

4.3.0 – Purchase Orders

Introduction

In lesson 4.2.0, you were shown how to bind data to a GridView control. In this lesson, you will see how to use a ListView control to display the data being returned from the database. For this lesson, we will use the following Sequence Diagram:

Supporting Documentation & Files

Sequence Diagram

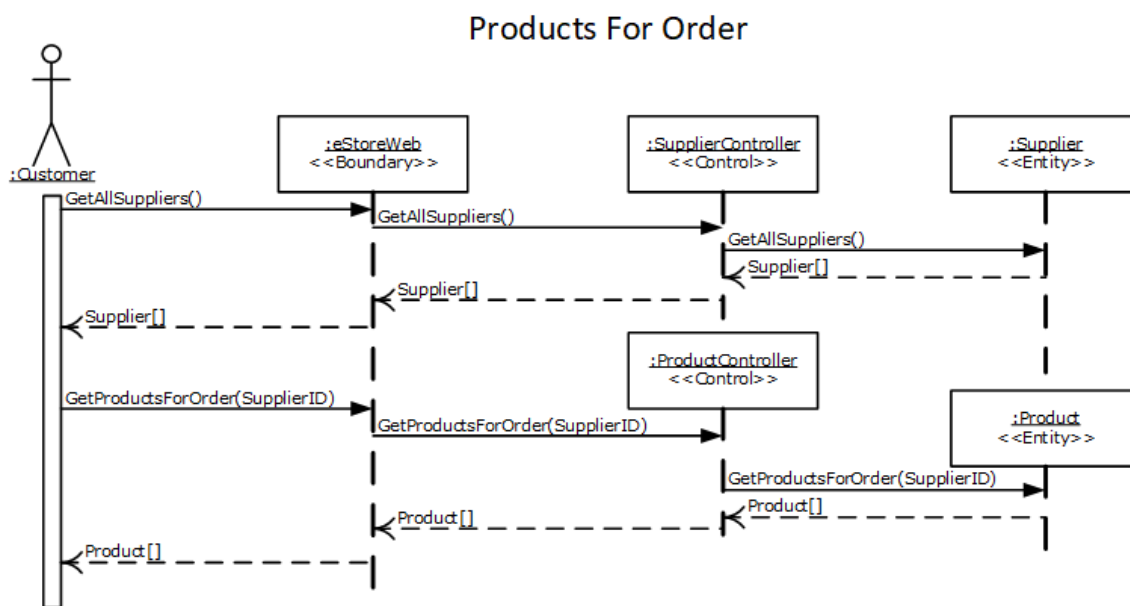


Figure 1: Products For Order Sequence Diagram

Web Form

We will also use the following prototype web form:

Products For Order

Supplier:

ID	Name	SKU	Cost	ROL	QOH	Ordered
1001	Apple	F-Apple-01	1.25	55	150	0
1002	Apple	F-Apple-02	1.35	45	140	0
1003	Apple	F-Apple-03	1.45	40	130	0

Figure 2: Products for Order Web Form

About the ListView Control

The ListView control is a very useful control. It can be customized in many ways (for this course, we will keep it simple). The ListView control is created using templates. There are 2 templates that **must** be used: **LayoutTemplate** and the **ItemTemplate**. There is one more template that we will need, the **EmptyDataTemplate** (the other templates created by Visual Studio can be deleted).

LayoutTemplate

This template is used as a design for all the data that will be displayed on the ListView control. In this course, we will be using a table-style layout so that its look will be like the GridView control.

ItemTemplate

The ItemTemplate is used to tell the ListView how to display the data that is being sent to it. It will be created as rows of a table.

EmptyDataTemplate

Like the GridView's EmptyDataTemplate, this template will display a message if there is no data sent to the ListView.

Setup

ProductForOrder

Add a new class to the **POCOs** folder of the **eStoreData** project and name the class **ProductForOrder.cs**:

Listing 1: ProductForOrder (namespace only)

```
namespace eStoreData.POCOs
{
    public class ProductForOrder
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string SKU { get; set; }
        public decimal Cost { get; set; }
        public int ROL { get; set; }
        public int QOH { get; set; }
        public int Ordered { get; set; }
    } //eoc
} //eon
```

This is like **ProductForSale.cs**, but with some new fields and some fields not present.

