Tagged as

C#
ASP.NET
Windows
HTML
Intermediate
MVC
jQuery
VS2013

Stats

877K views
29.5K downloads
369 bookmarked

Posted 24 Nov 2013

Multiple Models in a View in ASP.NET MVC 4 / MVC 5

Snesh Prajapati, 3 Nov 2014

★★★★★ 4.89 (321 votes) Rate:

Using Multiple Models in a View in ASP.NET MVC 4

Download source code for MVC 4 / Visual Studio 2012 - 2 MB

Download source code for MVC 5 / Visual Studio 2013 - 2 MB

Introduction

When I was a beginner for ASP.NET MVC, I faced a question many times: how many ways do you know to use/pass multiple models in/to a view? At that time, I knew only two ways, then I learned some other ways to achieve the same. In this article, I will share my learning so far. Hope it will be helpful for others.

Pre-requisite: *you should be at intermediate level with C# and ASP.NET MVC. If you are absolute new to C# and ASP.NET MVC, please look for other resources to understand terms used here. You can put comment below article in case you need any detail. I will be happy to help. Thanks.*

There are many ways of using multiple models in a view, most frequently used are given below:

- `ViewData`
- `ViewBag`
- `PartialView`
- `TempData`
- `ViewModel`
- `Tuple`

All the above ways of using multiple models in view have their place but we need to think and pick which one fits our requirements. All techniques have their own advantages and disadvantages and we have such discussion in another article ***How to Choose the Best Way to Pass Multiple Models***.

To understand the article better, please download the attached code, have an overview of the code, then follow the steps given in this article.

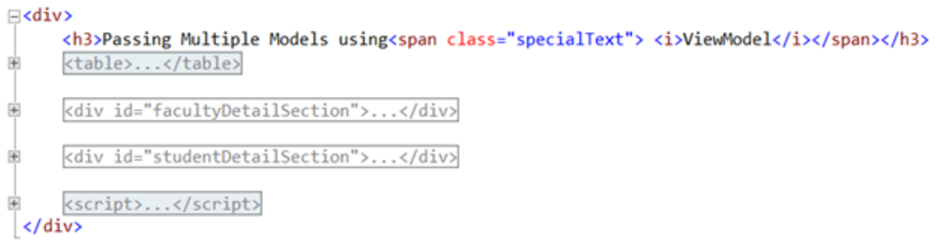
Overview of Sample Demo

The attached code solution has six views demonstrating the following:

- How to pass multiple models using `ViewData`
- How to pass multiple models using `ViewBag`
- How to pass multiple models using `PartialView`
- How to pass multiple models using `TempData`
- How to pass multiple models using `ViewModel`
- How to pass multiple models using `Tuple`

In the sample demo, all views will have similar HTML structure to get same layout but implementation to pass models to the view will be different.

Structure of the HTML tags is shown below only code inside the `scripts` tag will be changed as per the scenarios.



