[Home](#)[About Us](#)[ASP.NET Online Courses](#)[Yoga Articles](#)[Contact Us](#)

#### Online Courses

[ASP.NET Core](#)[Design Patterns](#)[Meditation](#)

#### Categories

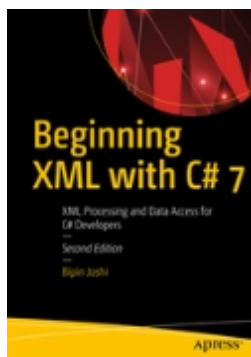
[ASP.NET](#)[ASP.NET MVC](#)[ASP.NET Core](#)[jQuery](#)[HTML5](#)[Web Services](#)[Data Access](#)[.Net Framework](#)

**Learn ASP.NET Core 3.1 with mini project : MVC, Razor Pages, Web API, Entity Framework Core, and Blazor. Hands-on online training course starting in June 2020. Click [here](#) to know more.**

## Load ASP.NET MVC Partial Views Dynamically Using JQuery

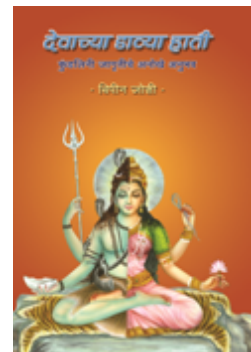
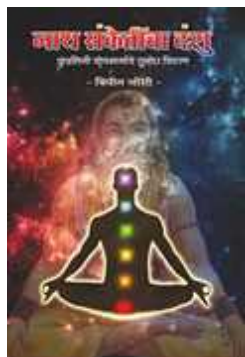
Most of the times ASP.NET MVC views are rendered as a result of user navigating to some action. For example, when a user navigates to /home/index in the browser (either through address bar or through a hyperlink), ASP.NET MVC executes the action method and usually returns a view to the browser. This means each view is rendered as a result of a full GET or POST request. At times, however, you may want to load views dynamically through Ajax. This way you can render contents of a view without full page refresh.

Consider the following page:

[Windows Forms](#)[Book Reviews](#)[Projects](#)[Miscellaneous](#)[Thoughts & Musings](#)[Videos](#)[Search Articles](#)[Article Updates](#)[RSS Feed](#)

Customer ID	Company Name	Actions	
ALFKI	Alfreds Futterkiste	Customer Details	Order Details
ANATR	Ana Trujillo Emparedados y helados	Customer Details	Order Details
ANTON	Antonio Moreno Taquería	Customer Details	Order Details
AROUT	Around the Horn	Customer Details	Order Details

The above page consists of a table that lists customers from the Customers table of Northwind database. Each customer row has two buttons - Customer Details and Order Details. Clicking on the respective button should display customer details and order details from the database. Without Ajax you would have submitted the page back to the server and then returned a view with the corresponding details. Using Ajax you can display the details without causing any postback to the server. This is shown below:



Index

localhost:49258/Home/Index

Order ID	Order Date	Shipping Date	Shipped To
10643	8/25/1997 12:00:00 AM	9/2/1997 12:00:00 AM	Germany
10692	10/3/1997 12:00:00 AM	10/13/1997 12:00:00 AM	Germany
10702	10/13/1997 12:00:00 AM	10/21/1997 12:00:00 AM	Germany
10835	1/15/1998 12:00:00 AM	1/21/1998 12:00:00 AM	Germany
10952	3/16/1998 12:00:00 AM	3/24/1998 12:00:00 AM	Germany
11011	4/9/1998 12:00:00 AM	4/13/1998 12:00:00 AM	Germany

Customer ID	Company Name	Actions	
ALFKI	Alfreds Futterkiste	Customer Details	Order Details
ANATR	Ana Trujillo Emparedados y helados	Customer Details	Order Details

As you can see the above figure shows order details for CustomerID ALFKI above the customers table. These details are fetched via Ajax request.