SupplierController

Add a new class to the **BLL** folder of the **eStoreSystem** project called **SupplierController.cs**. The code we need is shown below (notice how similar this is to the **CategoryController.cs** created earlier):

Listing 2: Supplier Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

#region Additional Namespaces
using System.ComponentModel;
using eStoreData.Entities;
using eStoreData.POCOs;
using eStoreSystem.DAL;
#endregion

namespace eStoreSystem.BLL
{
    [DataContract]
    public class SupplierController
    {
        [DataContractMethod(DataObjectMethodType.Select, false)]
        public List<SelectionList> GetAllSuppliers()
        {
            // Setup transaction area
            using (var context = new eStoreContext())
            {
                var results = from x in context.Suppliers
                              orderby x.SupplierName
                              select new SelectionList
                              {
                                  IDValue = x.SupplierID,
                                  DisplayText = x.SupplierName
                              };
                return results.ToList();
            }
        }
    }
}
```

Update ProductController

Previously we created a query in LINQPad before coding the method in a <<Control>> class. The LINQPad query used for the **ProductController**, with its results, is shown below:

```

var results = from x in Products
               where x.SupplierID == 1
               orderby x.ProductName
               select new
               {
                   ID = x.ProductID,
                   Name = x.ProductName,
                   SKU = x.SKU,
                   Cost = x.OrderCost,
                   ROL = x.ReOrderLevel,
                   QOH = x.QuantityOnHand,
                   Ordered = x.OnOrder
               };
results.Dump();

```

Results λ SQL IL

^ IOrderedQueryable<> (4 items)

ID	Name	SKU	Cost	ROL	QOH	Ordered
1	Apple	F-Apple-01	1.25	55	150	0
2	Apple	F-Apple-02	1.35	45	140	0
3	Apple	F-Apple-03	1.45	40	130	0
5	Banana	F-Banan-02	1.74	70	67	0
			5.79	210	487	0

Figure 3: GetProductsForOrder Query and Results

Open your existing **ProductController.cs** file and add the new method to this file below the **ProductForSale** method:

Listing 3: ProductController.cs New Method

```

[DataObjectMethod(DataObjectMethodType.Select, false)]
public List<ProductForOrder> GetProductsForOrder(int supplierID)
{
    // Setup transaction area
    using (var context = new eStoreContext())
    {
        var results = from x in context.Products
                      where x.SupplierID == supplierID
                      orderby x.ProductName
                      select new ProductForOrder
                      {
                          ID = x.ProductID,
                          Name = x.ProductName,
                          SKU = x.SKU,
                          Cost = x.OrderCost,
                          ROL = x.ReOrderLevel,
                          QOH = x.QuantityOnHand,
                          Ordered = x.OnOrder
                      };
        return results.ToList();
    }
} //end using
} //eom

```

BUILD!

Now, open Visual Studio. Fix any errors before proceeding. A successful **Build** should show something like:

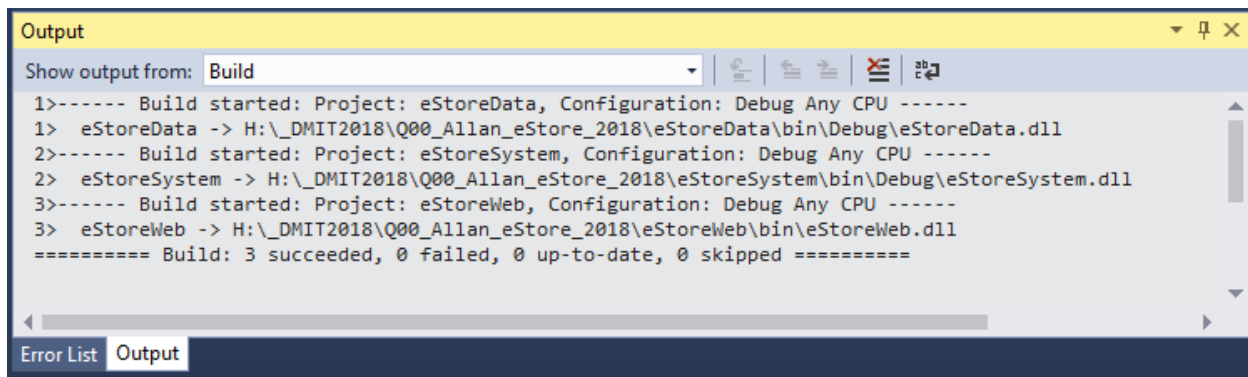


Figure 4: Successful Build

The **Build** is required as we have made modifications, and additions, to the class library projects. By doing the **Build**, the linked library files will be updated for the web site. If you forget to **Build** before coding the web form, you will **NOT** be able to use the new code we added above.

Web Form

Open the **PurchaseOrders.aspx** you created earlier and modify the code to be like that shown below (notice the similarities to **ShoppingCart.aspx**):

Listing 4: ProductsForSupplier.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"
CodeBehind="PurchaseOrders.aspx.cs" Inherits="eStoreWeb.Orders.PurchaseOrders" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">
    <div class="jumbotron">
        <h1>Products For Order</h1>
        <p>This web form will load a DropDownList of Suppliers which the Customer can
select from.
```

