Build an EF and ASP.NET Core 2.0 App HOL

Welcome to the Build an Entity Framework Core and ASP.NET Core 2.0 Application in a Day Hands On Lab. This lab walks you through creating the core of the data access library.

Prior to starting this lab, you must have completed Lab 1.

All labs and files are available at https://github.com/skimedic/dotnetcore_hol.

SpyStore_HOL.Models Project Changes

Step 1: Creating the Models

Step 1: Create the Base Entity

1) Create a new folder in the SpyStore_HOL.Models project named Entities. Create a subfolder under that named Base.
2) Add a new class to the Base folder named EntityBase.cs
3) Add the following using statements to the class:

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

4) Update the code for the EntityBase.cs class to the following:

```
public abstract class EntityBase
{
  [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  public int Id { get; set; }
  [Timestamp]
  public byte[] TimeStamp { get; set; }
}
```

Step 2: Create the Category Model

NOTE: The project won't compile until all of the models have been added.

1) Add a new class to the Entities folder named Category.cs
2) Add the following using statements to the class:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the Category.cs class to the following:

```
[Table("Categories", Schema = "Store")]
public class Category : EntityBase
```

```
{
  [DataType(DataType.Text), MaxLength(50)]
  public string CategoryName { get; set; }
  [InverseProperty(nameof(Product.Category))]
  public List<Product> Products { get; set; } = new List<Product>();
}
```

Step 3: Create the Customer Model

1) Add a new class to the Entities folder named Customer.cs
2) Add the following using statements to the class:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the Customer.cs class to the following:

```
[Table("Customers", Schema = "Store")]
public class Customer : EntityBase
{
  [DataType(DataType.Text), MaxLength(50), Display(Name = "Full Name")]
  public string FullName { get; set; }
  [Required]
  [EmailAddress]
  [DataType(DataType.EmailAddress), MaxLength(50), Display(Name = "Email Address")]
  public string EmailAddress { get; set; }
  [Required]
  [DataType(DataType.Password), MaxLength(50)]
  public string Password { get; set; }
  [InverseProperty(nameof(Order.Customer))]
  public List<Order> Orders { get; set; }= new List<Order>();
  [InverseProperty(nameof(ShoppingCartRecord.Customer))]
  public virtual List<ShoppingCartRecord> ShoppingCartRecords { get; set; } = new List<ShoppingCartRecord>();
}
```

Step 4: Create the Order Model

1) Add a new class to the Entities folder named Order.cs
2) Add the following using statements to the class:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the Order.cs class to the following:

```
[Table("Orders", Schema = "Store")]
public class Order : EntityBase
{
  public int CustomerId { get; set; }
  [DataType(DataType.Date)]
```

```
  [Display(Name = "Date Ordered")]
  public DateTime OrderDate { get; set; }
  [DataType(DataType.Date)]
  [Display(Name = "Date Shipped")]
  public DateTime ShipDate { get; set; }
  [ForeignKey("CustomerId")]
  public Customer Customer { get; set; }
  [InverseProperty("Order")]
  public List<OrderDetail> OrderDetails { get; set; } = new List<OrderDetail>();
}
```

Step 5: Create the OrderDetail Model

1) Add a new class to the Entities folder named OrderDetail.cs
2) Add the following using statements to the class:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the OrderDetail.cs class to the following:

```
[Table("OrderDetails", Schema = "Store")]
public class OrderDetail : EntityBase
{
  [Required]
  public int OrderId { get; set; }
  [Required]
  public int ProductId { get; set; }
  [Required]
  public int Quantity { get; set; }
  [Required, DataType(DataType.Currency), Display(Name = "Unit Cost")]
  public decimal UnitCost { get; set; }
  [DataType(DataType.Currency), Display(Name = "Total")]
  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  public decimal? LineItemTotal { get; set; }
  [ForeignKey(nameof(OrderId))]
  public Order Order { get; set; }
  [ForeignKey(nameof(ProductId))]
  public Product Product { get; set; }
}
```