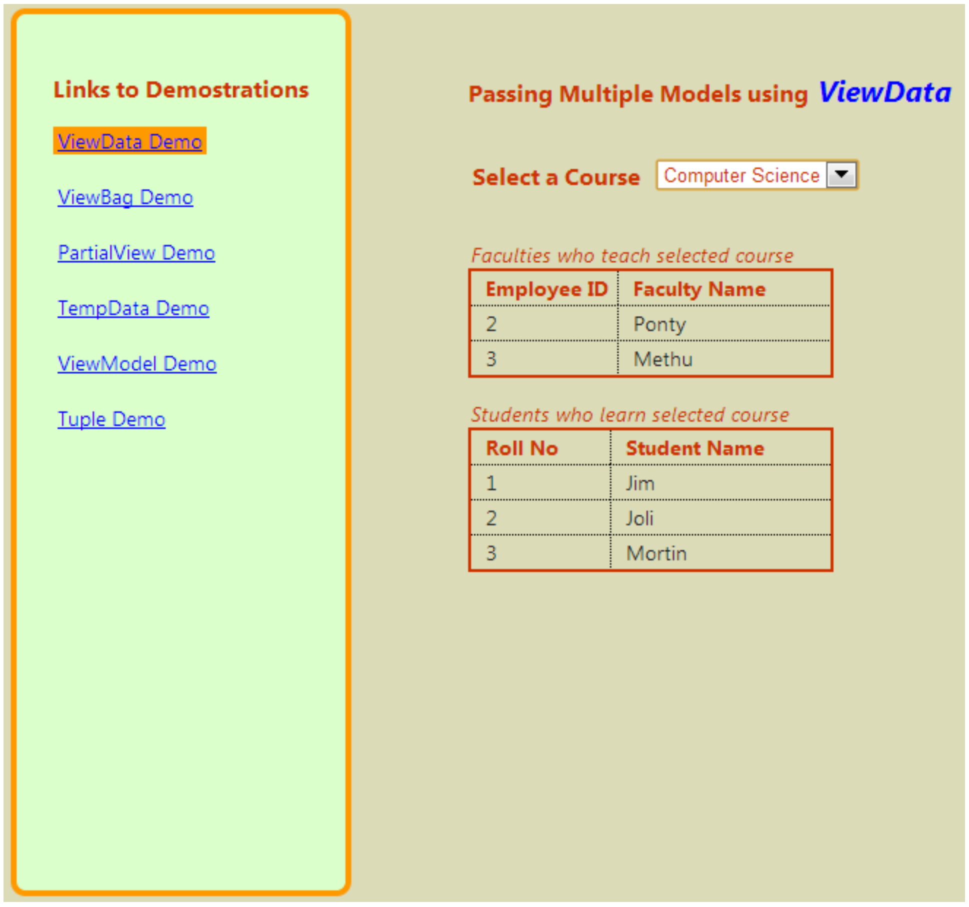
```
<div>
    <h3>Passing Multiple Models using<span class="specialText"> <i>ViewModel</i></span></h3>
    <table>...</table>

    <div id="facultyDetailSection">...</div>

    <div id="studentDetailSection">...</div>

    <script>...</script>
</div>
```

Output of all demos will be similar to the screenshot shown below:



Links to Demostrations

ViewData Demo

ViewBag Demo

PartialView Demo

TempData Demo

ViewModel Demo

Tuple Demo

Passing Multiple Models using *ViewData*

Select a Course: Computer Science

Faculties who teach selected course

Employee ID	Faculty Name
2	Ponty
3	Methu

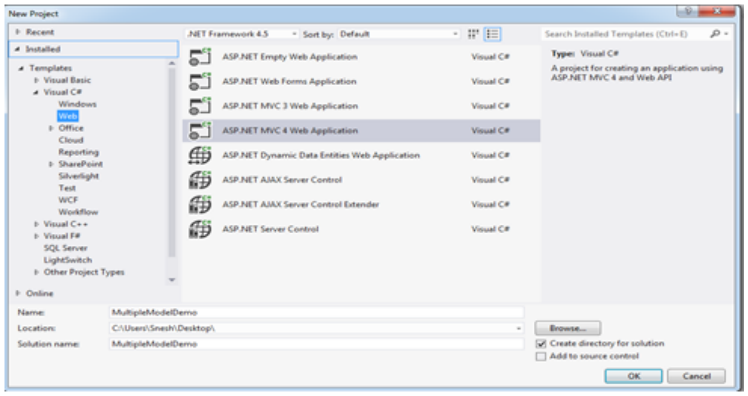
Students who learn selected course

Roll No	Student Name
1	Jim
2	Joli
3	Mortin

Creating Basic Structure of Sample Demo

Let's get started by creating a sample demo application. Follow the below steps:

1. Open Visual Studio 2012, select ASP.NET MVC4 Web Application template and give it project name as `MultipleModelDemo` and click OK. If you are using Visual Studio 2013 and MVC 5, Please look into the ***Update for VS 2013 / MVC 5*** section below.



