Build an EF and ASP.NET Core 2.1 App HOL

Lab 2

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 Models and DbContext as well as running your first migration.

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

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.

All files from this section can be copied from Code\Completed\Lab2\SpyStore_HOL.Models\Entities

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. Any 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.Collections.Generic;
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.

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

Step 1: Create the DbContext

1) Create a new folder in the SpyStore_HOL.DAL project named EfStructures.
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) Make the class public and inherit from DbContext. Add in a constructor that takes an instance of DbContextOptions and passes it to the base class:

```
public class StoreContext : DbContext
{
  public StoreContext(DbContextOptions<StoreContext> options) : base(options) { }
}
```

5) Add a DbSet<T> for each of the model classes.

```
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; }
```

6) Add the override for OnModelCreating.
   Note: The OnModelCreating allows for additional shaping of the database using the FluentAPI.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
}
```

7) Add the following code into the OnModelCreating handler:
   a) Add a unique index for the EmailAddress property of the Customer table:

```
modelBuilder.Entity<Customer>(entity =>
{
  entity.HasIndex(e => e.EmailAddress).HasName("IX_Customers").IsUnique();
});
```

   b) Set the SQL Server Data type and the default value for the OrderDate and ShipDate properties of the Order table:

```
modelBuilder.Entity<Order>(entity =>
{
  entity.Property(e => e.OrderDate).HasColumnType("datetime").HasDefaultValueSql("getdate()");
  entity.Property(e => e.ShipDate).HasColumnType("datetime").HasDefaultValueSql("getdate()");
});
```

c) The LineItemTotal is an in-table computed column. The specific computation is Quantity*UnitCost, and must be set using the Fluent API. The SQL Server datatype is set to "money" for the LineItemTotal and UnitCost fields.
NOTE: The Data Annotation on the property in the model class is unnecessary. I add it for clarity since it doesn't cause any issues.

```
modelBuilder.Entity<OrderDetail>(entity =>
{
  entity.Property(e => e.LineItemTotal).HasColumnType("money")
    .HasComputedColumnSql("[Quantity]*[UnitCost]");
  entity.Property(e => e.UnitCost).HasColumnType("money");
});
```

d) Set the SQL Server datatypes for the Product properties UnitCost and CurrentPrice to "money":

```
modelBuilder.Entity<Product>(entity =>
{
  entity.Property(e => e.UnitCost).HasColumnType("money");
  entity.Property(e => e.CurrentPrice).HasColumnType("money");
});
```

e) Create a unique index for the ProductId and CustomerId fields for the ShoppingCartRecord table. Set the default values for the DateCreated and the Quantity fields.
NOTE: Complex indices can only be set using the FluentAPI in EF Core.

```
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).HasDefaultValue(1);
});
```

Step 2: Create the DbContextFactory

The EF Core Tools Migrate and Database Commands must be able to create a context. In prior versions of EF Core, a parameterless constructor was used. That conflicts with the DbContextPool in ASP.NET Core 2. The DesignTimeDbContextFactory class (if found) is used by the EF Core Tools to create the DbContext.

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

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

3) Make the class public and implement IDesignTimeDbContextFactory<StoreContext>:

```
public class StoreContextFactory : IDesignTimeDbContextFactory<StoreContext>
{
}
```

4) The interface has one method, CreateDbContext.
NOTE: The args argument is not used by EF Core at this time (reserved for later use).

```
public StoreContext CreateDbContext(string[] args)
{
}
```

5) If you are using any of LocalDb that isn't SQL Server 2017: In this method, create a new instance of DbContextOptionsBuilder types for the StoreContext and create a variable to hold the connection string (update as necessary):

```
var optionsBuilder = new DbContextOptionsBuilder<StoreContext>();
var connectionString =
@"Server=(localdb)\mssqllocaldb;Database=SpyStore_HOL2.1;Trusted_Connection=True;
  MultipleActiveResultSets=true;";
```

6) If you are using SQL Server 2017 LocalDb: In this method, create a new instance of DbContextOptionsBuilder types for the StoreContext and create a variable to hold the connection string (update as necessary):

```
var optionsBuilder = new DbContextOptionsBuilder<StoreContext>();
var path = Environment.GetEnvironmentVariable("APPDATA");
var connectionString =
$@"Data Source=(localdb)\mssqllocaldb2017;Initial
Catalog=SpyStore_HOL2.1_2017;Trusted_Connection=True;MultipleActiveResultSets=true;AttachDbFileNam
e={path}\SpyStore_HOL_2017.mdf;";
```

7) Opt-in to using SQL Server, setting the connection string and enabling connection resiliency. Next, configure EF to treat mixed mode query evaluation as an exception and not a warning. Finally, rerun the configured StoreContext using the DbContextOptions.

```
optionsBuilder
  .UseSqlServer(connectionString,options => options.EnableRetryOnFailure())
  .ConfigureWarnings(warnings => warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
return new StoreContext(optionsBuilder.Options);
```

Part 3: Update the Database

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):

NOTE: The following lines must be entered as one line in Package Manager Console – copying and pasting from this document doesn't work

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

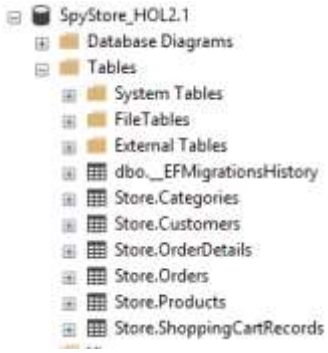
NOTE: The above lines must be entered as one line in Package Manager Console – copying and pasting from this document doesn't work

NOTE: If you get the message "`The EF Core tools version '2.1.1-rtm-30846' is older than that of the runtime '2.1.2-rtm-30932'. Update the tools for the latest features and bug fixes.`**" Ignore it. This is a known bug, and will be resolved in a future release.**

4) This creates three files in the EfStructures\Migrations Directory:
    a) A file named XYZ_Initial.cs (where XYZ is a series of numbers)
    b) A file named XYZ_Initial.Designer.cs (where XYZ is the same series of numbers)
    c) StoreContextModelSnapshot.cs
5) Open up the XYZ_Initial.cs file. Check the Up and Down methods to make sure the database and table/column creation code is there
6) Update the database with the following command:

```
dotnet ef database update
```

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



Summary

In this lab, you created the Models the DbContext (and DbContextFactory), then updated the database using an EF migration.

Next steps

In the next part of this tutorial series, you will create the UserDefinedFunction, add a computed column to the Orders table, and add in all of the ViewModels.