Step 6: Create the Product Model

1) Add a new class to the Entities folder named Product.cs
2) Add the following using statements to the class:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the Product.cs class to the following:

```
[Table("Products", Schema = "Store")]
public class Product : EntityBase
{
  [MaxLength(3800)]
  public string Description { get; set; }
  [MaxLength(50)]
  public string ModelName { get; set; }
  public bool IsFeatured { get; set; }
  [MaxLength(50)]
  public string ModelNumber { get; set; }
  [MaxLength(150)]
  public string ProductImage { get; set; }
  [MaxLength(150)]
  public string ProductImageLarge { get; set; }
  [MaxLength(150)]
  public string ProductImageThumb { get; set; }
  [DataType(DataType.Currency)]
  public decimal UnitCost { get; set; }
  [DataType(DataType.Currency)]
  public decimal CurrentPrice { get; set; }
  public int UnitsInStock { get; set; }
  [Required]
  public int CategoryId { get; set; }
  [ForeignKey(nameof(CategoryId))]
  public Category Category { get; set; }
  [InverseProperty(nameof(ShoppingCartRecord.Product))]
  public List<ShoppingCartRecord> ShoppingCartRecords { get; set; } = new List<ShoppingCartRecord>();
  [InverseProperty(nameof(OrderDetail.Product))]
  public List<OrderDetail> OrderDetails { get; set; } = new List<OrderDetail>();
}
```

Step 7: Create the ShoppingCartRecord Model

1) Add a new class to the Entities folder named ShoppingCartRecord.cs
2) Add the following using statements to the class:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the ShoppingCartRecord.cs class to the following:

```
[Table("ShoppingCartRecords", Schema = "Store")]
public class ShoppingCartRecord : EntityBase
{
  [DataType(DataType.Date)]
  public DateTime? DateCreated { get; set; }
  public int CustomerId { get; set; }
  [ForeignKey(nameof(CustomerId))]
  public Customer Customer { get; set; }
```

```
  public int Quantity { get; set; }
  public decimal LineItemTotal { get; set; }
  public int ProductId { get; set; }
  [ForeignKey(nameof(ProductId))]
  public Product Product { get; set; }
}
```

Part 2: Create the ViewModels

Step 1: Create the Base ViewModel

The base view model combines the Product and the Category classes, adding in the Display data annotations for the MVC rendering engine.

1) Create a new folder in the SpyStore_HOL.Models project named ViewModels. Create a subfolder under that named Base.
2) Add a new class to the Base folder named ProductAndCategoryBase.cs
3) Add the following using statements to the class:

```
using System.ComponentModel.DataAnnotations;
using SpyStore_HOL.Models.Entities.Base;
```

4) Update the code for the ProductAndCategoryBase.cs class to the following:

```
public class ProductAndCategoryBase : EntityBase
{
  public int CategoryId { get; set; }
  [Display(Name = "Category")]
  public string CategoryName { get; set; }
  public int ProductId { get; set; }
  [MaxLength(3800)]
  public string Description { get; set; }
  [MaxLength(50)]
  [Display(Name = "Model")]
  public string ModelName { get; set; }
  [Display(Name="Is Featured Product")]
  public bool IsFeatured { get; set; }
  [MaxLength(50)]
  [Display(Name = "Model Number")]
  public string ModelNumber { get; set; }
  [MaxLength(150)]
  public string ProductImage { get; set; }
  [MaxLength(150)]
  public string ProductImageLarge { get; set; }
  [MaxLength(150)]
  public string ProductImageThumb { get; set; }
  [DataType(DataType.Currency), Display(Name = "Cost")]
  public decimal UnitCost { get; set; }
  [DataType(DataType.Currency), Display(Name = "Price")]
  public decimal CurrentPrice { get; set; }
  [Display(Name="In Stock")]
```

```
  public int UnitsInStock { get; set; }
}
```

Step 2: Create the CartRecordWithProductInfo Model

1) Add a new class to the ViewModels folder named CartRecordWithProductInfo.cs
2) Add the following using statements to the class:

```
using System;
using System.ComponentModel.DataAnnotations;
using SpyStore_HOL.Models.ViewModels.Base;
```

