoCreateSqlTable = false,
                        SchemaName = schema,
                        TableName = tableName,
                        BatchPeriod = new TimeSpan(0,0,0,1)
                    },
                    restrictedToMinimumLevel: level,
                    columnOptions: ColumnOptions);
        });
    return builder;
}
```

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.