While displaying data through Ajax request you have two options:

- Fetch raw data from the server and embed it in HTML markup on the client side



# Build apps that matter

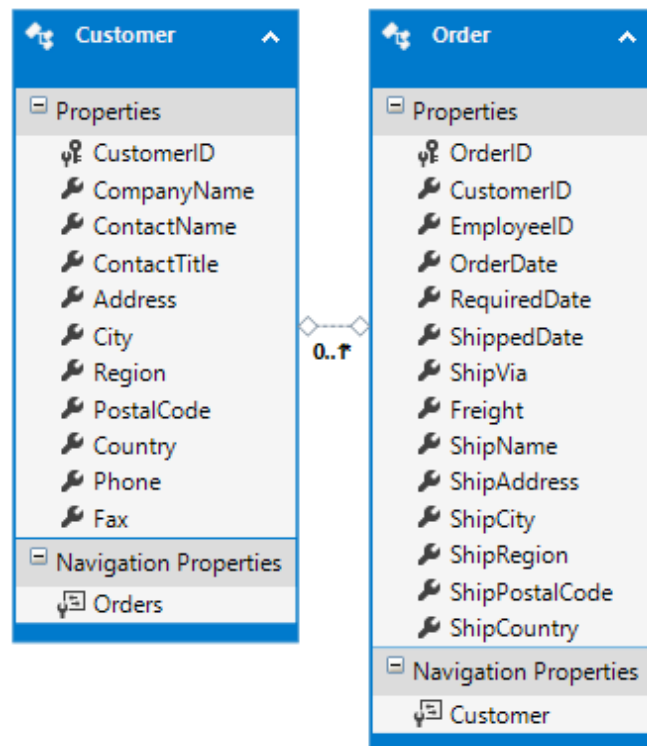
GET THE EBOOK



## ■ Fetch HTML markup with data embedded from the server

Although the choice of the approach depends on a situation, it can be said that the former approach is suitable to make calls to Web API or when HTML display is dynamically decided by the client script. The later approach is suitable when ASP.NET MVC strongly typed views (or partial views) are being used to render the UI. In this example we will be using the later approach.

To develop this example, create a new ASP.NET MVC application based on the Empty template. Then add ADO.NET Entity Data Model for Customers and Orders tables of Northwind database. The Customer and Order entities are shown below:



Next, add HomeController and write the Index() action method as shown below:

**AWS + C3.ai****50x**  
less code**25x**  
faster  
development**1/10**  
time to deploy**Get the report**

```
public ActionResult Index()
{
    using (NorthwindEntities db = new NorthwindEntities())
    {
        List<Customer> model = db.Customers.ToList();
        return View(model);
    }
}
```

The Index() action simply retrieves all the Customer entities from the Customers DbSet and passes them to the Index view.

Now, add another action method - GetView() - to the HomeController as shown below:

```
public ActionResult GetView(string customerID, string viewName)
{
    object model = null;
    if (viewName == "CustomerDetails")
    {
        using (NorthwindEntities db = new NorthwindEntities())
        {
            model = db.Customers.Find(customerID);
        }
    }
    if (viewName == "OrderDetails")
    {
        using (NorthwindEntities db = new NorthwindEntities())
```

```
{  
    model = db.Orders.Where(o => o.CustomerID == customerId)  
        .OrderBy(o => o.OrderID).ToList();  
}  
}  
return PartialView(viewName, model);  
}
```

The `GetView()` action method accepts two parameters - `customerId` and `viewName`. These two parameters are passed through an Ajax request. Depending on the `viewName` parameter either `CustomerDetails` partial view is returned to the caller or `OrderDetails` partial view is returned. These two view need model in the form of a `Customer` object and a `List` of `Order` entities respectively. That's why model variable is declared as object. Once model variable is populated the partial view name and the model is passed to the `PartialView()` method. Here, we used partial views because the HTML output is to be inserted in an existing page through Ajax.

Next, add one view (`Index.cshtml`) and two partial views (`CustomerDetails.cshtml` and `OrderDetails.cshtml`) to the Home sub-folder of Views folder.

Add the following markup to the `CustomerDetails.cshtml` partial view:

```
@model MVCViewsThroughAjax.Models.Customer  
  
<table border="1" cellpadding="10">  
    <tr>  
        <td>Customer ID :</td>  
        <td>@Model.CustomerID</td>
```

```
</tr>
<tr>
    <td>Company Name :</td>
    <td>@Model.CompanyName</td>
</tr>
<tr>
    <td>Contact Name :</td>
    <td>@Model.ContactName</td>
</tr>
<tr>
    <td>Country :</td>
    <td>@Model.Country</td>
</tr>
</table>
```

The above markup is quite straightforward. The CustomerDetails partial view simply displays CustomerID, CompanyName, ContactName and Country of a Customer in a table.

