Entity Framework 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: Add Entity Framework

Open the **CustomerRouter.cs** file and remove the call to **.RequireAuthorization("GetCustomersClaim")**.

Add Entity Framework

Right mouse-click on the project and select **Manage NuGet Packages…**

Click on the Browse tab and install the **Microsoft.EntityFrameworkCore.SqlServer** (add the latest version that works with .NET 6, probably 7.0.3) package.

Lab 2: Add Data Annotations to Customer Class

Open the **Customer.cs** file and replace the entire contents of this file with the following code.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AdvWorksAPI.EntityLayer;

[Table("Customer", Schema = "SalesLT")]
public partial class Customer
{
  public Customer()
  {
    Title = string.Empty;
    FirstName = string.Empty;
    MiddleName = string.Empty;
    LastName = string.Empty;
    CompanyName = string.Empty;
    EmailAddress = string.Empty;
    Phone = string.Empty;
    PasswordHash = string.Empty;
    PasswordSalt = string.Empty;
    SalesPerson = string.Empty;
    Suffix = string.Empty;
  }

  [Key]
  [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  [Required()]
  public int CustomerID { get; set; }

  [Required()]
  public bool NameStyle { get; set; }

  public string? Title { get; set; }

  [Required()]
  public string FirstName { get; set; }

  public string? MiddleName { get; set; }

  [Required()]
  public string LastName { get; set; }

  public string? Suffix { get; set; }

  public string? CompanyName { get; set; }

  public string? SalesPerson { get; set; }
  public string? EmailAddress { get; set; }

  public string? Phone { get; set; }

  [Required()]
  public string PasswordHash { get; set; }

  [Required()]
  public string PasswordSalt { get; set; }

  [Required()]
  public Guid Rowguid { get; set; }

  [Required()]
  public DateTime ModifiedDate { get; set; }

  #region ToString Override
  public override string ToString()
  {
    return $"{LastName}, {FirstName} ({CustomerID})";
  }
  #endregion
}
```

Lab 3: Add an EF DbContext Class

Right mouse-click on the project and create a folder named **Models**.

Right mouse-click on the **Models** folder and add a class named **AdvWorksLTDbContext**

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

```
using AdvWorksAPI.EntityLayer;
using Microsoft.EntityFrameworkCore;

namespace AdvWorksAPI.Models;

public partial class AdvWorksLTDbContext : DbContext
{
  public
AdvWorksLTDbContext(DbContextOptions<AdvWorksLTDbContext
> options) : base(options)
  {
  }

  public virtual DbSet<Customer> Customers { get; set; }

  protected override void OnModelCreating(ModelBuilder
modelBuilder)
  {
    base.OnModelCreating(modelBuilder);
  }
}
```

Add Connection String to appsettings file

Open the **appsettings.Development.json** file. Add a "ConnectionStrings" property.

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=Localhost;Database=AdventureWorksLT;Trusted_Conn
ection=True;MultipleActiveResultSets=true;TrustServerCer
tificate=True;Application Name=AdvWorksAPI;"
  },
  // REST OF THE JSON HERE
}
```

Modify the connection string to correspond to your SQL Server and database.

NOTE: If running local, you might need to add the **TrustServerCertificate=True**

Lab 4: Add DbContext to DI Services

Open the **ServiceExtension.cs** file and add two using statements.

```
using AdvWorksAPI.Models;
using Microsoft.EntityFrameworkCore;
```

Add a new method to configure the Entity Framework.

```
public static IServiceCollection
ConfigureAdventureWorksDB(this IServiceCollection
services, string? cnn)
{
  // Setup the DbContext object
  return services.AddDbContext<AdvWorksLTDbContext>(
    options => options.UseSqlServer(cnn));
}
```

Open the **Program.cs** file and call this new method just after you call the ConfigureGlobalSettings() as shown in the code in **bold** below.

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

// Add & Configure AdventureWorksLT DbContext
builder.Services.ConfigureAdventureWorksDB(builder.Confi
guration.GetConnectionString("DefaultConnection"));
```

Lab 5: Get Customers from Database

Open the **CustomerRepository.cs** file and add a using statement.

```
using AdvWorksAPI.Models;
```

Add a private variable and a constructor to this class.

```
private readonly AdvWorksLTDbContext _DbContext;

public CustomerRepository(AdvWorksLTDbContext context) {
  _DbContext = context;
}
```

Locate the Get() method and **replace** the return statement and all of the hard-coded customer objects with the code shown in **bold** below.

```
public List<Customer> Get()
{
  return _DbContext.Customers.OrderBy(row =>
row.LastName).ToList();
}
```

Try it Out

Run the application and click on the **GET /api/Customer** button.

You should now be getting customer data from the database.

Lab 6: Refactor CustomerRouter Class

When you use the Entity Framework and a DbContext object, that DbContext object should NOT be passed into the constructor of the Router classes. Since that class is created one time at the beginning of the program, the DbContext class is then just a single instance. You need different instances for each call to each Web API endpoint for best performance and to avoid concurrency issues.

Open the **CustomerRouter.cs** file and remove the *private readonly IRepository<Customer> _Repo*. Also remove the injection into the constructor of this *IRepository<Customer>* object.

Locate the AddRoutes() method and inject the *IRepository<Customer>* object into both methods.

```
public override void AddRoutes(WebApplication app)
{
  app.MapGet($"/{UrlFragment}",
      (IRepository<Customer> repo) => Get(repo))
       .WithTags(TagName)
       .Produces(200)
       .Produces<List<Customer>>()
       .Produces(404)
       .Produces(500);
       //.RequireAuthorization("GetCustomersClaim");

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

Locate the **Get()** method and add a parameter of the type IRepository<Customer>. Change the **_Repo** to be just **repo**.

```
protected virtual IResult Get(IRepository<Customer>
repo)
{
  IResult ret;
  List<Customer> list;
  InfoMessage = "No Customers Found.";

  try {
    // Intentionally Cause an Exception
    //throw new ApplicationException("ERROR!");

    list = repo.Get();

    // REST OF THE CODE HERE
}
```

Locate the **Get(int id)** method and add a parameter of the type IRepository<Customer>. Change the **_Repo** to be just **repo**.

```
protected virtual IResult Get(int id,
  IRepository<Customer> repo)
{
  Customer? entity;

  // Attempt to get a single product
  entity = repo.Get(id);
  // REST OF THE CODE HERE
}
```

Lab 7: Get a Single Customer

Open the **CustomerRepository.cs** file and modify the Get(id) method with the code shown below.

```
public Customer Get(int id) {
  return _DbContext.Customers.Where(row =>
row.CustomerID == id).FirstOrDefault();
}
```

Try it Out

Run the application and click on the **GET /api/Customer/{id}** button

Enter **235** and execute