



**CODE
PROJECT**
For those who code

VMware Tanzu Application Service
on Kubernetes is now in beta.

LEARN MORE



VMware Tanzu

articles

quick answers

discussions

features

community

help

Search for articles, questions, tips



Articles » Web Development » ASP.NET » Howto



10 ways to Bind Multiple Models on a View in MVC



VijayRana

26 Jun 2016 CPOL

Rate this:  4.92 (72 votes)

10 ways to Bind Multiple Models on a View in MVC

Introduction

In MVC we cannot use multiple model tag on a view. But Many times we need to pass multiple models from controller to view or we want to show data from multiple model on a view. For example we want to show a Blog and List of Comments of this Blog on a Single View or suppose if we want to show a Category and List Of Products related to this category on a View. So in such situations we need to pass multiple models to a Single View . We have many solutions of this problem. So in this article we will see different approaches to Bind Multiple Models on a View.

Problem Statement

We have two models named Blog and Comment and we want to create a view where we will show Blog Title and Blog Description and List of Comments related to the Blog.

Problem while binding two Models

If we will try to Bind Blog and Comment Model on a View as below then at run time we will get an exception that we can not Bind Multiple Models on a View.

```
@model MultipleModels.Models.BlogModel
@model MultipleModels.Models.CommentModel
<h2>Blog Detail</h2>
```

And the exception

Parser Error

Description: An error occurred during the parsing of a resource required to service this request. Please review the following specific parse error details and modify your source file appropriately.

Parser Error Message: Only one 'model' statement is allowed in a file.

Source Error:

```
Line 1: @model MultipleModels.Models.BlogModel
Line 2: @model MultipleModels.Models.CommentModel
Line 3: <h2>Blog Detail</h2>
Line 4: <table style="border:solid 2px;">
```

So In MVC we need some approach through which we can achieve this and luckily i am giving 10 approaches to achieve the same. I am not an expert, I'm sure I'll make mistakes or go against best practices several times. I will not discuss the pros and cons of different approaches. My intent is to show different approaches through which we can solve this problem.

Different ways to achieve

we will see below 10 ways to bind multiple models on a single view

1. View Model
2. View Bag
3. View Data
4. Temp Data
5. Session
6. Dynamic
7. Tuples
8. Render Action

9. JSON

10. Navigation Properties

We will see all these approaches with the help of an example (Solution of above problem statement). So in our problem statement we want to display a Blog and List of Comments on a view.

Below are the main players of this article

1. Blog Model
2. Comment Model
3. GetBlog() : In this method where i will create dummy data for Blog.
4. GetComments() : In this method where i will create dummy data for comments.
5. A Controller (I will name it BlogCommentCotroller)
6. View

I created a project named MultipleModels and now i will create all the required elements before starting different approaches.

1. Blog Model :- Create a Model named BlogModel . Right click on Models Folder and add a class named **BlogModel** . I created below BlogModel class in Models folder.

[Hide](#) [Copy Code](#)

```
public class BlogModel
{
    public int BlogID { get; set; }
    public string BlogTitle { get; set; }
    public string BlogDescription { get; set; }
    public DateTime CreatedDate { get; set; }
}
```

2. Comment Model :- Right click on Models folder and add another class named **CommentModel** as below

[Hide](#) [Copy Code](#)

```
public class CommentModel
{
    public int CommentID { get; set; }
    public string Comment { get; set; }
    public string CommentedBy { get; set; }
    public int BlogID { get; set; }
}
```

1. View Model

ViewModel is a class which contains the properties which are represented in view or represents the data that you want to display on your view/page.