3) Update the code for the CartRecordWithProductInfo.cs class to the following:

```
public class CartRecordWithProductInfo : ProductAndCategoryBase
{
  [DataType(DataType.Date), Display(Name = "Date Created")]
  public DateTime? DateCreated { get; set; }
  public int? CustomerId { get; set; }
  public int Quantity { get; set; }
  [DataType(DataType.Currency), Display(Name = "Line Total")]
  public decimal LineItemTotal { get; set; }
}
```

Step 3: Create the OrderDetailWithProductInfo Model

1) Add a new class to the ViewModels folder named OrderDetailWithProductInfo.cs
2) Add the following using statements to the class:

```
using System;
using System.ComponentModel.DataAnnotations;
using SpyStore_HOL.Models.ViewModels.Base;
```

3) Update the code for the OrderDetailWithProductInfo.cs class to the following:

```
public class OrderDetailWithProductInfo : ProductAndCategoryBase
{
  public int OrderId { get; set; }
  [Required]
  public int Quantity { get; set; }
  [DataType(DataType.Currency), Display(Name = "Total")]
  public decimal? LineItemTotal { get; set; }
}
```

Step 4: Create the OrderWithDetailsAndProductInfo Model

1) Add a new class to the ViewModels folder named OrderWithDetailsAndProductInfo.cs
2) Add the following using statements to the class:

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel.DataAnnotations;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the OrderWithDetailsAndProductInfo.cs class to the following:

```
public class OrderWithDetailsAndProductInfo : EntityBase
{
  public int CustomerId { get; set; }
  [DataType(DataType.Currency), Display(Name = "Total")]
  public decimal? OrderTotal { get; set; }
  [DataType(DataType.Date)]
  [Display(Name = "Date Ordered")]
  public DateTime OrderDate { get; set; }
  [DataType(DataType.Date)]
  [Display(Name = "Date Shipped")]
  public DateTime ShipDate { get; set; }
  public IList<OrderDetailWithProductInfo> OrderDetails { get; set; }
}
```

SpyStore_HOL.DAL Project and Database Updates

Part 1: Create the InvalidQuantityException

Step 1: Create the Custom Exception

1) Create a new folder in the **SpyStore_HOL.DAL** project named Exceptions.
2) Add a new class to the folder named InvalidQuantityException.cs
3) Add the following using statements to the class:

```
using System;
```

4) Update the code to the following:

```
public class InvalidQuantityException : Exception
{
  public InvalidQuantityException() { }
  public InvalidQuantityException(string message) : base(message) { }
  public InvalidQuantityException(string message, Exception innerException) : base(message, innerException)
  { }
}
```

Note: If the project can't find any of the files you just created, close VS and reopen it (that's the problem with brand new software)

Part 2: Create the DbContext (SpyStore_HOL.DAL)

Step 1: Create the DbContext

1) Create a new folder in the SpyStore_HOL.DAL project named EF.

2) Add a new class to the folder named StoreContext.cs

3) Add the following using statements to the class:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Diagnostics;
using SpyStore_HOL.Models.Entities;
```

4) Update the code for the StoreContext.cs class to the following:

   a) Make sure to update the connection string if necessary

```
public class StoreContext : DbContext
{
 internal StoreContext()
 {
 }
 public StoreContext(DbContextOptions options) : base(options)
 {
 }
 protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
 {
  var connectionString =