New Project

.NET Framework 4.5 Sort by: Default

ASP.NET MVC 4 Web Application Visual C#

Type: Visual C#

A project for creating an application using ASP.NET MVC 4 and Web API

Name: MultipleModelDemo

Location: C:\Users\Snesh\Desktop\

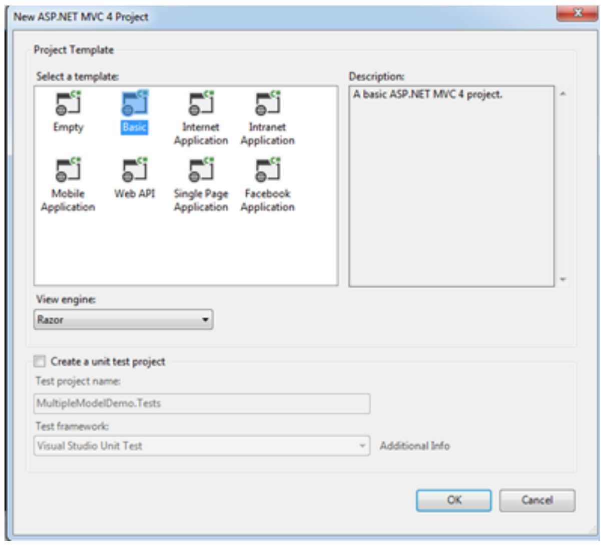
Solution name: MultipleModelDemo

[✓] Create directory for solution

[ ] Add to source control

Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
SQL Server
LightSwitch
Other Project Types
Online
ASP.NET Empty Web Application
ASP.NET Web Forms Application
ASP.NET MVC 3 Web Application
ASP.NET Dynamic Data Entities Web Application
ASP.NET AJAX Server Control
ASP.NET AJAX Server Control Extender
ASP.NET Server Control
Search Installed Templates (Ctrl+E)
Browse...
OK
Cancel

2. Select a template as Basic application then click OK. Visual Studio adds a `MultipleModelDemo` project in solution as shown below in screenshot.



New ASP.NET MVC 4 Project

Project Template

Select a template: Basic

Description: A basic ASP.NET MVC 4 project.

View engine: Razor

[ ] Create a unit test project

Test project name: MultipleModelDemo.Tests

Test framework: Visual Studio Unit Test

3. Right click on the *Models* folder, add a *Models.cs* file. Now we need to add three models named as `Course`, `Faculty` and `Student` by writing the following code as shown below:

Hide Copy Code

```
public class Course
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
}

public class Faculty
{
    public int FacultyId { get; set; }
    public string FacultyName { get; set; }
    public List<Course> AllotedCourses { get; set; }
}

public class Student
{
    public int EnrollmentNo { get; set; }
    public string StudentName { get; set; }
    public List<Course> EnrolledCourses { get; set; }
}
```

4. Add a class file under the *Models* folder named as *Repository.cs* file, which will have the implementation of methods to get hardcoded data for application in order to keep it convenient.

   Following is the code for `GetCourse` method which will return a list of courses:

Hide Copy Code

```
public List<Course> GetCourses()
{
    return new List<Course> {
        new Course () {  CourseId = 1, CourseName = "Chemistry"},
        new Course () {  CourseId = 2, CourseName = "Physics"},
        new Course () {  CourseId = 3, CourseName = "Math" },
        new Course () {  CourseId = 4, CourseName = "Computer Science" }
    };
}
```

Following is the code `GetFaculties` method which will return a list of faculties:

Hide Copy Code

```
public List<Faculty> GetFaculties()
{
    return new List<Faculty> {
        new Faculty () {  FacultyId = 1, FacultyName= "Prakash",
            AllotedCourses = new List<Course>
            {new Course () { CourseId = 1, CourseName = "Chemistry"},
                             new Course () { CourseId = 2, CourseName = "Physics"},
                             new Course () { CourseId = 3, CourseName = "Math"},
        }},
        new Faculty () {  FacultyId = 2, FacultyName= "Ponty" ,
            AllotedCourses = new List<Course>
            {new Course () { CourseId = 2, CourseName = "Physics"},
                             new Course () { CourseId = 4, CourseName = "Computer
Science"}
        }},
        new Faculty () {  FacultyId = 3, FacultyName= "Methu",
            AllotedCourses = new List<Course>
            {new Course () { CourseId = 3, CourseName = "Math"},
                             new Course () { CourseId = 4, CourseName = "Computer
Science"}
        }}
    };
}
```

Following is the code for `GetStudents` method which will return a list of students:

Hide Copy Code

```
public List<Student> GetStudents()
{
    List<Student> result = new List<Student> {
        new Student () { EnrollmentNo = 1, StudentName= "Jim",
            EnrolledCourses = new List<Course>
            { new Course () { CourseId = 1, CourseName = "Chemistry"},
                              new Course () { CourseId = 2, CourseName = "Physics"},
                              new Course () { CourseId = 4, CourseName = "Computer
Science"}
        }},

        new Student () {  EnrollmentNo = 2, StudentName= "Joli",
            EnrolledCourses = new List<Course>
            { new Course () { CourseId = 2, CourseName = "Physics"} ,
                              new Course ()
                              { CourseId = 4, CourseName = "Computer Science"}
        }},

        new Student () {  EnrollmentNo = 3, StudentName= "Mortin",
            EnrolledCourses = new List<Course>
            {  new Course () { CourseId = 3, CourseName = "Math"},
                               new Course () { CourseId = 4, CourseName = "Computer
Science"}
        }}
    };

    return result;
}
```

5. Add a `HomeController` to *Controller* folder. We will write the code in `HomeController` later.

6. In *Shared* folder, we will modify the existing code in *_Layout.cshtml* file. First write the following code in `head` tag:

Hide Copy Code

```
 <head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width" />

@*Referenced  js and css files which comes with template*@
 <script src="~/Scripts/jquery-1.8.2.min.js"></script>
 <link href="~/Content/Site.css" rel="stylesheet" />

 @*Referenced appcss file having custom CSS*@
 <link href="~/Content/appcss/appcss.css" rel="stylesheet" />
 </head>
```

Write the following code in `body` tag:

Hide Copy Code

```
<body>
    @* Define the place for navigation links in left side of the page*@
    <div class="navigationPanel">
        <div style="margin: 40px 25px 2px 25px;">
            <h3>Links to Demostrations</h3>
            <a href="~/Home/ViewDataDemo">ViewData Demo</a>
            <br />
            <br />
            <a href="~/Home/ViewBagDemo">ViewBag Demo</a>
            <br />
            <br />
            <a href="~/Home/PartialViewDemo">PartialView Demo</a>
            <br />
            <br />
            <a href="~/Home/TempDataDemo">TempData Demo</a>
            <br />
            <br />
            <a href="~/Home/ViewModelDemo">ViewModel Demo</a>
            <br />
            <br />
            <a href="~/Home/TupleDemo">Tuple Demo</a>
        </div>
    </div>
    <div style="float: left; margin: 25px 2px 2px 80px;">
        @RenderBody()
    </div>
</body>
```

So far, we have created basic code which we will be using in all scenarios. Further, we will learn each scenario one by one.

Passing Multiple Models using ViewData

`ViewData` is used to pass data from a controller to a view. `ViewData` is a dictionary of objects that is a type of `ViewDataDictionary` class. `ViewData` is defined as property in `ControllerBase` class. Values stored in `ViewData` require typecasting to their `datatype` in view. The values in `ViewData` are accessible using a key.

1. First, create a object of `Repository` class in `HomeController`.

Hide Copy Code

```
Repository _repository = new Repository();
```

Add the following code in HomeController.