Suppose if we want to create a View where we want to display a Blog and List of Comments then we will create our View Model (BlogCommentViewModel.cs) .Create a folder named ViewModel in your project and right click on that folder and create a class file named "BlogCommentViewModel.cs"

[Hide](#) [Copy Code](#)

```
public class BlogCommentViewModel
{
    public BlogModel Blog { get; set; }

    public List<CommentModel> Comments { get; set; }
}
```

In above ViewModel (**BlogCommentViewModel**) we created a property named **Blog** of **BlogModel** Type because we want to display a single Blog Detail on our View/Page and a second property named **Comments** of **List of CommentModel** type because we want to display the list of comments associated to the Blog.

Note : We can place the viewmodel (**BlogCommentViewModel**) in any folder in our project but i created a folder named ViewModel where i will place all the viewmodels created for this project.

Its time to create a controller named BlogController and create a action method inside named GetBlogComment as below

[Hide](#) [Copy Code](#)

```
public class BlogController : Controller
{
    public ActionResult GetBlogComment()
    {
        return View();
    }
}
```

Now first create a object of view Model and then set Blog and Comments properties as below. I created two functions for Blog (**GetBlogModel()**) and Comment (**GetCommentModel()**) dummy data.

[Hide](#) [Shrink ▲](#) [Copy Code](#)

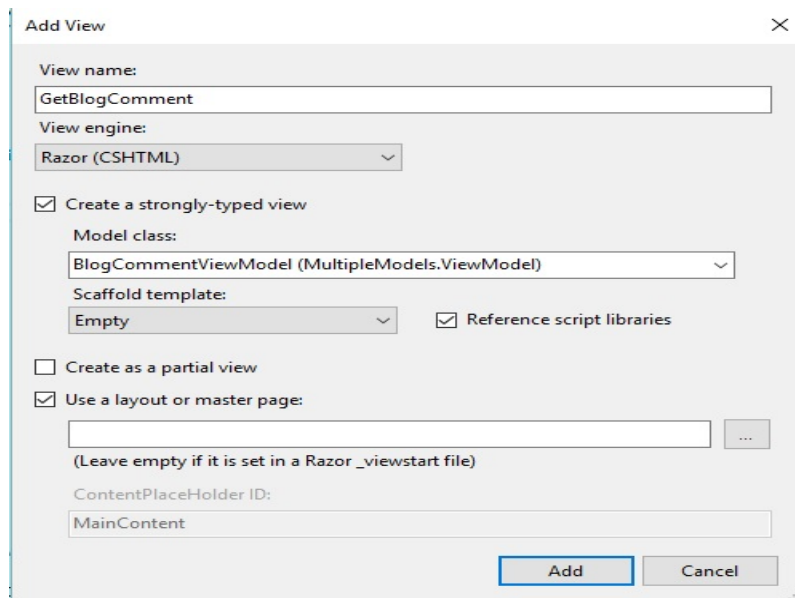
```
public class BlogController : Controller
{
    public ActionResult GetBlogComment()
    {
        BlogCommentViewModel BCVM = new BlogCommentViewModel();
        BCVM.Blog = GetBlogModel();
        BCVM.Comments = GetCommentModel();
        return View(BCVM);
    }

    public BlogModel GetBlogModel()
    {
        BlogModel bModel = new BlogModel() { BlogID = 1,
                                              BlogTitle = "MVC Blog",
}
```

```
        BlogDescription = "This is MVC Blog",  
        CreatedDate = DateTime.Now };  
  
    return bModel;  
}  
  
public List GetCommentModel()  
{  
    List cModel = new List();  
  
    cModel.Add(new CommentModel() { BlogID = 1, CommentID = 1, Comment = "Good One", CommentedBy = "Vijay" });  
    cModel.Add(new CommentModel() { BlogID = 1, CommentID = 2, Comment = "Nice", CommentedBy = "Nishant" });  
    cModel.Add(new CommentModel() { BlogID = 1, CommentID = 2, Comment = "Perfect", CommentedBy = "Saurabh" });  
    return cModel;  
}  
}
```

Note : Don't forget to add Models and ViewModels namespaces.