Now add the following markup to the OrderDetails.cshtml partial page:

```
@model List<MVCViewsThroughAjax.Models.Order>

<table border="1" cellpadding="10">
    <tr>
        <th>Order ID</th>
        <th>Order Date</th>
        <th>Shipping Date</th>
```

```
        <th>Shipped To</th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>@item.OrderID</td>
            <td>@item.OrderDate</td>
            <td>@item.ShippedDate</td>
            <td>@item.ShipCountry</td>
        </tr>
    }
</table>
```

The above markup iterates through the List of Order entities and renders a table with four columns - OrderID, OrderDate, ShippedDate and ShipCountry.

Now, add the following markup to the Index view:

```
@model List<MVCViewsThroughAjax.Models.Customer>

...
<html>
<head>
...
</head>
<body>
    <div id="viewPlaceHolder"></div>
```



```
<br /><br />
<table border="1" cellpadding="10">
  <tr>
    <th>Customer ID</th>
    <th>Company Name</th>
    <th colspan="2">Actions</th>
  </tr>
  @foreach(var item in Model)
  {
    <tr>
      <td>@item.CustomerID</td>
      <td>@item.CompanyName</td>
      <td><input type="button" class="customerDetails"
        value="Customer Details" /></td>
      <td><input type="button" class="orderDetails"
        value="Order Details" /></td>
    </tr>
  }
</table>
</body>
</html>
```

The Index view receives a List of Customer entities as its model and renders a table with CustomerID, CompanyName and two buttons - Customer Details and Order Details.

Now comes the important part - making Ajax calls to display customer details and order details. Noticed the <div> at the beginning of the body section? The viewPlaceHolder is where the output of

CustomerDetails.cshtml and OrderDetails.cshtml will be loaded. To do so we will use load() method of jQuery. Here is how that can be done:

```
$(document).ready(function () {

    $(".customerDetails").click(function (evt) {
        var cell=$(evt.target).closest("tr").children().first();
        var custID=cell.text();
        $("#viewPlaceholder").load("/home/getview",
            { customerID: custID, viewName: "CustomerDetails" });
    });

    $(".orderDetails").click(function (evt) {
        var cell = $(evt.target).closest("tr").children().first();
        var custID = cell.text();
        $("#viewPlaceholder").load("/home/getview",
            { customerID: custID, viewName: "OrderDetails" });
    });

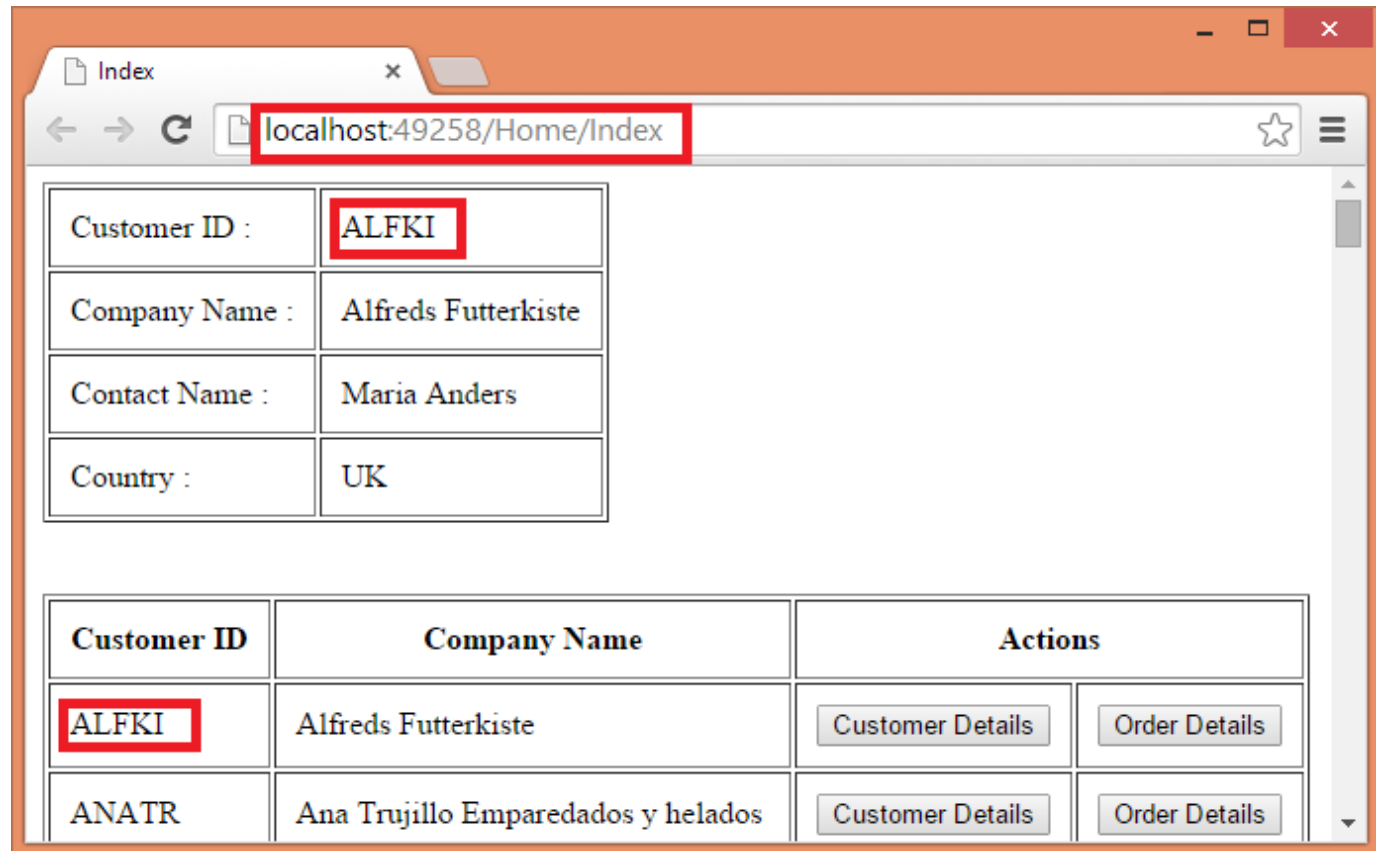
});
```

