Build an EF and ASP.NET Core 2 App HOL

Welcome to the Build an Entity Framework Core and ASP.NET Core 2 Application in a Day Hands On Lab. This lab walks you through creating the repositories and their interfaces for the data access library.

Prior to starting this lab, you must have completed Lab 2 Part 1.

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

Part 1: Creating the Repositories

Step 1: Create the Base Repository Interface

While the DbContext can be considered an implementation of the repository pattern, it's better to create specific repositories for the models. These will be leveraged by ASP.NET Core in a later module.

1) Create a new folder in the SpyStore_HOL.DAL project named Repos. Create a subfolder under that named Base.
2) Add a new interface to the Base folder named IRepo.cs
3) Add the following using statements to the interface:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using SpyStore_HOL.Models.Entities.Base;
```

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

```
public interface IRepo<T> where T : EntityBase
{
 int Count { get; }
 bool HasChanges { get; }
 bool Any();
 bool Any(Expression<Func<T, bool>> where);
 IEnumerable<T> GetAll();
 IEnumerable<T> GetAll<TIncludeField>(Expression<Func<T, TIncludeField>> include);
 IEnumerable<T> GetAll<TSortField>(Expression<Func<T, TSortField>> orderBy, bool ascending);
 IEnumerable<T> GetAll<TIncludeField, TSortField>(
  Expression<Func<T, TIncludeField>> include,
  Expression<Func<T, TSortField>> orderBy, bool ascending);
 T First();
 T First(Expression<Func<T, bool>> where);
 T First<TIncludeField>(Expression<Func<T, bool>> where, Expression<Func<T, TIncludeField>> include);
 T Find(int id);
 T Find(Expression<Func<T, bool>> where);
 T Find<TIncludeField>(Expression<Func<T, bool>> where, Expression<Func<T, TIncludeField>> include);
 IEnumerable<T> GetSome(Expression<Func<T, bool>> where);
 IEnumerable<T> GetSome<TIncludeField>(
```

```
  Expression<Func<T, bool>> where, Expression<Func<T, TIncludeField>> include);
 IEnumerable<T> GetSome<TSortField>(
  Expression<Func<T, bool>> where, Expression<Func<T, TSortField>> orderBy, bool ascending);
 IEnumerable<T> GetSome<TIncludeField, TSortField>(
  Expression<Func<T, bool>> where, Expression<Func<T, TIncludeField>> include,
  Expression<Func<T, TSortField>> orderBy, bool ascending = true);
 IEnumerable<T> FromSql(string sqlString);
 IEnumerable<T> GetRange(int skip, int take);
 IEnumerable<T> GetRange(IQueryable<T> query, int skip, int take);
 int Add(T entity, bool persist = true);
 int AddRange(IEnumerable<T> entities, bool persist = true);
 int Update(T entity, bool persist = true);
 int UpdateRange(IEnumerable<T> entities, bool persist = true);
 int Delete(T entity, bool persist = true);
 int DeleteRange(IEnumerable<T> entities, bool persist = true);
 int Delete(int id, byte[] timeStamp, bool persist = true);
 int SaveChanges();
 void BeginTransaction();
 void CommitTransaction();
 void RollbackTransaction();
}
```

Step 2: Create the Base Repository

1) Add a new class to the Repos/Base folder named RepoBase.cs
2) Add the following using statements to the class:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Linq.Expressions;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.ChangeTracking;
using Microsoft.EntityFrameworkCore.Storage;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.Models.Entities.Base;
```