Now right click on GetBlogComment() action method and select Add View and while creating view select "Create a Strongly-Typed view" and select your view model named 'BlogCommentViewModel' and select "empty" in scaffold Template as below



Now change your created view (GetBlogComment.cshtml) as below

```

@model MultipleModels.ViewModel.BlogCommentViewModel
<h2>Blog Detail</h2>
<table style="border:solid 2px;">
<tr>
<td>
Blog Title
</td>
<td>
@Html.DisplayFor(m => m.Blog.BlogTitle) <br />
</td>
</tr>
<tr>
<td>
Blog Description
</td>
<td>
@Html.DisplayFor(m => m.Blog.BlogDescription) <br />
</td>
</tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
@foreach (var comment in Model.Comments)
{
<tr>
<td>
Comment
</td>
<td>
@comment.Comment ( By @comment.CommentedBy)
</td>
</tr>
}
</table>

```

I added some table css also in Style.css

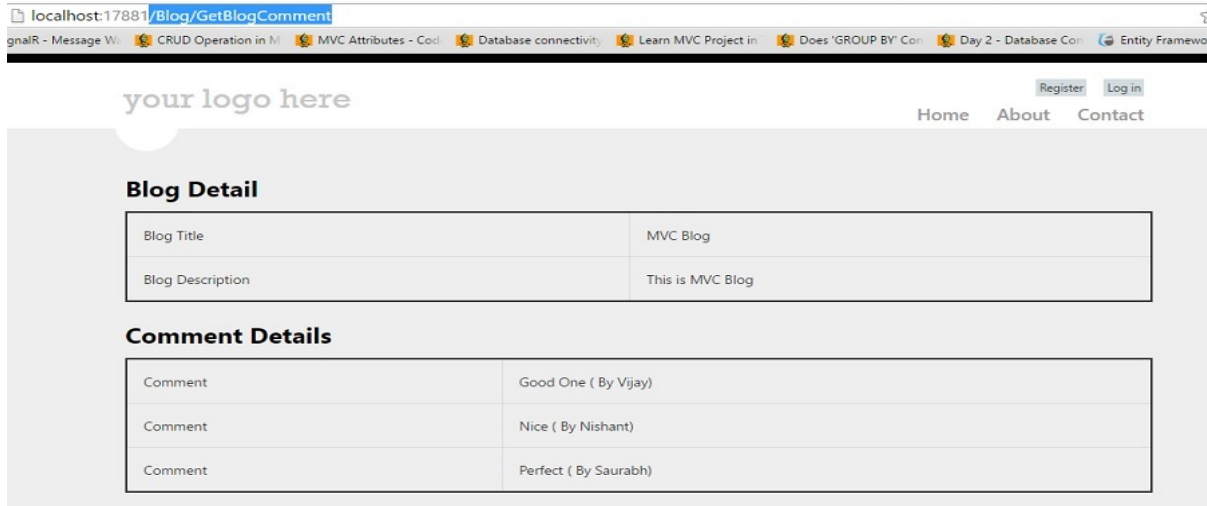
Hide Copy Code

```

table, td, th {
    border: 1px solid #ddd;
    text-align: left;
}
table {
    border-collapse: collapse;
    width: 100%;
}
th, td {
    padding: 15px;
}

```

Now run the application and type /Blog/GetBlogComment(ControllerName/ActionName) in browser and we will get output as below



2. View Bag

We use view Bag to pass data from controller to view. ViewBag use the property that takes advantage of the new dynamic features in C# 4.0. There is no typecasting needed in case of ViewBag. So we can take help of ViewBag to pass multiple data from controller to View. ViewBag is a property of ControllerBase class. View Bag scope is only during the current request.

To pass multiple model we will create two view bag in our action method as below.

