

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**.

```
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 IServiceCollection ConfigureCors(this
    IServiceCollection services)
    {
        // Add & Configure CORS
        return services.AddCors(options =>
        {
            options.AddPolicy(AdvWorksAPIConstants.CORS_POLICY,
                builder =>
                {
                    builder.WithOrigins("http://localhost:5081");
                });
        });
    }
}
```

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

```
using AdvWorksAPI.ExtensionClasses;
```

Locate the call to the AddCors() method and replace it with the following code.

```
// Add & Configure CORS  
builder.Services.ConfigureCors();
```

Lab 3: Create MVC Builder Extension Class

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

```
namespace AdvWorksAPI.ExtensionClasses;  
  
public static class MvcBuilderExtensions  
{  
    public static IMvcBuilder ConfigureJsonOptions(this  
        IMvcBuilder builder)  
    {  
        // Configure JSON Options  
        return builder.AddJsonOptions(options =>  
        {  
            // Make all property names start with upper-case  
  
            //options.JsonSerializerOptions.PropertyNamingPolicy =  
            null;  
            // Ignore "readonly" fields  
  
            options.JsonSerializerOptions.IgnoreReadOnlyProperties =  
            true;  
        });  
    }  
}
```

Open the **Program.cs** file and locate where the configuration of the JSON options was and replace it with the following code.

```
// Add & Configure JSON Options  
mvcBuilder.ConfigureJsonOptions();
```

Feel free to comment out the code in the ConfigureJsonOptions() method if you don't want to have PascalCase property names, or to ignore readonly properties.

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) =>
        {
            // Log to Console
            lc.WriteTo.Console();
            // Log to Rolling File
            lc.WriteTo.File("Logs/InfoLog-.txt",
                rollingInterval: RollingInterval.Day,
                restrictedToMinimumLevel:
                LogEventLevel.Information);
            // Log Errors to Rolling File
            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.ConstantClasses;
using AdvWorksAPI.EntityLayer;

namespace AdvWorksAPI.ExtensionClasses
{
    public static class WebApplicationBuilderExtensions
    {
        // Configure Global Settings
        public static void ConfigureGlobalSettings(this
WebApplicationBuilder builder)
        {
            // The following line is only used for the
SettingsController
            builder.Services.AddSingleton<AdvWorksAPIDefaults,
AdvWorksAPIDefaults>();

            // Read "AdvWorksAPI" section
            // Use the IOptionsMonitor<AdvWorksAPIDefaults> in
controller's constructor

            builder.Services.Configure<AdvWorksAPIDefaults>(builder.
Configuration.GetSection("AdvWorksAPI"));
        }
    }
}
```

Open the **Program.cs** file and immediately after the **// Add and Configure Services** comment, add the following code.

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

Lab 6: Create Repository Method

Open the **ServiceExtensions.cs** file and add a new method

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

Open the **Program.cs** file and just after the code **builder.ConfigureGlobalSettings()** add the following code.

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

Try it Out

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

Open the **Program.cs** file and go to the top and **remove** any unused **using** statements.