Hide Copy Code

```
public ActionResult ViewDataDemo()
{
    ViewData["Courses"] = _repository.GetCourses();
    ViewData["Students"] = _repository.GetStudents();
    ViewData["Faculties"] = _repository.GetFaculties();
    return View();
}
```

Here `ViewDataDemo` action method will be passing all three models to its view using `ViewData`.

2. Add a view named as `ViewDataDemo` and write the same code as written below in `body` tag.

Hide Shrink ▲ Copy Code

```
<body style="background-color: #DBDBB7">
 <h3>Passing Multiple Models using <span class="
 specialText">  <i>ViewData </i> </span> </h3>
 <table>
     <tr>
         <td class="sltCourseText">
             <h3>Select a Course  </h3>
         </td>
         <td>
             <select id="sltCourse" class="sltStyle">
                <option>Select Course </option>
                @*Iterating Course model using ViewData string as a key *@
                @foreach (var item in ViewData["Courses"]
                as List <MultipleModelDemo.Models.Course>)
                {
                     <option>@item.CourseName
                     </option>
                }
             </select>
         </td>
     </tr>
 </table>

 <div id="facultyDetailSection">
     <h4>Faculties who teach selected course </h4>
     <div id="facultyDetailTable"> </div>
 </div>
```

```
<div id="studentDetailSection">
    <h4>Students who learn selected course </h4>
    <div id="studentDetailTable"> </div>
</div>
</body>
```

Here we are iterating `Course` model using `ViewData` and casting it into a `List<Course>`.

3. Inside the `body` tag, add a `script` tag and write the following code. Here we would show the table only when user will select a valid course otherwise the `facultyDetailSection` and `studentDetailSection` should not be appearing. We are using `fadeout` and `fadeIn` function of jQuery for that purpose.

Hide Copy Code

```
<script>
       $(document).ready(function(){
           $("#facultyDetailSection").fadeOut(0);
           $("#studentDetailSection").fadeOut(0);
       });

       var selectedCourseName;

       $("#sltCourse").change(function () {
           selectedCourseName = $("#sltCourse").val().trim();

           if (selectedCourseName === "Select Course") {
               $("#facultyDetailSection").fadeOut();
               $("#studentDetailSection").fadeOut();
           }
           else {
               getFacultyTable();
               getStudentTable();
               $("#facultyDetailSection").fadeIn();
               $("#studentDetailSection").fadeIn();
           }
       });
</script>
```

Here on the change event of `Course` dropdown, we will read the value and confirm if there is a valid selection to display faculties and students. In case of a valid selection, `getFacultyTable` and `getStudentTable` function will be executed. Add the following functions in `script` tag:

Hide Shrink ▲ Copy Code

```
    //Create table to display faculty detail
    function getFacultyTable() {
    $("#facultyDetailTable").empty();
    $("#facultyDetailTable").append("<table id='tblfaculty'
    class='tableStyle'><tr><th class='tableHeader' style='width:60px;'>
                                    Employee ID </th>
                                    <th class='tableHeader'>Faculty Name
                                    </th> </tr> </table>");

    //Get all faculties with the help of model
    //(ViewData["Faculties"]), and convert data into json format.
    var allFaculties = @Html.Raw(Json.Encode(ViewData["Faculties"]));

    for (var i = 0; i  < allFaculties.length; i++) {
        var allotedCourses = allFaculties[i].AllotedCourses;
        for (var j = 0; j  < allotedCourses.length; j++) {
            if(allotedCourses[j].CourseName === selectedCourseName)
             $("#tblfaculty").append
             (" <tr> <td class='tableCell'>"  +
             allFaculties[i].FacultyId
                 + " </td> <td class='tableCell'>"+
                 allFaculties[i].FacultyName+" </td> </tr>");
        }
    }
}

//Create table to display student detail
function getStudentTable() {
    $("#studentDetailTable").empty();
    $("#studentDetailTable").append(" <table id='tblStudent'
    class='tableStyle'> <tr> <th class='tableHeader' style='width:60px;'>
                                    Roll No </th> <th
                                    class='tableHeader'>Student Name
                                    </th> </tr> </table>");

    //Get all students with the help of model
    //(ViewData["Students"]), and convert data into json format.
    var allStudents = @Html.Raw(Json.Encode(ViewData["Students"]));

    for (var i = 0; i  < allStudents.length; i++) {
        var studentCourses = allStudents[i].EnrolledCourses;
        for (var j = 0; j  < studentCourses.length; j++) {
            if(studentCourses[j].CourseName === selectedCourseName)
```

```
                    $("#tblStudent").append(" <tr>
                    <td class='tableCell'>"  + allStudents[i].EnrollmentNo
                        + " </td> <td  class='tableCell'>
                        "+allStudents[i].StudentName+" </td> </tr>");
            }
        }
}
```