[Hide](#) [Copy Code](#)

```
public ActionResult GetBlogComment()
{
    ViewBag.Blog = GetBlogModel();
    ViewBag.Comments = GetCommentModel();
    return View();
}
```

Now change your view (GetBlogComment.cshtml) according to view Bag as below

```

<h2>Blog Detail</h2>
<table style="border:solid 2px;">
  <tr>
    <td>
      Blog Title
    </td>
    <td>
      @ViewBag.Blog.BlogTitle
    </td>
  </tr>
  <tr>
    <td>
      Blog Description
    </td>
    <td>
      @ViewBag.Blog.BlogDescription
    </td>
  </tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
  @foreach (var comment in ViewBag.Comments)
  {
    <tr>
      <td>
        Comment
      </td>
      <td>
        @comment.Comment ( By @comment.CommentedBy )
      </td>
    </tr>
  }
</table>

```

Run the application and we will get the required output.

3. View Data

ViewData is used to pass data from controller to view. View Data scoper is only during the current request. View Data needs typecasting for getting data. ViewData is a property of ControllerBase class. ViewData is a derivative of the ViewDataDictionary class, so we can access with "key/value"

To use the ViewData for multiple models change your action method as below

Hide Copy Code

```

public ActionResult GetBlogComment()
{
    ViewData["Blog"] = GetBlogModel();
    ViewData["Comments"] = GetCommentModel();

    return View();
}

```

Now change your view (GetBlogComment.cshtml) according to view Data as below


```

@using MultipleModels.Models;
<h2>Blog Detail</h2>
@{
    BlogModel Blog = ViewData["Blog"] as BlogModel;
    List<CommentModel> Comments = ViewData["Comments"] as List<CommentModel>;
}
<table style="border:solid 2px;">
<tr>
<td>
        Blog Title
    </td>
<td>
        @Blog.BlogTitle
    </td>
</tr>
<tr>
<td>
        Blog Description
    </td>
<td>
        @Blog.BlogDescription
    </td>
</tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
    @foreach (var comment in Comments)
    {
        <tr>
            <td>
                Comment
            </td>
            <td>
                @comment.Comment ( By @comment.CommentedBy )
            </td>
        </tr>
    }
</table>

```

Run the application and we will get the same result as in approach 1.

4. Temp Data

TempData is also a dictionary derivative from TempDataDictionary class. TempData stored in short lives session. We can pass multiple models through TempData also.

Change your action method as below

Hide Copy Code

```

public ActionResult GetBlogComment()
{
    TempData["Blog"] = GetBlogModel();
    TempData["Comments"] = GetCommentModel();

    return View();
}

```

Now change your view (GetBlogComment.cshtml) according to Temp Data as below

```

using MultipleModels.Models;
<h2>Blog Detail</h2>
@{
    BlogModel Blog = TempData["Blog"] as BlogModel;
    List<CommentModel> Comments = TempData["Comments"] as List<CommentModel>;
}
<table style="border:solid 2px;">
<tr>
<td>
        Blog Title
    </td>
<td>
        @Blog.BlogTitle
    </td>
</tr>
<tr>
<td>
        Blog Description
    </td>
<td>
        @Blog.BlogDescription
    </td>
</tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
    @foreach (var comment in Comments)
    {
        <tr>
        <td>
            Comment
        </td>
        <td>
            @comment.Comment ( By @comment.CommentedBy )
        </td>
        </tr>
    }
</table>

```

Now run the application and we will get the required result.

5. Session

Session is used to pass data across controllers in MVC Application. Session data never expires (by default session expiration time is 20 minutes but it can be increased). Session is valid for all requests, not for a single redirect.