3) Update the code for the RepoBase.cs class to the following (or add the class from the Code\Assets folder from the GitHub repo):

```
public abstract class RepoBase<T> : IDisposable, IRepo<T> where T : EntityBase, new()
{
 protected readonly StoreContext Db;
 private readonly bool _disposeContext;
 private IDbContextTransaction _transaction;
 protected DbSet<T> Table;
 public StoreContext Context => Db;

 protected RepoBase() : this(new StoreContext())
 {
```

```
  _disposeContext = true;
}

protected RepoBase(DbContextOptions<StoreContext> options)
  : this(new StoreContext(options))
{
  _disposeContext = true;
}

protected RepoBase(StoreContext context)
{
  Db = context;
  Table = Db.Set<T>();
}

public int Count => Table.Count();
public bool HasChanges => Db.ChangeTracker.HasChanges();

public bool Any() => Table.Any();
public bool Any(Expression<Func<T, bool>> where) => Table.Any(@where);

public virtual IEnumerable<T> GetAll() => Table;

public IEnumerable<T> GetAll<TIncludeField>(Expression<Func<T, TIncludeField>> include)
  => Table.Include(include);

public IEnumerable<T> GetAll<TSortField>(Expression<Func<T, TSortField>> orderBy, bool ascending)
  => ascending ? Table.OrderBy(orderBy) : Table.OrderByDescending(orderBy);

public IEnumerable<T> GetAll<TIncludeField, TSortField>(
  Expression<Func<T, TIncludeField>> include, Expression<Func<T, TSortField>> orderBy, bool ascending)
  => ascending ? Table.Include(include).OrderBy(orderBy) : Table.Include(include).OrderByDescending(orderBy);

public T First() => Table.FirstOrDefault();
public T First(Expression<Func<T, bool>> where) => Table.FirstOrDefault(where);

public T First<TIncludeField>(Expression<Func<T, bool>> where, Expression<Func<T, TIncludeField>> include)
  => Table.Where(where).Include(include).FirstOrDefault();

//return Table.SingleOrDefault(x => x.Id == id) mixed mode evaluation;
public T Find(int id) => Table.Find(id);

public T Find(Expression<Func<T, bool>> where) => Table.Where(where).FirstOrDefault();

public T Find<TIncludeField>(Expression<Func<T, bool>> where,
  Expression<Func<T, TIncludeField>> include)
  => Table.Where(@where).Include(include).FirstOrDefault();

public IEnumerable<T> GetSome(Expression<Func<T, bool>> where) => Table.Where(where);

public IEnumerable<T> GetSome<TIncludeField>(Expression<Func<T, bool>> where,
```

```
  Expression<Func<T, TIncludeField>> include)
  => Table.Where(where).Include(include);

public IEnumerable<T> GetSome<TSortField>(
  Expression<Func<T, bool>> where, Expression<Func<T, TSortField>> orderBy, bool ascending)
  => ascending ? Table.Where(where).OrderBy(orderBy) : Table.Where(where).OrderByDescending(orderBy);

public IEnumerable<T> GetSome<TIncludeField, TSortField>(
  Expression<Func<T, bool>> where, Expression<Func<T, TIncludeField>> include,
  Expression<Func<T, TSortField>> orderBy, bool ascending)
  => ascending
    ? Table.Where(where).OrderBy(orderBy).Include(include)
    : Table.Where(where).OrderByDescending(orderBy).Include(include);

public IEnumerable<T> FromSql(string sqlString) => Table.FromSql(sqlString);

public virtual IEnumerable<T> GetRange(int skip, int take) => GetRange(Table, skip, take);

public IEnumerable<T> GetRange(IQueryable<T> query, int skip, int take) => query.Skip(skip).Take(take);

public virtual int Add(T entity, bool persist = true)
{
  Table.Add(entity);
  return persist ? SaveChanges() : 0;
}

public virtual int AddRange(IEnumerable<T> entities, bool persist = true)
{
  Table.AddRange(entities);
  return persist ? SaveChanges() : 0;
}

public virtual int Update(T entity, bool persist = true)
{
  Table.Update(entity);
  return persist ? SaveChanges() : 0;
}

public virtual int UpdateRange(IEnumerable<T> entities, bool persist = true)
{
  Table.UpdateRange(entities);
  return persist ? SaveChanges() : 0;
}

public virtual int Delete(T entity, bool persist = true)
{
  Table.Remove(entity);
  return persist ? SaveChanges() : 0;
}

public virtual int DeleteRange(IEnumerable<T> entities, bool persist = true)
```

```
{
 Table.RemoveRange(entities);
 return persist ? SaveChanges() : 0;
}

internal T GetEntryFromChangeTracker(int? id)
{
 return Db.ChangeTracker.Entries<T>()
  .Select((EntityEntry e) => (T) e.Entity)
  .FirstOrDefault(x => x.Id == id);
}

//TODO: Check For Cascade Delete
public int Delete(int id, byte[] timeStamp, bool persist = true)
{
 var entry = GetEntryFromChangeTracker(id);
 if (entry != null)
 {
  if (timeStamp != null && entry.TimeStamp.SequenceEqual(timeStamp))
  {
   return Delete(entry, persist);
  }
  throw new Exception("Unable to delete due to concurrency violation.");
 }
 Db.Entry(new T {Id = id, TimeStamp = timeStamp}).State = EntityState.Deleted;
 return persist ? SaveChanges() : 0;
}

public int SaveChanges()
{
 try
 {
  return Db.SaveChanges();
 }
 catch (DbUpdateConcurrencyException ex)
 {
  //A concurrency error occurred
  //Should handle intelligently
  Console.WriteLine(ex);
  throw;
 }
 catch (RetryLimitExceededException ex)
 {
  //DbResiliency retry limit exceeded
  //Should handle intelligently
  Console.WriteLine(ex);
  throw;
 }
 catch (Exception ex)
 {
  //Should handle intelligently
```

```
    Console.WriteLine(ex);
    throw;
    //-2146232060
    //throw new Exception($"{ex.HResult}");
   }
  }

  public void BeginTransaction()
  {
   _transaction = Context.Database.BeginTransaction(IsolationLevel.RepeatableRead);
  }

  public void CommitTransaction()
  {
   _transaction.Commit();
  }

  public void RollbackTransaction()
  {
   _transaction.Rollback();
  }

  bool _disposed = false;

  public void Dispose()
  {
   Dispose(true);
   GC.SuppressFinalize(this);
  }

  protected virtual void Dispose(bool disposing)
  {
   if (_disposed)
    return;
   if (disposing)
   {
    // Free any other managed objects here.
    //
   }
   if (_disposeContext)
   {
    Db.Dispose();
   }
   _disposed = true;
  }
}
```

