The View

Let us now look at the View. How does the framework know which View or aspx page to call? Remember that we passed the value "Index" in the action parameter (in the default route in the `global.asax.cs` file), so the `Index.aspx` will get called. Here is the code-behind of `Index.apsx`:

```
public partial class Index : ViewPage
    {
    }
```

There is absolutely no code here, which is a very important characteristic of the MVC design. The GUI should have no logical or data fetching code. Note that the `Index` class is derived from the `ViewPage` class. Using this `ViewPage` class, we can access all of the items in the `ViewData` dictionary that were set in the controller's `Index()` method and passed on to the View. Here is how we are accessing the `ViewData` in HTML:

```
<asp:Content ID="indexContent" ContentPlaceHolderID="MainContent"
                runat="server">
    <h2><%= Html.Encode(ViewData["Message"]) %></h2>
    <p>
        To learn more about ASP.NET MVC visit <a href="http://asp.net/
mvc" title="ASP.NET MVC Website">http://asp.net/mvc</a>.
    </p>
</asp:Content>
```

We can directly access the `ViewData` dictionary in HTML. Now that we have seen how MVC works, we will create a new page to learn how to show data using a custom DAL and strongly typed objects, instead of the `ViewData` dictionary. Our example page will show a list of all the customers.

The Model

We will use the 5-Tier solution we created in the previous chapter and change the GUI layer to make it follow the MVC design using the ASP.NET MVC framework. Open the solution we created in the previous chapter and delete the ASP.NET web project from it. The solution will then only contain `5Tier.BL`, `5Tier.DAL` and `5Tier.Common` projects.

Right click the solution in VS, and select **Add New Project**, and then select **ASP.NET MVC Web Application** from the dialog box. Name this new web project as `Chapter05.MVC`. This web project will be the new MVC based UI tier of our OMS application in this chapter.

The `Customer.cs` and `CustomerCollection.cs` class files in the business tier (5Tier. Business class library) will be the Model in our MVC application. To show a list of customers, the `CustomerCollection` class simply calls the `FindCustomer()` method in `CustomerDAL.cs`. We have already seen these classes in action in the previous chapter. So we can use an n-tier architecture in an MVC application, hence this shows that MVC and n-tier are not mutually exclusive options while considering the application architecture of your web application. Both actually compliment each other.

We can also create a utility class named `CustomerViewData` to transfer the Model objects to the View. There are multiple ways to pass- in the Model to the View through the Controller, and creating `ViewData` classes is one of them. Here is the `CustomerViewData` class created in the `CustomerComtroller.cs` file in the `Chapter05.MVC` web project:

```
#region ViewData
        /// <summary>
        /// Class used for transferring data to the View
        /// </summary>
        public class CustomerViewData
        {
            public CustomerViewData() { }
            public CustomerViewData(Collection<Customer> customers)
            {
                this.customers = customers;
            }
            public Collection<Customer> customers;
            public Customer customer;
        }
        #endregion
```

Notice that this `ViewData` class is simply wrapping the business object inside it so that we can use this class in the UI layer instead of directly passing and manipulating domain objects.

Wiring Controller, Model, and View

We will now create routes in the `global.asax` file under the existing home page route as follows:

```
routes.MapRoute(
"Customer", "Customer/{action}/{id}", new {
 controller = "Customer", action = "Show", id="" } );
```