

Organizing Program.cs Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Create Constants Class

You want to avoid hard-coding whenever you can in your applications. Let's get rid of the hard-coded string you used for CORS.

Right mouse-click on the Web API project and add a new folder named **ConstantClasses**.

Right mouse-click on the ConstantClasses folder and add a new class named **AdvWorksAPIConstants**. Replace the entire contents of this new file with the following code.

```
namespace AdvWorksAPI.ConstantClasses;

public class AdvWorksAPIConstants
{
    public const string CORS_POLICY =
    "AdvWorksAPICorsPolicy";
}
```

Open the **Program.cs** file and add a using statement:

```
using AdvWorksAPI.ConstantClasses;
```

Replace the **two locations** in the **Program.cs** file where you used the string "AdvWorksAPICorsPolicy" with this new constant.

```
AdvWorksAPIConstants.CORS_POLICY
```

Lab 2: Create Service Extension Class

The Program.cs file is getting to be quite large. It is a good idea to start breaking it up into smaller chunks.

Right mouse-click on the Web API project and add a new folder named **ExtensionClasses**.

Right mouse-click on the ExtensionClasses folder and add a new class named **ServiceExtensions**.

Replace the entire contents of this new file with the following code.

```
using AdvWorksAPI.ConstantClasses;

namespace AdvWorksAPI.ExtensionClasses;

public static class ServiceExtension
{
    public static void AddRepositoryClasses(this
IServiceCollection services)
    {
        // Add Repository Classes
        services.AddScoped<IRepository<Customer>,
CustomerRepository>();
    }
}
```

Open the **Program.cs** file and add a using statement:

```
using AdvWorksAPI.ExtensionClasses;
```

Replace the line of code that added the CustomerRepository to the Services collection with the following code.

```
// Add & Configure Repository Classes
builder.Services.AddRepositoryClasses();
```

Lab 4: Create Host Extension Class

Right mouse-click on the ExtensionClasses folder and add a new class named **HostExtensions**. Replace the entire contents of this new file with the following code.

```
using Serilog;
using Serilog.Events;

namespace AdvWorksAPI.ExtensionClasses;

public static class HostExtension
{
    public static IHostBuilder ConfigureSerilog(this
    IHostBuilder host)
    {
        return host.UseSerilog((ctx, lc) =>
        {
            lc.WriteTo.File("Logs/InfoLog-.txt",
                rollingInterval: RollingInterval.Day,
                restrictedToMinimumLevel:
                LogEventLevel.Information);
            lc.WriteTo.File("Logs/ErrorLog-.txt",
                rollingInterval: RollingInterval.Day,
                restrictedToMinimumLevel: LogEventLevel.Error);
        });
    }
}
```

Open the **Program.cs** file and locate the call to the `UseSerilog()` method and replace it with the following code.

```
// Add & Configure Logging using Serilog
builder.Host.ConfigureSerilog();
```

Try it Out

Compile the code and run the application to ensure everything still works as it should.

Lab 5: Create Configure Global Defaults

Right mouse-click on the ExtensionClasses folder and create a new class named **WebApplicationBuilderExtensions**. Replace the entire contents of this new file with the following code.

```
using AdvWorksAPI.EntityLayer;

namespace AdvWorksAPI.ExtensionClasses
{
    public static class WebApplicationBuilderExtensions
    {
        public static void ConfigureGlobalSettings(this
WebApplicationBuilder builder)
        {
            // Configure Global Settings
            builder.Services.AddSingleton<AdvWorksAPIDefaults,
AdvWorksAPIDefaults>();

            // Read "AdvWorksAPI" section and add as a
singleton
            AdvWorksAPIDefaults settings = new();

            builder.Configuration.GetSection("AdvWorksAPI").Bind(set
tings);

            builder.Services.AddSingleton<AdvWorksAPIDefaults>(setti
ngs);
        }
    }
}
```

Open the **Program.cs** file and replace the code where you added the AdvWorksAPIDefaults to the services container with the following code.

```
// *****
// Add and Configure Services
// *****
// Add & Configure Global Application Settings
builder.ConfigureGlobalSettings();
```

Lab 7: Add Router Classes

Open the **ServiceExtension.cs** file and add a new method named `AddRouterClasses()`.

```
public static void AddRouterClasses(this
IServiceCollection services)
{
    // Add "Router" classes as a service
    services.AddScoped<RouterBase, CustomerRouter>();
    services.AddScoped<RouterBase, SettingsRouter>();
    services.AddScoped<RouterBase, LogTestRouter>();
    services.AddScoped<RouterBase, ErrorRouter>();
}
```

Open the **Program.cs** file and where you previously added these router classes, replace those lines of code with the following:

```
// Add "Router" classes as a service
builder.Services.AddRouterClasses();
```

Try it Out

Compile the code and run the application to ensure everything still works as it should.

Go to the top of the `Program.cs` file and remove any unused **using** statements. Also notice how much better organized the `Program.cs` file is now.