@"Server=(localdb)\mssqllocaldb;Database=SpyStore_HOL2;Trusted_Connection=True;MultipleActiveResultSets=
true;";
  if (!optionsBuilder.IsConfigured)
  {
   //optionsBuilder.UseSqlServer(connectionString);
   optionsBuilder
    .UseSqlServer(connectionString, options=>options.EnableRetryOnFailure())
    .ConfigureWarnings(warnings=>warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
  }
 }
 protected override void OnModelCreating(ModelBuilder modelBuilder)
 {
  modelBuilder.Entity<Customer>(entity =>
  {
   entity.HasIndex(e => e.EmailAddress).HasName("IX_Customers").IsUnique();
  });
  modelBuilder.Entity<Order>(entity =>
  {
   entity.Property(e => e.OrderDate)
    .HasColumnType("datetime")
    .HasDefaultValueSql("getdate()");
   entity.Property(e => e.ShipDate)
    .HasColumnType("datetime")
    .HasDefaultValueSql("getdate()");
  });
  modelBuilder.Entity<OrderDetail>(entity =>
  {
   entity.Property(e => e.LineItemTotal)
    .HasColumnType("money")
    .HasComputedColumnSql("[Quantity]*[UnitCost]");
```

```
    entity.Property(e => e.UnitCost).HasColumnType("money");
   });
   modelBuilder.Entity<Product>(entity =>
   {
    entity.Property(e => e.UnitCost).HasColumnType("money");
    entity.Property(e => e.CurrentPrice).HasColumnType("money");
   });
   modelBuilder.Entity<ShoppingCartRecord>(entity =>
   {
    entity.HasIndex(e => new { ShoppingCartRecordId = e.Id, e.ProductId, e.CustomerId })
     .HasName("IX_ShoppingCart").IsUnique();
    entity.Property(e => e.DateCreated)
     .HasColumnType("datetime")
     .HasDefaultValueSql("getdate()");
    entity.Property(e => e.Quantity)
     .ValueGeneratedNever()
     .HasDefaultValue(1);
   });
  }
  public DbSet<Category> Categories { get; set; }
  public DbSet<Customer> Customers { get; set; }
  public DbSet<OrderDetail> OrderDetails { get; set; }
  public DbSet<Order> Orders { get; set; }
  public DbSet<Product> Products { get; set; }
  public DbSet<ShoppingCartRecord> ShoppingCartRecords { get; set; }
}
```

Step 2: Create the DbContextFactory

1) Add a new class to the folder named StoreContextFactory.cs
2) Add the following using statements to the class:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.EntityFrameworkCore.Diagnostics;
```

3) Add the following using statements to the class:

```
public class StoreContextFactory : IDesignTimeDbContextFactory<StoreContext>
{
  public StoreContext CreateDbContext(string[] args)
  {
   var optionsBuilder = new DbContextOptionsBuilder<StoreContext>();
   var connectionString = @"Server=(localdb)\mssqllocaldb;Database=SpyStore_HOL2;Trusted_Connection=True;
     MultipleActiveResultSets=true;";
   optionsBuilder
    .UseSqlServer(connectionString,options => options.EnableRetryOnFailure())
    .ConfigureWarnings(warnings => warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
   return new StoreContext(optionsBuilder.Options);
  }
}
```

Part 3: Update the Database and Add the UDF

Step 1: Create and Execute the Initial Migration

1) Open Package Manager Console (View -> Other Windows -> Package Manager Console)
2) Change to the SpyStore_HOL.DAL directory:

```
cd .\SpyStore_HOL.DAL
```

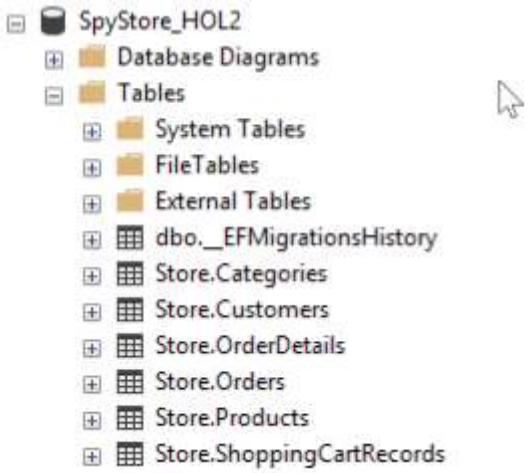
3) Create the initial migration with the following command (-o = output directory, -c = Context File):

```
dotnet ef migrations add Initial -o EF\Migrations -c SpyStore_HOL.DAL.EF.StoreContext
```