For Session, Change your action Method as below

Hide Copy Code

```

public ActionResult GetBlogComment()
{
    Session["Blog"] = GetBlogModel();
    Session["Comments"] = GetCommentModel();
}

```

```

    return View();
}

```

Now change your view (GetBlogComment.cshtml) according to Session as below

```

@using MultipleModels.Models;
<h2>Blog Detail</h2>
@{
    BlogModel Blog = Session["Blog"] as BlogModel;
    List<CommentModel> Comments = Session["Comments"] as List<CommentModel>;
}
<table style="border:solid 2px;">
    <tr>
        <td>
            Blog Title
        </td>
        <td>
            @Blog.BlogTitle
        </td>
    </tr>
    <tr>
        <td>
            Blog Description
        </td>
        <td>
            @Blog.BlogDescription
        </td>
    </tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
    @foreach (var comment in Comments)
    {
        <tr>
            <td>
                Comment
            </td>
            <td>
                @comment.Comment ( By @comment.CommentedBy )
            </td>
        </tr>
    }
</table>

```

Run the application and we will get the expected result.

6. Dynamic (ExpandoObject)

The ExpandoObject class enables us to add and delete members at run time. So If we want to build up an object to work with on the fly at runtime then we can use Expando Object.

I am not going in detail about ExpandoObject. let's see how can we use Expando with Multiple Models.

Change the action method as below

Hide Copy Code

```
public ActionResult GetBlogComment()
{
    dynamic BCVM = new ExpandoObject();
    BCVM.Blog = GetBlogModel();
    BCVM.Comments = GetCommentModel();
    return View(BCVM);
}
```

Note :To use ExpandoObject Add System.Dynamic Namespace.

Change your view as below

```
@model dynamic
@using MultipleModels.Models;
<h2>Blog Detail</h2>
<table style="border:solid 2px;">
    <tr>
        <td>
            Blog Title
        </td>
        <td>
            @Model.Blog.BlogTitle<br />
        </td>
    </tr>
    <tr>
        <td>
            Blog Description
        </td>
        <td>
            @Model.Blog.BlogDescription<br />
        </td>
    </tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
    @foreach (var comment in Model.Comments)
    {
        <tr>
            <td>
                Comment
            </td>
            <td>
                @comment.Comment ( By @comment.CommentedBy )
            </td>
        </tr>
    }
</table>
```

Run the application and we will get the expected result.

7. Tuples

Tuple is an ordered sequence, immutable, fixed-size and of heterogeneous objects. each object in tuple is being of a specific type.

Tuples are not new in programming. They are already used in F#, Python and databases. They are new to C#. The tuples were introduced in C# 4.0 with dynamic programming.

we can use Tuples also to pass multiple models to view as below

[Hide](#) [Copy Code](#)

```
public ActionResult GetBlogComment()
{
    var BCVM = new Tuple<BlogModel, List<CommentModel>>(GetBlogModel(), GetCommentModel());
    return View(BCVM);
}
```

And now change your view as below

```
@model Tuple<BlogModel, List<CommentModel>>
@using MultipleModels.Models;
<h2>Blog Detail</h2>
<table style="border:solid 2px;">
<tr>
<td>
Blog Title
</td>
<td>
@Model.Item1.BlogTitle<br />
</td>
</tr>
<tr>
<td>
Blog Description
</td>
<td>
@Model.Item1.BlogDescription<br />
</td>
</tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
@foreach (var comment in Model.Item2)
{
<tr>
<td>
Comment
</td>
<td>
@comment.Comment ( By @comment.CommentedBy )
</td>
</tr>
}
</table>
```

Run the application to see the result.

8. Render Action

Render actions are special controller methods defined only to be called from view. We create Render Action Method same as we create regular Action Method.

A Render action is a public method on the controller class. Render Action is called from view not from URL so we should decorate RenderAction with the [ChildActionOnly] attribute.

I am creating two Action Methods from where i will return partial view Results and will render those results on view using RenderAction Methods.