Recollect that Customer Details and Order Details buttons have assigned CSS class of customerDetails and orderDetails respectively. The above jQuery code uses class selector to wire click event handlers to the respective buttons. Inside the click event handler of Customer Details button, the code retrieves the CustomerID from the table row. This is done using closest(), children() and first() methods. The CustomerID is stored in custID variable. Then load() method is called on viewPlaceholder <div>. The first parameter of the load() method is the URL that will be requested through an Ajax request. The second parameter is a JavaScript object that supplies the data needed by the requested URL. In our example,

GetView() action method needs two parameters - customerId and viewName. Hence the object has customerId and viewName properties. The customerId property is set to custID variable and viewName is set to CustomerDetails.

The click event handler of Order Details is similar but loads OrderDetails partial view.

That's it! You can now run the application and try clicking on both the buttons. The following figure shows customer details loaded successfully.



Notice that through out the application run the URL shown in the browser address bar remains unchanged indicating that Ajax requests are being made to display customer details and order details.

In the above example Ajax requests were made to /home/getview action. A user can also enter this URL in the browser's address bar producing undesirable results. As a precaution you can check the customerId and viewName parameters inside the GetView() action method (not shown in the above code). If these parameters are empty or contain invalid values you can throw an exception.

**Bipin Joshi** is an independent software consultant, trainer, author, yoga mentor, and meditation teacher. He has been programming, meditating, and teaching for 24+ years. He conducts instructor-led **online training courses** in ASP.NET family of technologies for individuals and small groups. He is a published author and has authored or co-authored books for Apress and Wrox press. Having embraced the Yoga way of life he also teaches **Ajapa Yoga** to interested individuals. To know more about him click [here](#).

Get connected : [Facebook](#) [Twitter](#) [LinkedIn](#) [YouTube](#)

**Posted On : 27 October 2014**



**Tags : ASP.NET MVC C# jQuery JavaScript**

#### Popular from this category

[Load ASP.NET MVC Partial Views Dynamically Using jQuery](#)

[Model Binding to List of Objects in ASP.NET MVC](#)

**[Perform List, Insert, Update and Delete in a Single View in ASP.NET MVC](#)**

**[Four Ways of Handling Multiple Submit Buttons in ASP.NET MVC](#)**

**[Generating Short URLs using Google URL Shortener API](#)**

**[Passing data from one controller to another in ASP.NET MVC](#)**

**[Display image from byte array in ASP.NET MVC](#)**

**[Creating Cascading DropDownLists using ASP.NET MVC 4 and jQuery](#)**

**[Using GridView and DetailsView in ASP.NET MVC - Part 1](#)**

**[Understanding ViewData, ViewBag and TempData](#)**

**Copyright © binaryintellect.net. All rights reserved.**

**[Read Copyright & Legal Terms of Use](#)**

© PROTECTED BY COPYSCAPE

**Hosted by DiscountASP.net**