Step 3: Create the Model Specific Interfaces

1) Create a new folder under the Repos folder named Interfaces.

2) Create the following files in the Interfaces folder:

ICategoryRepo.cs
ICustomerRepo.cs
IOrderDetailRepo.cs
IOrderRepo.cs
IProductRepo.cs
IShoppingCartRepo.cs

Step 3a: Implement the ICategoryRepo Interface

1) Add the following using statements to the ICategoryRepo.cs class:

```
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.Models.Entities;
```

2) Update the code for the ICategoryRepo.cs class to the following:

```
public interface ICategoryRepo : IRepo<Category>
{
}
```

Step 3b: Implement the ICustomerRepo Interface

1) Add the following using statements to the ICustomerRepo.cs class:

```
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.Models.Entities;
```

2) Update the code for the ICustomerRepo.cs class to the following:

```
public interface ICustomerRepo : IRepo<Customer>
{
}
```

Step 3c: Implement the IOrderDetailRepo Interface

1) Add the following using statements to the IOrderDetailRepo.cs class:

```
using System.Collections.Generic;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels;
```

2) Update the code for the IOrderDetailRepo.cs class to the following:

```
public interface IOrderDetailRepo : IRepo<OrderDetail>
{
 IEnumerable<OrderDetailWithProductInfo> GetSingleOrderWithDetails(int orderId);
}
```

Step 3d: Implement the IOrderRepo Interface

1) Add the following using statements to the IOrderRepo.cs class:

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

```
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels;
```

2) Update the code for the IOrderRepo.cs class to the following:

```
public interface IOrderRepo : IRepo<Order>
{
 IList<Order> GetOrderHistory(int customerId);
 OrderWithDetailsAndProductInfo GetOneWithDetails(int customerId, int orderId);
}
```

Step 3e: Implement the IProductRepo Interface

1) Add the following using statements to the IProductRepo.cs class:

```
using System.Collections.Generic;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels.Base;
```

2) Update the code for the IProductRepo.cs class to the following:

```
public interface IProductRepo : IRepo<Product>
{
 IList<ProductAndCategoryBase> Search(string searchString);
 IList<ProductAndCategoryBase> GetProductsForCategory(int id);
 IList<ProductAndCategoryBase> GetFeaturedWithCategoryName();
 ProductAndCategoryBase GetOneWithCategoryName(int id);
}
```

Step 3f: Implement the IShoppingCartRepo Interface

1) Add the following using statements to the IShoppingCartRepo.cs class:

```
using System.Collections.Generic;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels;
```

2) Update the code for the IShoppingCartRepo.cs class to the following:

```
public interface IShoppingCartRepo : IRepo<ShoppingCartRecord>
{
 CartRecordWithProductInfo GetShoppingCartRecord(int customerId, int productId);
 IEnumerable<CartRecordWithProductInfo> GetShoppingCartRecords(int customerId);
 ShoppingCartRecord Find(int customerId, int productId);
 int Update(ShoppingCartRecord entity, int? quantityInStock, bool persist = true);
 int Add(ShoppingCartRecord entity, int? quantityInStock, bool persist = true);
}
```

Step 4: Create the Model Specific Repos

As an alternative to typing all of the following code, you can copy the fully implemented repos from the Assets folder.