[Hide](#) [Copy Code](#)

```
public ActionResult GetBlogComment()
{
    return View();
}

public PartialViewResult RenderBlog()
{
    return PartialView(GetBlogModel());
}

public PartialViewResult RenderComments()
{
    return PartialView(GetCommentModel());
}
```

Right click on RenderBlog() and add a view as below (Make sure we need to create a partial view so check that option)

View name: RenderBlog

View engine: Razor (CSHTML)

☒ Create a strongly-typed view

Model class: BlogModel (MultipleModels.Models)

Scaffold template: Empty ☒ Reference script libraries

☒ Create as a partial view

☒ Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

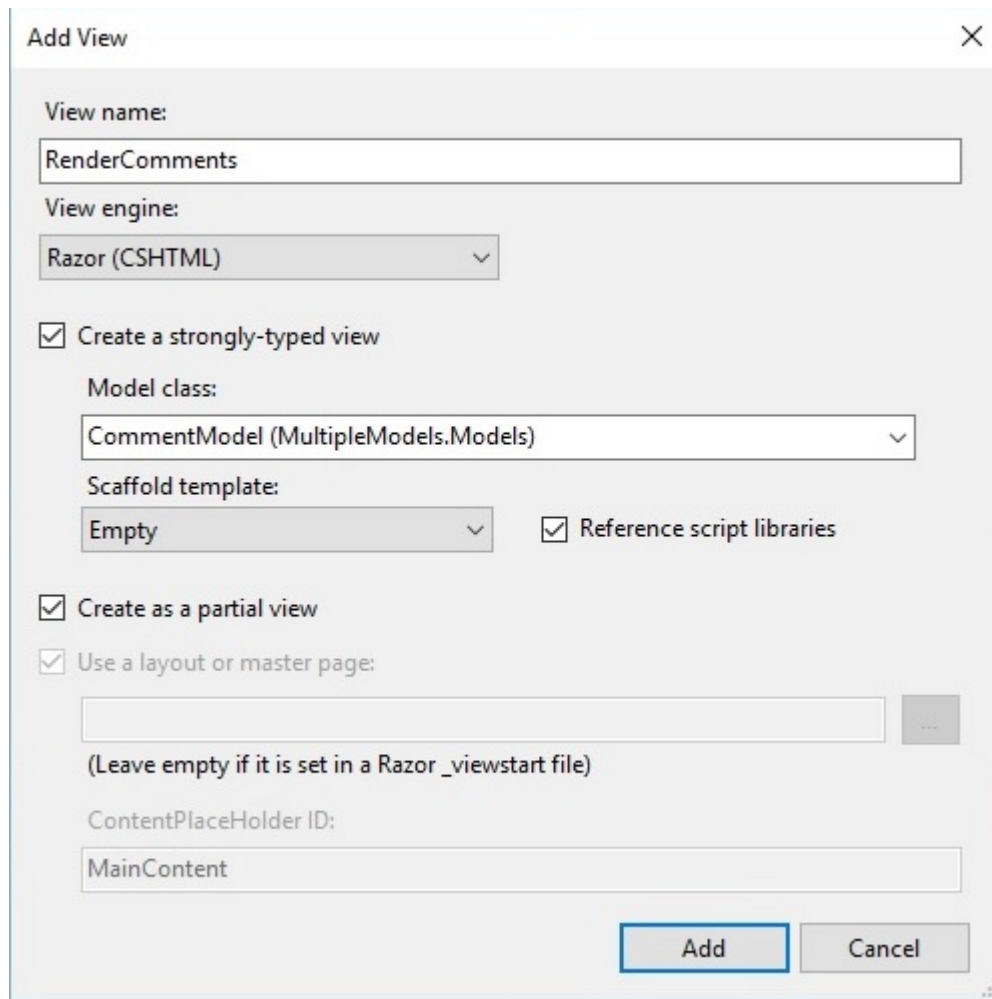
ContentPlaceHolder ID: MainContent

Add Cancel

Now change the code of your created partial(RenderBlog.cshtml) view as below

```
@model MultipleModels.Models.BlogModel
<table style="border:solid 2px;">
<tr>
<td>
Blog Title
</td>
<td>
@Model.BlogTitle<br />
</td>
</tr>
<tr>
<td>
Blog Description
</td>
<td>
@Model.BlogDescription<br />
</td>
</tr>
</table>
```