Once a Supplier is selected, a list of Products for the Supplier will be displayed.

[illegible]

There are two changes. One is the button styling, `CssClass="btn btn-primary"` (line 13): this just makes the button look better. The other change is the `asp:ListView ID="ProductListLV"`.

As with the **ShoppingCart.aspx** web form, we need to configure the **ObjectDataSource** controls, and the **DropDownList** control, before we can configure the control to display all the data.

SupplierListODS

This ODS control will connect to the **SupplierController**, and use the **GetAllSuppliers()** method:

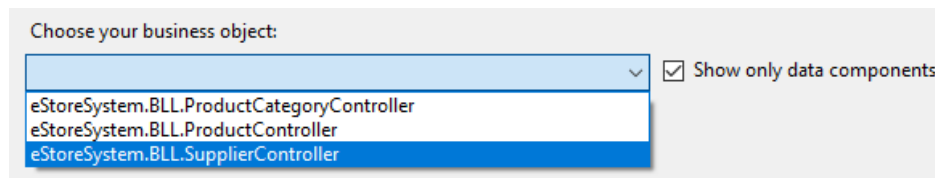


Figure 5: Business Object for SupplierListODS

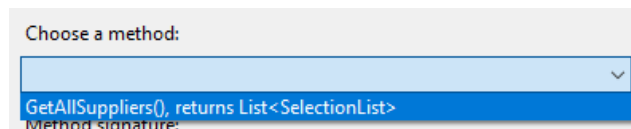


Figure 6: Method Selection for SupplierListODS

Listing 5: SupplierListODS Code

```
<asp:ObjectDataSource ID="SupplierListODS" runat="server"
    OldValuesParameterFormatString="original_{0}"
    SelectMethod="GetAllSuppliers"
    TypeName="eStoreSystem.BLL.SupplierController">
</asp:ObjectDataSource>
```

SupplierListDDL

Once the **SupplierListODS** control has its data source, the **SupplierListDDL** needs to have its data source set to the **SupplierListODS** control.

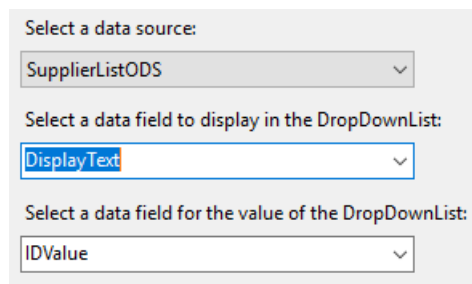


Figure 7: SupplierListDDL Data Source Selection

Additionally, you will need to add a manual list item for “Select Supplier”

Listing 6: SupplierListDDL Code

```
<asp:DropDownList ID="SupplierListDDL" runat="server"
    DataSourceID="SupplierListODS"
    DataTextField="DisplayText"
    DataValueField="IDValue"
    AppendDataBoundItems="true">
    <asp:ListItem Value="0">Select Supplier</asp:ListItem>
</asp:DropDownList>
```

ProductListODS

This ODS control will connect to the **ProductController**, and use the **GetProductsForSupplier(SupplierID)** method. The SupplierID parameter will come from the **SupplierListDDL** control.

