Build an EF and ASP.NET Core 2.2 App HOL

Lab 5

This lab walks you through creating the Data Initializer. Prior to starting this lab, you must have completed Lab 4.

Step 1: Create the Sample Data provider

1) Create a new folder named `Initialization` in the `SpyStore.Hol.Dal` project
2) Copy the `StoreSampleData.cs` file from the `Code\Completed\Lab5\SpyStore.Hol.Dal\Initialization` folder into the `Initialization` folder created in the prior step. An excerpt of the file is listed here:

```
public static IEnumerable<Category> GetCategories() => new List<Category>
{
  new Category
  {
    CategoryName = "Communications",
    Products = new List<Product>
    {
      new Product
      {
        Details = new ProductDetails
        {
          ProductImage = "product-image.png",
          ProductImageLarge = "product-image-lg.png",
          ProductImageThumb = "product-thumb.png",
          ModelName = "Communications Device",
          Description =
            "Subversively stay in touch with this miniaturized wireless communications device.
Speak into the pointy end and listen with the other end! Voice-activated dialing makes calling for
backup a breeze. Excellent for undercover work at schools, rest homes, and most corporate
headquarters. Comes in assorted colors.",
          ModelNumber = "RED1",
        },
        UnitCost = 49.99M,
        CurrentPrice = 49.99M,
        UnitsInStock = 2,
        IsFeatured = true
      },
      new Product
      {
        Details = new ProductDetails
        {
          ProductImage = "product-image.png",
          ProductImageLarge = "product-image-lg.png",
          ProductImageThumb = "product-thumb.png",
          ModelName = "Persuasive Pencil",
          Description =
            "Persuade anyone to see your point of view!  Captivate your friends and enemies alike!
Draw the crime-scene or map out the chain of events.  All you need is several years of training or
copious amounts of natural talent. You're halfway there with the Persuasive Pencil. Purchase this
item with the Retro Pocket Protector Rocket Pack for optimum disguise.",
```

```
          ModelNumber = "LK4TLNT",
        },
        UnitCost = 1.99M,
        CurrentPrice = 1.99M,
        UnitsInStock = 5,
      }
    }
  }
}
public static IEnumerable<Customer> GetAllCustomerRecords(IList<Product> products)
  => new List<Customer>
  {
    new Customer()
    {
      EmailAddress = "spy@secrets.com",
      Password = "Foo",
      FullName = "Super Spy",
      Orders = new List<Order>
      {
        new Order()
        {
          OrderDate = DateTime.Now.Subtract(new TimeSpan(20, 0, 0, 0)),
          ShipDate = DateTime.Now.Subtract(new TimeSpan(5, 0, 0, 0)),
          OrderDetails = new List<OrderDetail>
          {
            new OrderDetail()
            {
               ProductNavigation = products[0], Quantity = 3, UnitCost = products[0].CurrentPrice
            },
            new OrderDetail()
            {
              ProductNavigation = products[1], Quantity = 2, UnitCost = products[1].CurrentPrice
            },
            new OrderDetail()
            {
              ProductNavigation = products[2], Quantity = 5, UnitCost = products[3].CurrentPrice
            },
          }
        }
      },
      ShoppingCartRecords = new List<ShoppingCartRecord>
      {
        new ShoppingCartRecord
        {
          DateCreated = DateTime.Now, ProductNavigation = products[3], Quantity = 1
        }
      }
    }
  };
}
```

Step 2: Create the Store Data Initializer

1) In the `Initialization` folder, create a new file named `StoreDataInitializer.cs`.
2) Update the using statements to match the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using SpyStore.Hol.Dal.EfStructures;
using SpyStore.Hol.Models.Entities;
```

3) Change the class to `public` and `static`. Add a method to drop and create the database and a method to reseed the identity columns for all tables:
   NOTE: The `EnsureCreated` method builds the database from the entity model, but doesn't execute any migrations and blocks future migrations. The `Migration` method creates the database (if needed) and runs all migrations for a DbContext.

```
public static class StoreDataInitializer
{
  public static void DropAndCreateDatabase(StoreContext context)
  {
    context.Database.EnsureDeleted();
    context.Database.Migrate();
  }
  internal static void ResetIdentity(StoreContext context)
  {
    var tables = new[] {"Categories", "Customers", "OrderDetails", "Orders", "Products",
                        "ShoppingCartRecords"};
    foreach (var itm in tables)
    {
      var rawSqlString = $"DBCC CHECKIDENT (\"Store.{itm}\", RESEED, 0);";
#pragma warning disable EF1000 // Possible SQL injection vulnerability.
      context.Database.ExecuteSqlCommand(rawSqlString);
#pragma warning restore EF1000 // Possible SQL injection vulnerability.
    }
  }
}
```

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

```
public static void ClearData(StoreContext context)
{
  context.Database.ExecuteSqlCommand("Delete from Store.Categories");
  context.Database.ExecuteSqlCommand("Delete from Store.Customers");
  ResetIdentity(context);
}
```

5) The `SeedData` method loads the data from the `StoreSampleData` class.

```
internal static void SeedData(StoreContext context)
{
  try
  {
    if (!context.Categories.Any())
    {
      context.Categories.AddRange(SampleData.GetCategories());
      context.SaveChanges();
    }
    if (!context.Customers.Any())
    {
      var prod1 = context.Categories.Include(c => c.Products)
                         .FirstOrDefault()?.Products.Skip(3).FirstOrDefault();
      var prod2 = context.Categories.Skip(2).Include(c => c.Products)
                         .FirstOrDefault()?.Products.Skip(2).FirstOrDefault();
      var prod3 = context.Categories.Skip(5).Include(c => c.Products)
                         .FirstOrDefault()?.Products.Skip(1).FirstOrDefault();
      var prod4 = context.Categories.Skip(2).Include(c => c.Products).FirstOrDefault()?
                         .Products.Skip(1).FirstOrDefault();
      context.Customers
               .AddRange(SampleData.GetAllCustomerRecords(
                        new List<Product> { prod1, prod2, prod3, prod4 }));
      context.SaveChanges();
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine(ex);
  }
}
```

6) The entry point method is `InitializeData`. First execute the `Migrate` method of the `DatabaseFacade` to make sure all migrations have been executed. Then call `ClearData` to reset the database, and then `SeedData` to load it with test data:

```
public static void InitializeData(StoreContext context)
{
  context.Database.Migrate();
  ClearData(context);
  SeedData(context);
}
```

Summary

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

Next steps

In the next part of this tutorial series, you will start working with ASP.NET Core.