Same way right click on RenderComments and create another partial view as below



Add View

View name:
RenderComments

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
CommentModel (MultipleModels.Models)

Scaffold template:
Empty

☒ Reference script libraries

☒ Create as a partial view

☒ Use a layout or master page:
(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceholder ID:
MainContent

Add Cancel

change this partial view(RenderComments.cshtml) as below


```

@model IEnumerable<MultipleModels.Models.CommentModel>
<h2>Comment Details</h2>
<table style="border:solid 2px;">

    @foreach (var comment in Model)
    {
        <tr>
        <td>
            Comment
        </td>
        <td>
            @comment.Comment ( By @comment.CommentedBy )
        </td>
        </tr>
    }
</table>

```

Its time to render these two partial view on our main view named GetBlogComment.cshtml.

```

<h2>Blog Details</h2>
@{
    Html.RenderAction("RenderBlog");
    Html.RenderAction("RenderComments");
}

```

Now run the application and we will get the required result.

9. JSON

We can Bind Multiple Models with the help of Json as well. We will return JsonResult from action Method and on View through JQuery we can parse the JSON data and Bind on View. Here is the code

Change your action method so that Action Method can return the JSON

Hide Copy Code

```

public ActionResult GetBlogComment()
{
    return View();
}

public JsonResult RenderBlog()
{
    return Json(GetBlogModel());
}

```

```
}  
  
public JsonResult RenderComments()  
{  
    return Json(GetCommentModel());  
}
```

Change your view as below

```

<script src="../../Scripts/jquery-1.8.2.min.js"></script>
<link href="../../Content/Site.css" rel="stylesheet" />
<script type="text/javascript">
    $(document).ready(function () {
        $.ajax({
            type: 'POST',
            url: '/Blog/RenderBlog',
            success: function (result) {
                $("table:eq(0) tr:eq(0) td:eq(1)").empty().text(result.BlogTitle);
                $("table:eq(0) tr:eq(1) td:eq(1)").empty().text(result.BlogDescription);
            }
        });
    });

    $(document).ready(function () {
        $.ajax({
            type: 'POST',
            url: '/Blog/RenderComments',
            success: function (result) {
                var i = 0;
                $.each(result, function () {
                    $("table:eq(1)").append('<tr><td>Comment</td><td>' + result[i].Comment + ' ( By ' + result[i].CommentedBy + ' ) ' + '</td></tr>');
                    i = i + 1;
                });
            }
        });
    });
</script>

```

<h2>Blog Detail</h2>

<table style="border:solid 2px;">

<tr>

<td>

Blog Title

</td>

<td id="BlogTitle">

</td>

</tr>

<tr>

<td>

Blog Description

</td>

<td id="BlogDescription">

</td>

</tr>

</table>

<h2>Comment Details</h2>

<table style="border:solid 2px;"></table>

10. Navigation Properties

If we have two related models then we can bind a model into another model as a property and can pass to a View. For example we have a Blog Model and Comment Model and A blog can contains multiple comments so we can declare a Navigation Property in Blog Model as below

[Hide](#) [Copy Code](#)

```
public class BlogModel
{
    public int BlogID { get; set; }
    public string BlogTitle { get; set; }
    public string BlogDescription { get; set; }
    public DateTime CreatedDate { get; set; }

    public List<CommentModel> Comments { get; set; } //Navigation Property
}
```

