Build an EF and ASP.NET Core 2.1 App HOL

Lab 5

Welcome to the Build an Entity Framework Core and ASP.NET Core 2.1 Application in a Day Hands-on Lab. 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.

Step 2: Create the Store Data Initializer

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

```
using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
```

3) Change the class to public and static:

```
public static class StoreDataInitializer
{
}
```

4) 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);
}
```

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

```
public static void ClearData(StoreContext context)
{
  ExecuteDeleteSql(context, "Categories");
  ExecuteDeleteSql(context, "Customers");
  ResetIdentity(context);
}
internal static void ExecuteDeleteSql(StoreContext context, string tableName)
{
  //With 2.0, must separate string interpolation if not passing in params
  var rawSqlString = $"Delete from Store.{tableName}";
#pragma warning disable EF1000 // Possible SQL injection vulnerability.
  context.Database.ExecuteSqlCommand(rawSqlString);
#pragma warning restore EF1000 // Possible SQL injection vulnerability.
}
internal static void ResetIdentity(StoreContext context)
{
  var tables = new[] {"Categories","Customers",
    "OrderDetails","Orders","Products","ShoppingCartRecords"};
  foreach (var itm in tables)
  {
    //With 2.0, must separate string interpolation if not passing in params
    var rawSqlString = $"DBCC CHECKIDENT (\"Store.{itm}\", RESEED, -1);";
#pragma warning disable EF1000 // Possible SQL injection vulnerability.
    context.Database.ExecuteSqlCommand(rawSqlString);
#pragma warning restore EF1000 // Possible SQL injection vulnerability.
  }
}
```

6) The SeedData method loads the data from the StoreSampleData class.

```
internal static void SeedData(StoreContext context)
{
  try
  {
    if (!context.Categories.Any())
    {
      context.Categories.AddRange(StoreSampleData.GetCategories());
      context.SaveChanges();
    }
    if (!context.Products.Any())
    {
      context.Products.AddRange(StoreSampleData.GetProducts(context.Categories.ToList()));
      context.SaveChanges();
    }
    if (!context.Customers.Any())
    {
      context.Customers.AddRange(StoreSampleData.GetAllCustomerRecords(context));
      context.SaveChanges();
    }
    var customer = context.Customers.FirstOrDefault();
    if (!context.Orders.Any())
    {
      context.Orders.AddRange(StoreSampleData.GetOrders(customer, context));
      context.SaveChanges();
    }
    if (!context.OrderDetails.Any())
    {
      var order = context.Orders.First();
      context.OrderDetails.AddRange(StoreSampleData.GetOrderDetails(order,context));
      context.SaveChanges();
    }
    if (!context.ShoppingCartRecords.Any())
    {
      context.ShoppingCartRecords.AddRange(StoreSampleData.GetCart(customer, context));
      context.SaveChanges();
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine(ex);
  }
}
```

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.