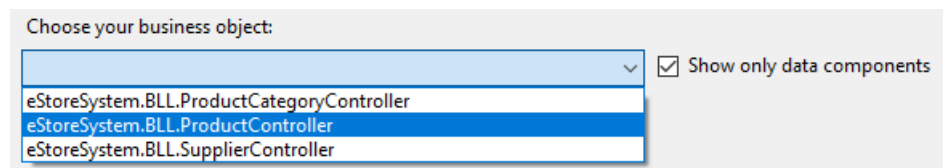


Figure 8: ProductListODS Business Object Selection

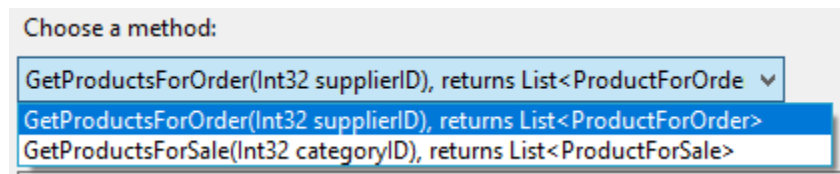


Figure 9: Method Selection for ProductListODS

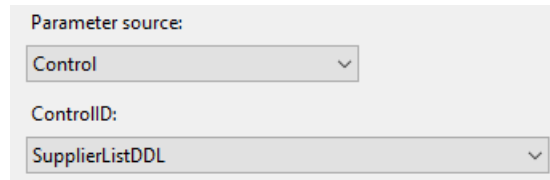


Figure 10: Parameter Source Selection

Listing 7: ProductListODS Code

```
<asp:ObjectDataSource ID="ProductListODS" runat="server"
    OldValuesParameterFormatString="original_{0}"
    SelectMethod="GetProductsForOrder"
    TypeName="eStoreSystem.BLL.ProductController">
    <SelectParameters>
        <asp:ControlParameter ControlID="SupplierListDDL"
            PropertyName="SelectedValue"
            Name="supplierID"
            Type="Int32">
        </asp:ControlParameter>
    </SelectParameters>
</asp:ObjectDataSource>
```

ListView – ProductListLV

The setup of a ListView control is a bit complex when coded manually. The good thing is that the ListView control can be automatically configured by selecting the Data Source.

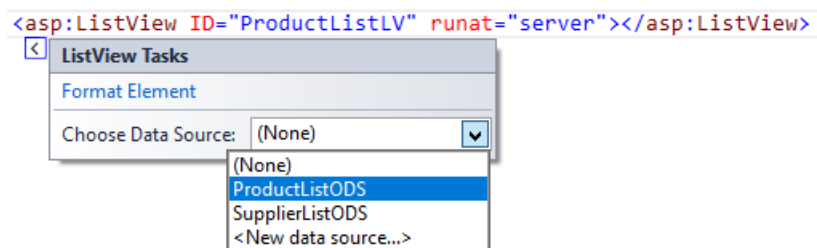


Figure 11: Configure Data Source for ListView

Once the Data Source has been selected, the next step is to refresh the schema to get the correct layout.

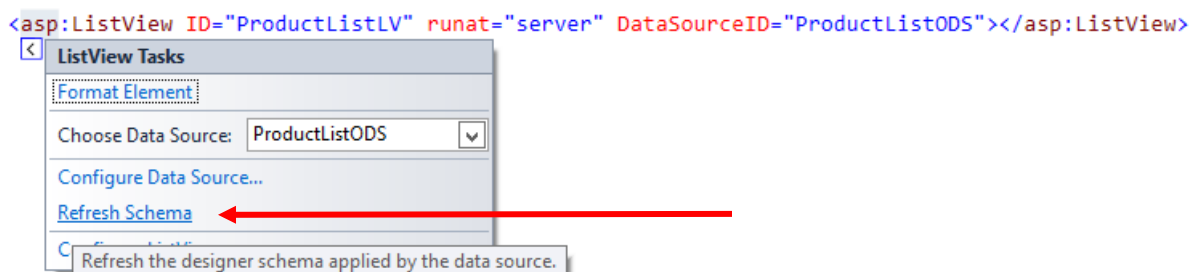


Figure 12: ListView with Data Source

Next, you will need to configure the ListView. Press the [Configure ListView...](#) link from the smart tag and configure the ListView as shown below:

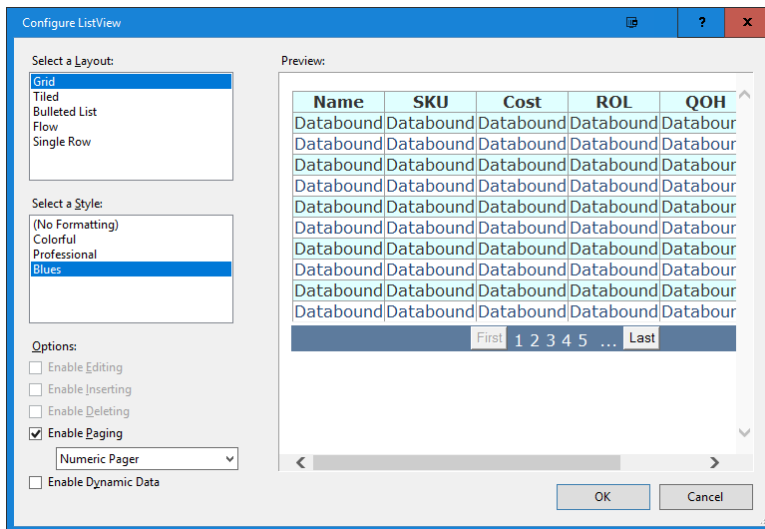


Figure 13: Configure ListView Wizard

Use the **Grid** layout option. Select a style that you want to use (Blues is used for this example). You can also enable paging, and can choose from Next/Previous or Numeric paging (Numeric paging will be used for our ListView controls). Once all the options are selected, press OK to close the wizard. Visual Studio will generate all the code for the ListView.

Examining the code created, you will see the following templates:

- AlternatingItemTemplate (optional template)
- EditItemTemplate (not needed for this example, **so delete this template**)
- EmptyDataTemplate (optional template, but required for this course)
- InsertItemTemplate (not needed for this example, **so delete this template**)
- ItemTemplate (required template)
- LayoutTemplate (required template)
- SelectedItemTemplate (not needed for this example, **so delete this template**)

LayoutTemplate

Listing 8: ProductListLV - LayoutTemplate Code

```
<LayoutTemplate>
    <table runat="server">
        <tr runat="server">
            <td runat="server">
                <table runat="server" id="itemPlaceholderContainer" style="background-
color: #FFFFFF; border-collapse: collapse; border-color: #999999; border-style: none;
border-width: 1px; font-family: Verdana, Arial, Helvetica, sans-serif;" border="1">
                    <tr runat="server" style="background-color: #E0FFFF; color:
#333333;">
                        <th runat="server">ID</th>
                        <th runat="server">Name</th>
                        <th runat="server">SKU</th>
                        <th runat="server">Cost</th>
                        <th runat="server">ROL</th>
                        <th runat="server">QOH</th>
                        <th runat="server">Ordered</th>
                    </tr>
                    <tr runat="server" id="itemPlaceholder"></tr>
                </table>
            </td>
        </tr>
        <tr runat="server">
            <td runat="server" style="text-align: center; background-color: #5D7B9D;
font-family: Verdana, Arial, Helvetica, sans-serif; color: #FFFFFF">
                <asp:DataPager runat="server" ID="DataPager1">
                    <Fields>
                        <asp:NextPreviousPagerField ButtonType="Button"
ShowFirstPageButton="True" ShowNextPageButton="False"
ShowPreviousPageButton="False"></asp:NextPreviousPagerField>
                        <asp:NumericPagerField></asp:NumericPagerField>
                        <asp:NextPreviousPagerField ButtonType="Button"
ShowLastPageButton="True" ShowNextPageButton="False"
ShowPreviousPageButton="False"></asp:NextPreviousPagerField>
                    </Fields>
                </asp:DataPager>
            </td>
        </tr>
    </table>
</LayoutTemplate>
```

```

        </tr>
    </table>
</LayoutTemplate>

```

Notice the table layout, and that the `table`, `tr`, and `th` tags have a `runat="server"`. This type of `LayoutTemplate` is one that you will use in this course. The only thing you need to change is the header text.

