6.1.0 – CRUD Setup (BLL)

Introduction

In previous lessons we used LINQ queries to get data from the database. In this lesson, you will learn another technique to get data from the database (i.e. all the records of a table) and learn how to set up paging on a `GridView`, and `ListView`, control. In this lesson, you will learn how to create methods to **R**ead all records, **C**reate a new record, and **U**pdate an existing record, and **D**elete an existing record. The web form you need will be created in the next lesson.

BLL Setup

In this lesson, you will be doing CRUD operations on the **Product** table, therefore you will only need to modify the **`ProductController.cs`** file. Add the following methods to the **`ProductController.cs`**:

CREATE

We will need to add a new method to the **`ProductController.cs`** for adding a new record. When adding a new record to the database, you must ensure that all the data being sent to the database is valid. When we added the Data Annotations to the **`Product.cs`** class, we made sure to have a validation annotation for each property that matched the constraints on each column of the database table. This is only one of several ways to ensure there is valid data.

In the method we will add to the **`ProductController.cs`,** we will be able to add additional validations. These validations will validate the data before the creation of a new **`Product`** entity. The code below is the method that needs to be added:

Listing 1: AddProduct(Product)

```
[DataObjectMethod(DataObjectMethodType.Insert, true)]
public void AddProduct(Product productInfo)
{
    using (var context = new eStoreContext())
    {
        // What are the business rules? Add the code for them here.

        // Description is an optional field; i.e. can be NULL
        // THIS CHECK ONLY NEEDS TO BE DONE FOR FIELDS THAT ARE NOT REQUIRED!
        productInfo.Description = string.IsNullOrEmpty(productInfo.Description) ? null :
productInfo.Description;

        // add and commit the changes
        context.Products.Add(productInfo);
        context.SaveChanges();
    }//end using
}//eom
```

Checks to see if the string has data or not; if no data then use a NULL.

READ

We will add a new record to the **ProductController.cs** for reading all the records in the database. We could use a LINQ Query, but a better approach is to use the **DbContext** class. The code for this is shown below:

Listing 2: ListAllTracks()

```
[DataObjectMethod(DataObjectMethodType.Select,false)]
public List<Product> ListAllProducts()
{
    using (var context = new eStoreContext())
    {
        //return all the Products
        return context.Products.ToList();
    }//end using
}//eom
```

Instead of a LINQ query, we use the DbContext class directly.

UPDATE

We will need to add a new method to the **ProductController.cs** for updating an existing record. When updating a record to the database, you must ensure that all the data being sent to the database is valid. When we added the Data Annotations to the **Product.cs** class, we made sure to have a validation annotation for each property that matched the constraints on each column of the database table.

The method we need to add is shown below:

Listing 3: UpdateProduct(Product)

```
[DataObjectMethod(DataObjectMethodType.Update,true)]
public void UpdateProduct(Product productInfo)
{
    using (var context = new eStoreContext())
    {
        // What are the business rules? Add the code for them here.

        // Description is an optional field; i.e. can be NULL
        // THIS CHECK ONLY NEEDS TO BE DONE FOR FIELDS THAT ARE NOT REQUIRED!
        productInfo.Description = string.IsNullOrEmpty(productInfo.Description) ? null :
productInfo.Description;

        // update the existing record
        context.Entry(productInfo).State = System.Data.Entity.EntityState.Modified;
        // commit the changes
        context.SaveChanges();
    }//end using
}//eom
```

If there are any Business Rules, add them here!

Sets the record as Modified.

DELETE

We will need to add a new method to the **`ProductController.cs`** for deleting an existing record. When deleting a record there is no need to validate the data being deleted. The code that needs to be added to the **`ProductController.cs`** is shown below:

Listing 4: ProductController Delete Method

```
[DataObjectMethod(DataObjectMethodType.Delete,true)]
public void DeleteProduct(Product productInfo)
{
    using (var context = new eStoreContext())
    {
        // find the existing record
        var existing = context.Products.Find(productInfo.ProductID);
        // delete the record
        context.Products.Remove(existing);
        // commit the change
        context.SaveChanges();
    }//end using
}//eom
```

Finds the record to delete.

BUILD!

Build your solution. Fix all errors before proceeding to the next lesson. When your Build is successful, you should see something like:

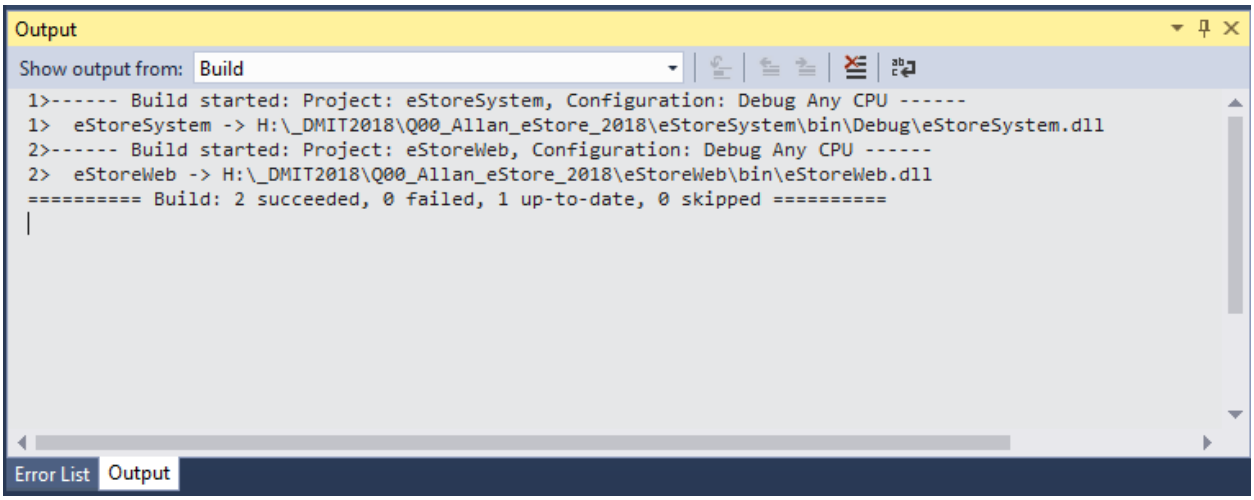

Figure 1: Successful Build

Exercise

Complete Exercise 6.1.1.