Build an ASP.NET Core MVC App with EF Core One-Day Hands-On Lab

Lab 6

This lab walks you through creating the data initialization code. 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 update the class to the following:

```
namespace AutoLot.Dal.Initialization;

public static class SampleData
{
  public static List<Customer> Customers => new()
  {
    new() { Id = 1, PersonInformation = new() { FirstName = "Dave", LastName = "Brenner" } },
    new() { Id = 2, PersonInformation = new() { FirstName = "Matt", LastName = "Walton" } },
    new() { Id = 3, PersonInformation = new() { FirstName = "Steve", LastName = "Hagen" } },
    new() { Id = 4, PersonInformation = new() { FirstName = "Pat", LastName = "Walton" } },
    new() { Id = 5, PersonInformation = new() { 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<Driver> Drivers => new()
  {
    new() { Id = 1, PersonInformation = new() { FirstName = "Fred", LastName = "Flinstone" } },
    new() { Id = 2, PersonInformation = new() { FirstName = "Barney", LastName = "Rubble" } }
  };
  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" },
    new() { Id = 10, MakeId = 1, Color = "Rust", PetName = "Lemon", IsDrivable = false }
  };
  public static List<Radio> Radios => new()
  {
    new() {Id= 1, CarId = 1, HasSubWoofers = true, RadioId = "SuperRadio 1", HasTweeters = true },
    new() {Id= 2, CarId = 2, HasSubWoofers = true, RadioId = "SuperRadio 2", HasTweeters = true },
    new() {Id= 3, CarId = 3, HasSubWoofers = true, RadioId = "SuperRadio 3", HasTweeters = true },
    new() {Id= 4, CarId = 4, HasSubWoofers = true, RadioId = "SuperRadio 4", HasTweeters = true },
    new() {Id= 5, CarId = 5, HasSubWoofers = true, RadioId = "SuperRadio 5", HasTweeters = true },
    new() {Id= 6, CarId = 6, HasSubWoofers = true, RadioId = "SuperRadio 6", HasTweeters = true },
    new() {Id= 7, CarId = 7, HasSubWoofers = true, RadioId = "SuperRadio 7", HasTweeters = true },
    new() {Id= 8, CarId = 8, HasSubWoofers = true, RadioId = "SuperRadio 8", HasTweeters = true },
    new() {Id= 9, CarId = 9, HasSubWoofers = true, RadioId = "SuperRadio 9", HasTweeters = true },
    new() {Id=10, CarId=10, HasSubWoofers = true, RadioId = "SuperRadio 10", HasTweeters = true }
  };
  public static List<CarDriver> CarsAndDrivers => new()
  {
    new() { Id = 1, CarId = 1, DriverId = 1 },
    new() { Id = 2, CarId = 2, DriverId = 2 }
  };
  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 },
    new() { Id = 5, CustomerId = 5, CarId = 10 }
  };
  public static List<CreditRisk> CreditRisks => new()
  {
    new()
      {
        Id = 1,
        CustomerId = Customers[4].Id,
        PersonInformation = new()
        {
          FirstName = Customers[4].PersonInformation.FirstName,
          LastName = Customers[4].PersonInformation.LastName
        }
      }
  };
}
```

Part 2: Update the Package Reference for Temporal Table Runtime Support

In order to programmatically determine the history table associated with a temporal table at runtime, the `Microsoft.EntityFrameworkCore.Design` package can't be trimmed, which it is by default.

- ☐ Comment out the `IncludeAssets` tag in the `AutoLot.Dal.csproj` file:

```
<PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="6.0.0">
  <!--<IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>-->
  <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

Part 3: Create the Store Data Initializer

- ☐ In the `Initialization` folder, create a new file named `SampleDataInitializer.cs`.
- ☐ Change the class to `public` and `static`.

```
namespace AutoLot.Dal.Initialization;
public static class SampleDataInitializer
{
  //Implementation goes here
}
```

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

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

- ☐ The `ClearData` method clears all data in the tables (including the history data) then resets the identity seeds to 1.

```
internal static void ClearData(ApplicationDbContext context)
{
  var entities = new[]
  {
    typeof(Order).FullName,
    typeof(Customer).FullName,
    typeof(CarDriver).FullName,
    typeof(Driver).FullName,
    typeof(Radio).FullName,
    typeof(Car).FullName,
    typeof(Make).FullName,
    typeof(CreditRisk).FullName
  };
  var serviceCollection = new ServiceCollection();
  serviceCollection.AddDbContextDesignTimeServices(context);
  var serviceProvider = serviceCollection.BuildServiceProvider();
  var designTimeModel = serviceProvider.GetService<IModel>();
  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);");
    if (entity.IsTemporal())
    {
      var strategy = context.Database.CreateExecutionStrategy();
      strategy.Execute(() =>
      {
        using var trans = context.Database.BeginTransaction();
        var designTimeEntity = designTimeModel.FindEntityType(entityName);
        var historySchema = designTimeEntity.GetHistoryTableSchema();
        var historyTable = designTimeEntity.GetHistoryTableName();
        context.Database.ExecuteSqlRaw(
            $"ALTER TABLE {schemaName}.{tableName} SET (SYSTEM_VERSIONING = OFF)");
        context.Database.ExecuteSqlRaw($"DELETE FROM {historySchema}.{historyTable}");
        context.Database.ExecuteSqlRaw(
            $"ALTER TABLE {schemaName}.{tableName} SET (SYSTEM_VERSIONING = ON
(HISTORY_TABLE={historySchema}.{historyTable}))");
        trans.Commit();
      });
    }
  }
}
```

- The `SeedData` method calls a local function to add data to each table if it’s empty:

```
internal static void SeedData(ApplicationDbContext context)
{
  try
  {
    ProcessInsert(context, context.Customers, SampleData.Customers);
    ProcessInsert(context, context.Makes, SampleData.Makes);
    ProcessInsert(context, context.Drivers, SampleData.Drivers);
    ProcessInsert(context, context.Cars, SampleData.Inventory);
    ProcessInsert(context, context.Radios, SampleData.Radios);
    ProcessInsert(context, context.CarsToDrivers, SampleData.CarsAndDrivers);
    ProcessInsert(context, context.Orders, SampleData.Orders);
    ProcessInsert(context, context.CreditRisks, SampleData.CreditRisks);
  }
  catch (Exception ex)
  {
    Console.WriteLine(ex);
    throw;
  }
  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 main entry point methods are `InitializeData` and `ClearAndReseendData`. 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)
{
  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.