ItemTemplate

Each column of the `LayoutTemplate` will have a corresponding column in the `ItemTemplate`. Each row of the `ItemTemplate` will be formatted the same. Each cell of a row can be formatted differently.

Listing 9: ProductListLV – Original ItemTemplate Code

```

<ItemTemplate>
    <tr style="background-color: #E0FFFF; color: #333333;">
        <td>
            <asp:Label Text='<%# Eval("ID") %>' runat="server" ID="IDLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("Name") %>' runat="server" ID="NameLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("SKU") %>' runat="server" ID="SKULabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("Cost") %>' runat="server" ID="CostLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("ROL") %>' runat="server" ID="ROLLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("QOH") %>' runat="server" ID="QOHLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("Ordered") %>' runat="server" ID="OrderedLabel"
    /></td>
    </tr>
</ItemTemplate>

```

First, notice that there is no `runat="server"` for any of the `tr` or `td` tags. These are not needed here as we have the

```

<tr runat="server" id="itemPlaceholderContainer"></tr>

```

in the `LayoutTemplate`. The `itemPlaceholderContainer` in the `LayoutTemplate` will automatically add the `runat="server"`.

Another thing to notice is the scripting to bind the data to the controls. i.e.:

```

<asp:Label Text='<%# Eval("Name") %>' runat="server" ID="NameLabel" /></td>

```

The keyword, `Eval`, does a one-way (read only) binding of the data to the control.

In this example, we will use some different controls for some of the cells. For each of the numeric data fields we will use `TextBox` controls instead of `Label` controls. Additionally, we

will format the **Cost** field to display the value as a numeric currency value. The **ItemTemplate** will look like the code below when completed:

Listing 10: *ItemTemplate Modified*

```
<ItemTemplate>
    <tr style="background-color: #E0FFFF; color: #333333;">
        <td>
            <asp:Label Text='<%# Eval("ID") %>' runat="server" ID="IDLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("Name") %>' runat="server" ID="NameLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("SKU") %>' runat="server" ID="SKULabel" /></td>
        <td>
            <asp:TextBox Text='<%# string.Format("{0:0.00}", Eval("Cost")) %>'
runat="server" ID="CostTextBox" Width="50" /></td>
        <td>
            <asp:TextBox Text='<%# Eval("ROL") %>' runat="server" ID="ROLTextBox"
Width="50"/></td>
        <td>
            <asp:TextBox Text='<%# Eval("QOH") %>' runat="server" ID="QOHTextBox"
Width="50" /></td>
        <td>
            <asp:TextBox Text='<%# Eval("Ordered") %>' runat="server"
ID="OrderedTextBox" Width="50" /></td>
    </tr>
</ItemTemplate>
```

The `{0:0.00}` is like the parameterized string output we saw in lesson 1.3.0. The first `0` is for the parameter `Eval("Cost")`. The second part, `0.00`, is for formatting the number as currency (2 decimal places).