1) Create the following files in the Repos folder:

```
CategoryRepo.cs
CustomerRepo.cs
OrderDetailRepo.cs
OrderRepo.cs
ProductRepo.cs
ShoppingCartRepo.cs
```

Step 4a: Implement the CategoryRepo Class

1) Add the following using statements to the CategoryRepo.cs class:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.DAL.Repos.Interfaces;
using SpyStore_HOL.Models.Entities;
```

2) Update the code for the CategoryRepo.cs class to the following:

```
public class CategoryRepo : RepoBase<Category>, ICategoryRepo
{
  public CategoryRepo(DbContextOptions<StoreContext> options) : base(options) { }
  public CategoryRepo() { }
  public override IEnumerable<Category> GetAll()  => Table.OrderBy(x => x.CategoryName);
  public override IEnumerable<Category> GetRange(int skip, int take)
    => GetRange(Table.OrderBy(x => x.CategoryName),skip,take);
}
```

Step 4b: Implement the CustomerRepo Class

1) Add the following using statements to the CustomerRepo.cs class:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.DAL.Repos.Interfaces;
using SpyStore_HOL.Models.Entities;
```

2) Update the code for the CustomerRepo.cs class to the following:

```
public class CustomerRepo : RepoBase<Customer>, ICustomerRepo
{
  public CustomerRepo(DbContextOptions<StoreContext> options) : base(options) {  }
  public CustomerRepo() : base() { }
  public override IEnumerable<Customer> GetAll()  => Table.OrderBy(x => x.FullName);
```

```
  public override IEnumerable<Customer> GetRange(int skip, int take)
    => GetRange(Table.OrderBy(x => x.FullName), skip, take);
}
```

Step 4c: Implement the OrderDetailRepo Class

1) Add the following using statements to the OrderDetailRepo.cs class:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.DAL.Repos.Interfaces;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels;
```

2) Update the code for the OrderDetailRepo.cs class to the following:

```
public class OrderDetailRepo : RepoBase<OrderDetail>, IOrderDetailRepo
{
  public OrderDetailRepo(DbContextOptions<StoreContext> options) : base(options)  { }
  public OrderDetailRepo() { }
  internal IEnumerable<OrderDetailWithProductInfo> GetRecords(IQueryable<OrderDetail> query)
    => query
      .Include(x => x.Product)
      .ThenInclude(p => p.Category)
      .Select(x => new OrderDetailWithProductInfo
      {
        OrderId = x.OrderId,
        ProductId = x.ProductId,
        Quantity = x.Quantity,
        UnitCost = x.UnitCost,
        LineItemTotal = x.LineItemTotal,
        Description = x.Product.Description,
        ModelName = x.Product.ModelName,
        ProductImage = x.Product.ProductImage,
        ProductImageLarge = x.Product.ProductImageLarge,
        ProductImageThumb = x.Product.ProductImageThumb,
        ModelNumber = x.Product.ModelNumber,
        CategoryName = x.Product.Category.CategoryName
      })
      .OrderBy(x => x.ModelName);
  public IEnumerable<OrderDetailWithProductInfo> GetSingleOrderWithDetails(int orderId)
    => GetRecords(Table.Where(x => x.Order.Id == orderId));
}
```

Step 4d: Implement the OrderRepo Class

1) Add the following using statements to the OrderRepo.cs class:

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

```
using Microsoft.EntityFrameworkCore;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.DAL.Repos.Interfaces;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels;
```

2) Update the code for the OrderRepo.cs class to the following:

```
public class OrderRepo : RepoBase<Order>, IOrderRepo
{
 private readonly IOrderDetailRepo _orderDetailRepo;
 public OrderRepo(DbContextOptions<StoreContext> options, IOrderDetailRepo orderDetailRepo) : base(options)
 {
  _orderDetailRepo = orderDetailRepo;
 }
 public OrderRepo(IOrderDetailRepo orderDetailRepo)
 {
  _orderDetailRepo = orderDetailRepo;
 }
 public IList<Order> GetOrderHistory(int customerId) => GetSome(x => x.CustomerId == customerId).ToList();
 public OrderWithDetailsAndProductInfo GetOneWithDetails(int customerId, int orderId)
  => Table
   .Where(x => x.CustomerId == customerId && x.Id == orderId)
   .Select(x => new OrderWithDetailsAndProductInfo
   {
    Id = x.Id,
    CustomerId = customerId,
    OrderDate = x.OrderDate,
    ShipDate = x.ShipDate,
    OrderDetails = _orderDetailRepo.GetSingleOrderWithDetails(orderId).ToList()
   })
   .FirstOrDefault();
}
```

Step 4e: Implement the ProductRepo Class

1) Add the following using statements to the ProductRepo.cs class:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.DAL.Repos.Interfaces;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels.Base;
```