4) Check the Up and Down methods to make sure the database and table/column creation code is there
5) Update the database with the following command:

```
dotnet ef database update
```

6) Examine your database in SQL Server Management Studio to make sure the tables were created:



Step 2: Create Migration for the UDF and update the database

1) Create an empty migration (but do **NOT** run database update):

```
dotnet ef migrations add TSQL -o EF\Migrations -c SpyStore_HOL.DAL.EF.StoreContext
```

2) Open up the new migration file (named <timestamp>_TSQL.cs). In the Up method, add the following to create the User Defined Function:

```
string sql = @"CREATE FUNCTION Store.GetOrderTotal ( @OrderId INT )
  RETURNS MONEY WITH SCHEMABINDING
  BEGIN
  DECLARE @Result MONEY;
  SELECT @Result = SUM([Quantity]*[UnitCost]) FROM Store.OrderDetails
  WHERE OrderId = @OrderId; RETURN @Result END";
migrationBuilder.Sql(sql);
```

3) In the Down method, add the following code:

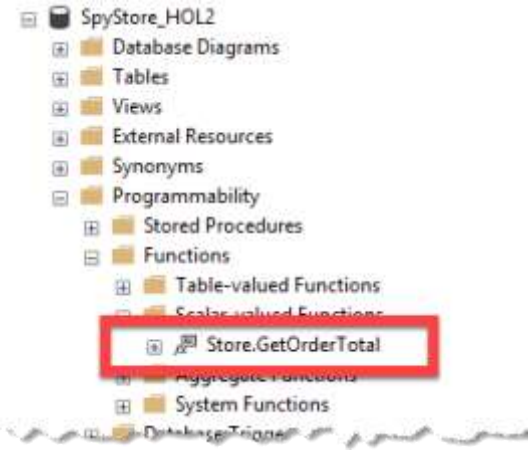
```
migrationBuilder.Sql("DROP FUNCTION [Store].[GetOrderTotal]");
```

4) **SAVE THE MIGRATION FILE**

5) Update the database:

```
dotnet ef database update
```

6) Check the database to make sure the function exists:



Part 4: Add the Calculated Field to the Order Table

Step 1: Update the Order Model

1) Open the Order.cs file in the Models project and add the following property:

```
[Display(Name = "Total")]
public decimal? OrderTotal { get; set; }
```

Step 2: Update the StoreContext OnModelCreating Method

1) Open the StoreContext.cs file in the DAL project, and add the following Fluent API command in the OnModelCreating method to the Order entity:

```
modelBuilder.Entity<Order>(entity =>
{
  entity.Property(e => e.OrderDate)
    .HasColumnType("datetime")
    .HasDefaultValueSql("getdate()");
  entity.Property(e => e.ShipDate)
    .HasColumnType("datetime")
    .HasDefaultValueSql("getdate()");
  entity.Property(e => e.OrderTotal)
    .HasColumnType("money")
    .HasComputedColumnSql("Store.GetOrderTotal([Id])");
});
```

Step 3: Create the Final Migration and Update the Database

1) **SAVE THE StoreContext.cs FILE**

2) Create a new migration using Package Manager Console:

```
dotnet ef migrations add Final -o EF\Migrations -c SpyStore_HOL.DAL.EF.StoreContext
```

3) Update the database using Package Manager Console:

```
dotnet ef database update
```

Step 4: Create the UDF in C#

1) In the StoreContext.cs class, add the following code:

```
[DbFunction("GetOrderTotal",Schema = "Store")]
public static int GetOrderTotal(int orderId)
{
  //code in here doesn't matter
  throw new Exception();
}
```

Part 5: Add the AssemblyInfo.cs File

1) Add an AssemblyInfo.cs file to the SpyStore_HOL.DAL project. Clear out the default code, and replace it with this:

```
using System.Runtime.CompilerServices;
[assembly: InternalsVisibleTo("SpyStore_HOL.Tests")]
```

Summary

This lab created the core parts of the data access library, including the models and the DbContext.

Next steps

In the next part of this tutorial series, you will create the repositories and the data initialization code.