AlternatingItemTemplate

The only difference between the **ItemTemplate** and the **AlternatingItemTemplate** is the styling. You will need to modify this template to be:

Listing 11: *AlternatingItemTemplate Code*

```
<AlternatingItemTemplate>
    <tr style="background-color: #FFFFFF; color: #284775;">
        <td>
            <asp:Label Text='<%# Eval("ID") %>' runat="server" ID="IDLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("Name") %>' runat="server" ID="NameLabel" /></td>
        <td>
            <asp:Label Text='<%# Eval("SKU") %>' runat="server" ID="SKULabel" /></td>
        <td>
            <asp:TextBox Text='<%# string.Format("{0:0.00}", Eval("Cost")) %>'
runat="server" ID="CostTextBox" Width="50" /></td>
        <td>
            <asp:TextBox Text='<%# Eval("ROL") %>' runat="server" ID="ROLTextBox"
Width="50"/></td>
        <td>
            <asp:TextBox Text='<%# Eval("QOH") %>' runat="server" ID="QOHTextBox"
Width="50" /></td>
        <td>
            <asp:TextBox Text='<%# Eval("Ordered") %>' runat="server" ID="OrderedTextBox"
Width="50" /></td>
```

```

</tr>
</AlternatingItemTemplate>

```

EmptyDataTemplate

Listing 12: ProductListLV - EmptyDataTemplate Code

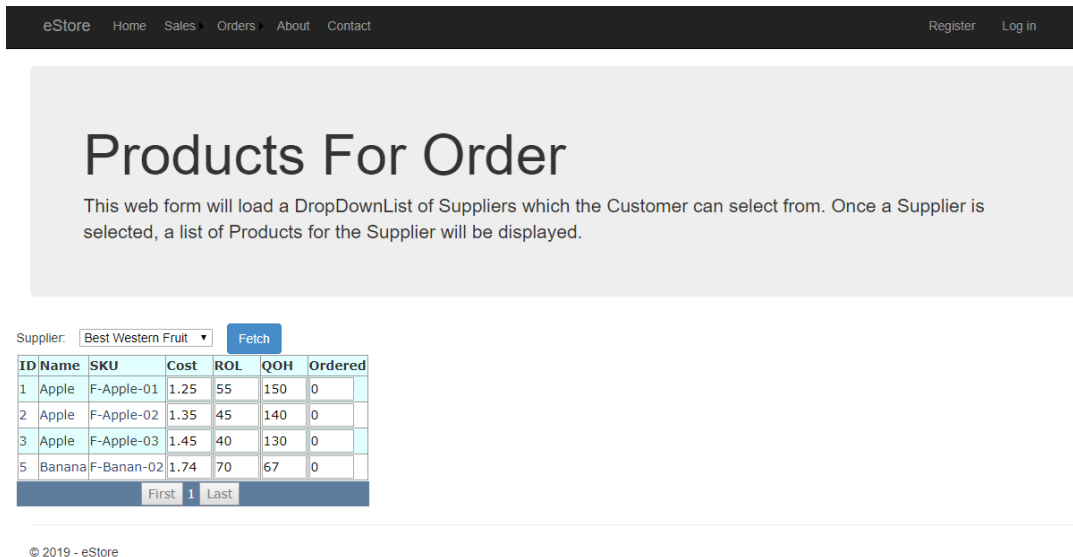
```

<EmptyDataTemplate>
    <table runat="server" style="background-color: #FFFFFF; border-collapse: collapse;
border-color: #999999; border-style: none; border-width: 1px;">
        <tr>
            <td>No data was returned.</td>
        </tr>
    </table>
</EmptyDataTemplate>

```

TEST!

Run your web application. Test this new web form with different Suppliers. You should get output like:



The screenshot shows a web application interface for ordering products. At the top is a navigation bar with links: eStore, Home, Sales, Orders, About, Contact, Register, and Log in. The main content area has a heading 'Products For Order' and a description: 'This web form will load a DropDownList of Suppliers which the Customer can select from. Once a Supplier is selected, a list of Products for the Supplier will be displayed.' Below this is a form with a 'Supplier:' label, a dropdown menu showing 'Best Western Fruit', and a 'Fetch' button. Under the form is a table with the following data:

ID	Name	SKU	Cost	ROL	QOH	Ordered
1	Apple	F-Apple-01	1.25	55	150	0
2	Apple	F-Apple-02	1.35	45	140	0
3	Apple	F-Apple-03	1.45	40	130	0
5	Banana	F-Banan-02	1.74	70	67	0

At the bottom of the table, there are pagination controls: 'First', '1', and 'Last'.

Figure 14: Test Output

Exercise

Complete Exercise 4.3.1.