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

Part 1: Creating the Models

The models represent the data that is persisted in SQL Server and can be shaped to be more application specific. For this lab, the mapping is table per hierarchy, the only inheritance model supported by EF Core at this time.

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
{
  public int Id { get; set; }
  public byte[] TimeStamp { get; set; }
}
```

5) Explicitly set the Id field to the primary key for the table and declare the field type as an Identity. Field. **Note:** This is not required because of EF conventions. And field named Id or [ClassName]Id will be set to the PK of the table, and any primary key field that is numeric will be set to an Identity field in SQL Server.

```
[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Id { get; set; }
```

6) Set the TimeStamp property to be a concurrency token using the [TimeStamp] attribute. This creates a timestamp datatype in the SQL Server table.
Note: The timestamp is updated by SQL Server every time a record is added or updated. This field will be used to detect concurrency issues in the application.

```
[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:

```
public class Category : EntityBase
{
  public string CategoryName { get; set; }
  public List<Product> Products { get; set; } = new List<Product>();
}
```

4) Add the [Table] attribute to the class. Set the table name to "Categories" and the Schema to "Store". **NOTE:** In EF Core, the database table name defaults to the name of the DbSet<T> in the DbContext (covered later in this lab).

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

5) Set the MaxLength for the CategoryName field to be 50 characters. This is used by EF Core in shaping the field as well as by ASP.NET Core validations:

```
[MaxLength(50)]
public string CategoryName { get; set; }
```

6) Set the Inverse property of the Products list to the Category property on the Product table.
NOTE: The Products list is the many end of a one-to-many relationship. The Category class itself is the one end. While EF conventions can usually determine the inverse properties, I find it better to be explicit in defining the relationships.

```
[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) Update the using statements to include the following:

```
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 (Data Annotations already discussed or are MVC specific are included in this listing):

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

4) Add the ConcurrencyCheck attribute to the FullName property. This adds the FullName to the base class TimeStamp property when determining if a Concurrency issue has occurred:

```
[MaxLength(50),ConcurrencyCheck]
[DataType(DataType.Text), Display(Name = "Full Name")]
public string FullName { get; set; }
```

5) Add the required attribute to the EmailAddress and Password properties. This sets them to be NotNull in SQL Server (and is also used in MVC validations).
NOTE: By EF Convention, any non-nullable .NET type is set to Not Null in SQL Server. Any nullable type is set to Null unless marked as Required through Data Annotations for the Fluent API.

```
[Required, MaxLength(50)]
[EmailAddress,DataType(DataType.EmailAddress), Display(Name = "Email Address")]
public string EmailAddress { get; set; }
[Required, MaxLength(50)]
public string Password { get; set; }
```

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
{
  [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 int CustomerId { get; set; }
  public Customer Customer { get; set; }
  [InverseProperty(nameof(OrderDetail.Order))]
  public List<OrderDetail> OrderDetails { get; set; } = new List<OrderDetail>();
}
```

4) Explicitly declare the foreign key to the Customer table with the ForeignKey Dat Annotation.
NOTE: By convention, a property of the same data type as the primary key for the related type and named <PrimaryKeyPropertyName>, <NavigationPropertyName><PrimaryKeyPropertyName> or <EntityName><PrimaryKeyPropertyName> will be the foreign key.

```
[ForeignKey(nameof(CustomerId))]
public Customer Customer { get; set; }
```

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")]
  public decimal? LineItemTotal { get; set; }
  [ForeignKey(nameof(OrderId))]
  public Order Order { get; set; }
  [ForeignKey(nameof(ProductId))]
  public Product Product { get; set; }
}
```

4) The LineItemTotal property will become a computed column in SQL Server. Add the DatabaseGenerated Data Annotation with the Computed option as follows:
Note: This is part one of setting this up. This property will be finished using the Fluent API later in this lab.

```
[DataType(DataType.Currency), Display(Name = "Total")]
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]
public decimal? LineItemTotal { 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

ViewModels are a common way to represent data from multiple tables in one class. This lab contains two sets of ViewModels, one set in the Models project and the other in the MVC project.

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; }
}
```

Summary

This lab created the Models and ViewModels for the data access library.

Next steps

In the next part of this tutorial series, you will create the DbContext and the calculated fields, as well as the migrations used to create and update the database.