2) Update the code for the ProductRepo.cs class to the following:

```
public class ProductRepo : RepoBase<Product>, IProductRepo
{
 public ProductRepo(DbContextOptions<StoreContext> options) : base(options) { }
```

```
  public ProductRepo() : base() { }
  public override IEnumerable<Product> GetAll() => Table.OrderBy(x => x.ModelName);
  public override IEnumerable<Product> GetRange(int skip, int take)
    => GetRange(Table.OrderBy(x => x.ModelName), skip, take);
  internal ProductAndCategoryBase GetRecord(Product p, Category c)
    => new ProductAndCategoryBase()
    {
      CategoryName = c.CategoryName,
      CategoryId = p.CategoryId,
      CurrentPrice = p.CurrentPrice,
      Description = p.Description,
      IsFeatured = p.IsFeatured,
      Id = p.Id,
      ModelName = p.ModelName,
      ModelNumber = p.ModelNumber,
      ProductImage = p.ProductImage,
      ProductImageLarge = p.ProductImageLarge,
      ProductImageThumb = p.ProductImageThumb,
      TimeStamp = p.TimeStamp,
      UnitCost = p.UnitCost,
      UnitsInStock = p.UnitsInStock
    };
  public IList<ProductAndCategoryBase> GetProductsForCategory(int id)
    => Table
      .Where(p => p.CategoryId == id)
      .Include(p => p.Category)
      .Select(item => GetRecord(item, item.Category))
      .OrderBy(x => x.ModelName)
      .ToList();
  public IList<ProductAndCategoryBase> GetFeaturedWithCategoryName()
    => Table
      .Where(p => p.IsFeatured)
      .Include(p => p.Category)
      .Select(item => GetRecord(item, item.Category))
      .OrderBy(x => x.ModelName)
      .ToList();
  public ProductAndCategoryBase GetOneWithCategoryName(int id)
    => Table
      .Where(p => p.Id == id)
      .Include(p => p.Category)
      .Select(item => GetRecord(item, item.Category))
      .SingleOrDefault();
  public IList<ProductAndCategoryBase> Search(string searchString)
    => Table
      .Where(p => Functions.Like(p.Description, $"%{searchString}%")
        || Functions.Like(p.ModelName, $"%{searchString}%"))
      .Include(p => p.Category)
      .Select(item => GetRecord(item, item.Category))
      .OrderBy(x => x.ModelName)
      .ToList();
}
```

Step 4f: Implement the ShoppingCartRepo Class

1) Add the following using statements to the ShoppingCartRepo.cs class:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using SpyStore_HOL.DAL.EF;
using SpyStore_HOL.DAL.Exceptions;
using SpyStore_HOL.DAL.Repos.Base;
using SpyStore_HOL.DAL.Repos.Interfaces;
using SpyStore_HOL.Models.Entities;
using SpyStore_HOL.Models.ViewModels;
```

2) Update the code for the ShoppingCartRepo.cs class to the following:

```
public class ShoppingCartRepo : RepoBase<ShoppingCartRecord>, IShoppingCartRepo
{
 private readonly IProductRepo _productRepo;
 public ShoppingCartRepo(DbContextOptions<StoreContext> options, IProductRepo productRepo) : base(options)
 {
  _productRepo = productRepo;
 }
 public ShoppingCartRepo(IProductRepo productRepo) : base()
 {
  _productRepo = productRepo;
 }
 public override IEnumerable<ShoppingCartRecord> GetAll()  => Table.OrderByDescending(x => x.DateCreated);
 public override IEnumerable<ShoppingCartRecord> GetRange(int skip, int take)
  => GetRange(Table.OrderByDescending(x => x.DateCreated), skip, take);
 public ShoppingCartRecord Find(int customerId, int productId)
 {
  return Table.FirstOrDefault(x => x.CustomerId == customerId && x.ProductId == productId);
 }
 public override int Update(ShoppingCartRecord entity, bool persist = true)
 {
  return Update(entity, _productRepo.Find(entity.ProductId)?.UnitsInStock, persist);
 }
 public int Update(ShoppingCartRecord entity, int? quantityInStock, bool persist = true)
 {
  if (entity.Quantity <= 0)
  {
   return Delete(entity, persist);
  }
  if (entity.Quantity > quantityInStock)
  {
   throw new InvalidQuantityException("Can't add more product than available in stock");
  }
  return base.Update(entity, persist);
```

