

Build an ASP.NET Core Service and App with .NET (Core) 5.0 Two-Day Hands-On Lab

Lab 6

This lab walks you through creating the Data Initializer. Prior to starting this lab, you must have completed Lab 5.

Part 1: Create the Sample Data provider

- Create a new folder named Initialization in the AutoLot.Dal project
- Add a file named SampleData.cs to the folder, and add the following using statements to the top of the class:

```
using System.Collections.Generic;
using AutoLot.Models.Entities;
using AutoLot.Models.Entities.Owned;
```

- Update the class to the following:

```
namespace AutoLot.Dal.Initialization
{
    public static class SampleData
    {
        public static List<Customer> Customers => new()
        {
            new() {Id = 1, PersonalInformation = new Person {FirstName = "Dave", LastName = "Brenner"}},
            new() {Id = 2, PersonalInformation = new Person {FirstName = "Matt", LastName = "Walton"}},
            new() {Id = 3, PersonalInformation = new Person {FirstName = "Steve", LastName = "Hagen"}},
            new() {Id = 4, PersonalInformation = new Person {FirstName = "Pat", LastName = "Walton"}},
            new() {Id = 5, PersonalInformation = new Person {FirstName = "Bad", LastName = "Customer"}},
        };

        public static List<Make> Makes => new()
        {
            new() {Id = 1, Name = "VW"},
            new() {Id = 2, Name = "Ford"},
            new() {Id = 3, Name = "Saab"},
            new() {Id = 4, Name = "Yugo"},
            new() {Id = 5, Name = "BMW"},
            new() {Id = 6, Name = "Pinto"},
        };
    }
}
```

```

public static List<Car> Inventory => new()
{
    new() {Id = 1, MakeId = 1, Color = "Black", PetName = "Zippy"},
    new() {Id = 2, MakeId = 2, Color = "Rust", PetName = "Rusty"},
    new() {Id = 3, MakeId = 3, Color = "Black", PetName = "Mel"},
    new() {Id = 4, MakeId = 4, Color = "Yellow", PetName = "Clunker"},
    new() {Id = 5, MakeId = 5, Color = "Black", PetName = "Bimmer"},
    new() {Id = 6, MakeId = 5, Color = "Green", PetName = "Hank"},
    new() {Id = 7, MakeId = 5, Color = "Pink", PetName = "Pinky"},
    new() {Id = 8, MakeId = 6, Color = "Black", PetName = "Pete"},
    new() {Id = 9, MakeId = 4, Color = "Brown", PetName = "Brownie"},
};

public static List<Order> Orders => new()
{
    new() {Id = 1, CustomerId = 1, CarId = 5},
    new() {Id = 2, CustomerId = 2, CarId = 1},
    new() {Id = 3, CustomerId = 3, CarId = 4},
    new() {Id = 4, CustomerId = 4, CarId = 7},
};

public static List<CreditRisk> CreditRisks => new()
{
    new()
    {
        Id = 1,
        CustomerId = Customers[4].Id,
        PersonalInformation = new Person
        {
            FirstName = Customers[4].PersonalInformation.FirstName,
            LastName = Customers[4].PersonalInformation.LastName
        }
    }
};
}
}

```

Part 2: Create the Store Data_INITIALIZER

- In the Initialization folder, create a new file named SampleDataInitializer.cs.
- Update the using statements to match the following:

```

using System;
using System.Collections.Generic;
using System.Linq;
using AutoLot.Dal.EfStructures;
using AutoLot.Models.Entities;
using AutoLot.Models.Entities.Base;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Storage;

```

- Change the class to public and static.

```
namespace AutoLot.Dal.Initialization
{
    public static class SampleDataInitializer
    {
    }
}
```

- The ClearData method clears all data then resets the identity seeds to 1.

```
internal static void ClearData(ApplicationDbContext context)
{
    var entities = new[] {
        typeof(Order).FullName,
        typeof(Customer).FullName,
        typeof(Car).FullName,
        typeof(Make).FullName,
        typeof(CreditRisk).FullName
    };
    foreach (var entityName in entities)
    {
        var entity = context.Model.FindEntityType(entityName);
        var tableName = entity.GetTableName();
        var schemaName = entity.GetSchema();
        context.Database.ExecuteSqlRaw($"DELETE FROM {schemaName}.{tableName}");
        context.Database.ExecuteSqlRaw($"DBCC CHECKIDENT (\"{schemaName}.{tableName}\", RESEED, 1);");
    }
}
```

- The ProcessInsert method adds data to the tables if the tables are empty:

```
internal static void ProcessInsert<TEntity>(ApplicationDbContext context,
    DbSet<TEntity> table, List<TEntity> records) where TEntity : BaseEntity
{
    if (table.Any()) { return; }
    IExecutionStrategy strategy = context.Database.CreateExecutionStrategy();
    strategy.Execute(() =>
    {
        using var transaction = context.Database.BeginTransaction();
        try
        {
            var metaData = context.Model.FindEntityType(typeof(TEntity).FullName);
            context.Database.ExecuteSqlRaw(
                $"SET IDENTITY_INSERT {metaData.GetSchema()}.{metaData.GetTableName()} ON");
            table.AddRange(records);
            context.SaveChanges();
            context.Database.ExecuteSqlRaw(
                $"SET IDENTITY_INSERT {metaData.GetSchema()}.{metaData.GetTableName()} OFF");
            transaction.Commit();
        }
        catch (Exception)
        {
            transaction.Rollback();
        }
    });
}
```

- The SeedData method uses the ProcessInsert method to load the data from the SampleData class.

```
internal static void SeedData(ApplicationDbContext context)
{
    try
    {
        ProcessInsert(context, context.Customers!, SampleData.Customers);
        ProcessInsert(context, context.Makes!, SampleData.Makes);
        ProcessInsert(context, context.Cars!, SampleData.Inventory);
        ProcessInsert(context, context.Orders!, SampleData.Orders);
        ProcessInsert(context, context.CreditRisks!, SampleData.CreditRisks);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
        throw;
    }
}
```

- The DropAndCreateDatabase method deletes the database and then creates the database using the migrations:

```
internal static void DropAndCreateDatabase(ApplicationDbContext context)
{
    context.Database.EnsureDeleted();
    //This doesn't run the migrations, so SQL objects will be missing
    //DON'T USE THIS => context.Database.EnsureCreated();
    context.Database.Migrate();
}
```

- The main entry point methods are InitializeData and ClearAndReseedData. The former drops and recreates the database, the latter clears the data. Then both reseed the data:

```
public static void InitializeData(ApplicationDbContext context)
{
    DropAndCreateDatabase(context);
    SeedData(context);
}

public static void ClearAndReseedDatabase(ApplicationDbContext context)
{
    Context.Database.Migrate();
    ClearData(context);
    SeedData(context);
}
```

Summary

This lab created data initializer, completing the data access layer.

Next steps

The next lab is optional and adds in integration tests for the data access layer. If you choose to skip lab 7 and integration testing, proceed to Lab 8, where you will build the shared services project.