Router Classes 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 Router Base Class

Right mouse-click on the project and add a new folder named **BaseClasses**.

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

```
namespace AdvWorksAPI.BaseClasses;

public abstract class RouterBase
{
  public RouterBase()
  {
    UrlFragment = string.Empty;
    TagName = string.Empty;
  }

  public string UrlFragment;
  public string TagName;

  public abstract void AddRoutes(WebApplication app);
}
```

Lab 2: Create Customer Router Class

Right mouse-click on the project and add a new folder named **RouterClasses**.

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

```
using AdvWorksAPI.BaseClasses;
using AdvWorksAPI.EntityLayer;
using AdvWorksAPI.RepositoryLayer;

namespace AdvWorksAPI.RouterClasses;

public class CustomerRouter : RouterBase
{
  public CustomerRouter()
  {
    UrlFragment = "api/Customer";
    TagName = "Customer";
  }

  /// <summary>
  /// Add routes
  /// </summary>
  /// <param name="app">A WebApplication object</param>
  public override void AddRoutes(WebApplication app)
  {
    app.MapGet($"/{UrlFragment}", () => Get())
       .WithTags(TagName)
       .Produces(200)
       .Produces<List<Customer>>()
       .Produces(404);
  }

  protected virtual IResult Get()
  {
    List<Customer> list;

    // Get all customers
    list = new CustomerRepository().Get();

    // Simulate not getting any data
    //list.Clear();

    if (list == null || list.Count == 0) {
      return Results.NotFound("No Customers Found.");
    }
    else {
      return Results.Ok(list);
    }
  }
}
```

NOTE:	We are not using DI yet for the Repository object. That will come later.

Open the **Program.cs** file and add two new using statements.

```
using AdvWorksAPI.BaseClasses;
using AdvWorksAPI.RouterClasses;
```

Locate the call to the **app.MapGet("/api/Customer", (IRepository<Customer> repo) => { … }** API and replace it with the following code:

```
//*********************************************
// Map Minimal API Endpoints
//*********************************************
new CustomerRouter().AddRoutes(app);
```

Try it Out

Run the program and make sure you can still get all customers

Lab 3: Get a Single Customer in Router Class

Open the **CustomerRouter.cs** file and add another method:

```
protected virtual IResult Get(int id)
{
  Customer? entity;

  // Attempt to get a single product
  entity = new CustomerRepository().Get(id);
  if (entity == null) {
    return Results.NotFound($"Customer with Customer ID
= '{id}' Not Found.");
  }
  else {
    return Results.Ok(entity);
  }
}
```

Modify the AddRoutes() method and add a new app.MapGet()

```
app.MapGet($"/{UrlFragment}/{{id:int}}", (int id) =>
Get(id))
       .WithTags(TagName)
       .Produces(200)
       .Produces<Customer>()
       .Produces(404);
```

Open the **Program.cs** file and locate the call to the **app.MapGet("/api/Customer/{id:int}", (int id, IRepository<Customer> repo) => {…}** API and **REMOVE** the code.

It is now included within the call to the AddRoutes(app) method.

Try it Out

Run the application and ensure both product APIs still work.

Lab 4: Inject into Router Methods

You want to inject the IRepository<Customer> interface into the CustomerRouter class. Open the **Program.cs** file and add the RouterBase and CustomerRouter as shown in the code in **bold** below.

```
// ***********************************************
// Add and Configure Services
// ***********************************************
builder.Services.AddSingleton<AdvWorksAPIDefaults,
AdvWorksAPIDefaults>();
builder.Services.AddScoped<IRepository<Customer>,
CustomerRepository>();
builder.Services.AddScoped<RouterBase,
CustomerRouter>();
```

In order to do ensure DI will inject the IRepository<T> into the CustomerRouter class, you must wrap the creation of the CustomerRouter() within the scope of the application object.

Remove the code you just wrote that called the AddRoutes(app).

```
new CustomerRouter().AddRoutes(app);
```

Scroll down to where you see app.Run() and replace that code with the following.

```
//*********************************************
// Map Minimal API Endpoints by
// Adding Routes from All Router Classes
// Run the Application
//*********************************************
using (var scope = app.Services.CreateScope()) {
  var services =
scope.ServiceProvider.GetServices<RouterBase>();
  // Loop through each RouterBase class
  foreach (var item in services) {
    // Invoke the AddRoutes() method to add the routes
    item.AddRoutes(app);
  }

  // Run the Application
  app.Run();
}
```

NOTE:	You must include the app.Run() method within the scope in order to keep the Router classes within the scope and to work with DI.

Open the **CustomerRouter.cs** file and add a readonly field

```
private readonly IRepository<Customer> _Repo;
```

Modify the constructor to look like the following:

```
public CustomerRouter(IRepository<Customer> repo)
{
  UrlFragment = "api/Customer";
  TagName = "Customer";
  _Repo = repo;
}
```

Remove the **new CustomerRepository()** with **_Repo** in both the Get() and Get(int id) methods.

Try it Out

Run the application and verify all APIs still work.

Lab 5: Create Settings Router Class

Let's get rid of the rest of the individual API calls in the Program.cs file.

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

```
using AdvWorksAPI.BaseClasses;
using AdvWorksAPI.EntityLayer;

namespace AdvWorksAPI.RouterClasses;

public class SettingsRouter : RouterBase
{
  public SettingsRouter()
  {
    UrlFragment = "api/Settings";
    TagName = "Settings";
  }

  /// <summary>
  /// Add routes
  /// </summary>
  /// <param name="app">A WebApplication object</param>
  public override void AddRoutes(WebApplication app)
  {
    app.MapGet($"/{UrlFragment}", (AdvWorksAPIDefaults
settings)
      => Results.Ok(settings))
      .WithTags(TagName);
    app.MapGet($"/{UrlFragment}Again",
(AdvWorksAPIDefaults settings)
      => Results.Ok(settings))
      .WithTags(TagName);
  }
}
```

Open the **Program.cs** file and delete all the /api/GetSettings API.

Inject the SettingsRouter into DI just below where you injected the CustomerRouter class.

```
builder.Services.AddScoped<RouterBase,
SettingsRouter>();
```

Try it Out

Run the application and verify all Settings APIs still work.

Lab 6: Delete WeatherForecast

Remove all traces of the weather forecast APIs and data from the Program.cs