```
}
public override int Add(ShoppingCartRecord entity, bool persist = true)
{
 return Add(entity, _productRepo.Find(entity.ProductId)?.UnitsInStock, persist);
}
public int Add(ShoppingCartRecord entity, int? quantityInStock, bool persist = true)
{
 var item = Find(entity.CustomerId, entity.ProductId);
 if (item == null)
 {
  if (quantityInStock != null && entity.Quantity > quantityInStock.Value)
  {
   throw new InvalidQuantityException("Can't add more product than available in stock");
  }
  return base.Add(entity, persist);
 }
 item.Quantity += entity.Quantity;
 return item.Quantity <= 0 ? Delete(item, persist) : Update(item, quantityInStock, persist);
}
internal CartRecordWithProductInfo GetRecord(int customerId, ShoppingCartRecord scr, Product p, Category c)
 => new CartRecordWithProductInfo
 {
  Id = scr.Id,
  DateCreated = scr.DateCreated,
  CustomerId = customerId,
  Quantity = scr.Quantity,
  ProductId = scr.ProductId,
  Description = p.Description,
  ModelName = p.ModelName,
  ModelNumber = p.ModelNumber,
  ProductImage = p.ProductImage,
  ProductImageLarge = p.ProductImageLarge,
  ProductImageThumb = p.ProductImageThumb,
  CurrentPrice = p.CurrentPrice,
  UnitsInStock = p.UnitsInStock,
  CategoryName = c.CategoryName,
  LineItemTotal = scr.Quantity * p.CurrentPrice,
  TimeStamp = scr.TimeStamp
 };
public CartRecordWithProductInfo GetShoppingCartRecord(
 int customerId, int productId)
 => Table
  .Where(x => x.CustomerId == customerId && x.ProductId == productId)
  .Include(x => x.Product)
  .ThenInclude(p => p.Category)
  .Select(x => GetRecord(customerId, x, x.Product, x.Product.Category))
  .FirstOrDefault();
public IEnumerable<CartRecordWithProductInfo> GetShoppingCartRecords(
 int customerId)
 => Table
  .Where(x => x.CustomerId == customerId)
```

```
    .Include(x => x.Product)
    .ThenInclude(p => p.Category)
    .Select(x => GetRecord(customerId, x, x.Product, x.Product.Category))
    .OrderBy(x => x.ModelName);
}
```

Part 2: Creating the Data Initializer

Step 1: Create the Sample Data provider

1) Create a new folder named Initialization under the EF folder in the SpyStore_HOL.DAL project
2) Copy the StoreSampleData.cs file from the Assets folder into the Initialization folder.

Step 2: Create the Store Data Initializer

1) Create a new file named StoreDataInitializer.cs.
2) Add the following using statements to the class:

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

3) Add the following using statements to the class:

```
public static class StoreDataInitializer
{
 public static void InitializeData(StoreContext context)
 {
  context.Database.Migrate();
  ClearData(context);
  SeedData(context);
 }
 public static void ClearData(StoreContext context)
 {
  ExecuteDeleteSQL(context, "Categories");
  ExecuteDeleteSQL(context, "Customers");
  ResetIdentity(context);
 }
 public 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}";
  context.Database.ExecuteSqlCommand(rawSqlString);
 }
 public 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);";
    context.Database.ExecuteSqlCommand(rawSqlString);
   }
  }
  public 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.ShoppingCartRecords.Any())
    {
     context.ShoppingCartRecords.AddRange(
      StoreSampleData.GetCart(customer, context));
     context.SaveChanges();
    }
   }
   catch (Exception ex)
   {
    Console.WriteLine(ex);
   }
  }
}
```

Summary

The lab created all of the repositories and their interfaces and the data initializers, completing the data access layer.

Next steps

In the next part of this tutorial series, you will work with the unit tests project.