Now change your GetBlogModel() function and GetBlogComment() Action as below

[Hide](#) [Shrink ▲](#) [Copy Code](#)

```
public ActionResult GetBlogComment()
{
    BlogModel BM = GetBlogModel();
    return View(BM);
}

public BlogModel GetBlogModel()
{
    BlogModel bModel = new BlogModel()
    {
        BlogID = 1,
        BlogTitle = "MVC Blog",
        BlogDescription = "This is MVC Blog",
        CreatedDate = DateTime.Now,
        Comments = GetCommentModel() //Add Comments here
    };
    return bModel;
}

public List<CommentModel> GetCommentModel()
{
    List<CommentModel> cModel = new List<CommentModel>();

    cModel.Add(new CommentModel() { BlogID = 1, CommentID = 1, Comment = "Good One", CommentedBy = "Vijay" });
    cModel.Add(new CommentModel() { BlogID = 1, CommentID = 2, Comment = "Nice", CommentedBy = "Nishant" });
    cModel.Add(new CommentModel() { BlogID = 1, CommentID = 2, Comment = "Perfect", CommentedBy = "Saurabh" });
}
```

```

    return cModel;
}

```

Now create your view as below

```

@model MultipleModels.Models.BlogModel
<h2>Blog Detail</h2>
<table style="border:solid 2px;">
<tr>
<td>
    Blog Title
</td>
<td id="BlogTitle">
    @Html.DisplayFor(m => m.BlogTitle)
</td>
</tr>
<tr>
<td>
    Blog Description
</td>
<td>
    @Html.DisplayFor(m=>m.BlogDescription)
</td>
</tr>
</table>
<h2>Comment Details</h2>
<table style="border:solid 2px;">
    @foreach(var comment in Model.Comments)
    {
<tr>
<td>
    Comment
</td>
<td>
    @comment.Comment (By @comment.CommentedBy )
</td>
</tr>
    }
</table>

```

At Last - If this is helpful and you liked this please vote above.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOl\)](https://www.codeproject.com/Articles/1108855/ways-to-Bind-Multiple-Models-on-a-View-in-MVC)

Share



About the Author



VijayRana




Technical Lead

India

Azure + C3.ai

10x less code
25x faster development
1/10 time to deploy

Get the report



Hi Myself Vijay having around 7 years of experience on Microsoft Technologies.

Comments and Discussions

You must [Sign In](#) to use this message board.



Spacing Relaxed Layout Normal Per page 25 [Update](#)

[First](#) [Prev](#) [Next](#)

Good Job	sreekanth Vangara	11-Sep-18 21:52
My vote of 5	krishna rana	20-Aug-18 21:35
Can we use the temdata for this??	udhaya2089	14-Apr-18 9:15
Blurry images	Member 13588846	8-Mar-18 6:36
My vote of 5	Humayun Kabir Mamun	6-Dec-17 2:55
Bind Multiple Models on a View in MVC	Huzaifa Abbas	28-Nov-17 7:40
Regarding Dynamic (ExpandoObject) (Point No 6)	Prem K	20-Nov-17 23:20
Good job	abd4web	7-Nov-17 16:41
My vote of 5	replication	24-Aug-17 5:08
Very Good!	Dat_Nguyen	28-May-17 6:45
Multiple models to a mvc view	A Rajab Nisha	19-Apr-17 21:28
My vote of 5	Karthik_Mahalingam	13-Apr-17 22:52
My vote of 5	H.Talebi	2-Apr-17 19:32
My vote of 5	Karthik_Mahalingam	21-Mar-17 21:29
My vote of 5	Praneet Nadkar	21-Feb-17 17:55
Any preference among these ways ?	RandySoft	23-Jan-17 16:34
My vote of 5	PauloJuanShirt	28-Jun-16 21:50
My vote of 5	wmjordan	27-Jun-16 16:20
Re: My vote of 5	VijayRana	27-Jun-16 19:07

 General  News  Suggestion  Question  Bug  Answer  Joke  Praise  Rant  Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#)

[Advertise](#)

[Privacy](#)

[Cookies](#)

[Terms of Use](#)

Layout: [fixed](#) | [fluid](#)

Article Copyright 2016 by VijayRana
Everything else Copyright © [CodeProject](#), 1999-2020

Web05 2.8.200606.1