Note: As we have mentioned, all view will have same layout so further demos will have almost the same code as you have written in `body` tag of *ViewDataDemo.cshtml* file, now in further demos we just need to modify `foreach` function to fill data in dropdown and two lines of code in `getFacultyTable` and `getStudentTable` JavaScript function.

Passing Multiple Models using ViewBag

`ViewBag` is also used to pass data from a controller to a view. It is a dynamic property which comes in `ControllerBase` class that is why it doesn't require typecasting for `datatype`.

1. Add the following code in `HomeController`:

Hide Copy Code

```
public ActionResult ViewBagDemo()
{
    ViewBag.Courses = _repository.GetCourses();
    ViewBag.Students = _repository.GetStudents();
    ViewBag.Faculties = _repository.GetFaculties();
    return View();
}
```

Here `ViewBagDemo` action method will be passing data to view (*ViewBagDemo.cshtml*) file using `ViewBag`.

2. Add a view named as `ViewBagDemo`. All code will be same as you have written in *ViewDataDemo.cshtml* file. Just modify input model to `foreach` function.

Hide Copy Code

```
@*Iterating Course model using ViewBag *@
@foreach (var item in ViewBag.Courses)
{
    <option>@item.CourseName</option>
}
```

3. In `script` tag, replace the following line of code in `getFacultyTable` function:

Hide Copy Code

```
//Get all faculties with the help of model
// (ViewBag.Faculties), and convert data into json format.
var allFaculties = @Html.Raw(Json.Encode(ViewBag.Faculties));
```

4. Replace the following line of code in `getStudentTable` function:

Hide Copy Code

```
//Get all students with the help of model(ViewBag.Students),
//and convert data into json format.
var allStudents = @Html.Raw(Json.Encode(ViewBag.Students));
```

Passing Multiple Models using PartialView

Partial view is used where you need to share the same code (Razor and HTML code) in more than one view. For more details about `PartialView`, please visit here.

1. In Home controller, add the following code, `PartialViewDemo` action method will return a view having the list of all courses only. This action method will not have or pass any faculty or student information as of now.

Hide Copy Code

```
public ActionResult PartialViewDemo()
{
    List<Course> allCourse = _repository.GetCourses();
    return View(allCourse);
}
```

2. Add a view named as `PartialViewDemo`. All HTML code will be same as you have written before. Just modify `foreach` function.

Hide Copy Code

```
@*Iterating Course Model*@
@foreach(var item in Model)
{
    <option>@item.CourseName
    </option>
}
```

3. In `script` tag, modify `getFacultyTable` function as written below:

Hide Copy Code

```
function getFacultyTable() {
        $.ajax({
            // Get Faculty PartialView
            url: "/Home/FacultiesToPVDemo",
        type: 'Get',
        data: { courseName: selectedCourseName },
        success: function (data) {
        $("#facultyDetailTable").empty().append(data);
        },
        error: function () {
        alert("something seems wrong");
    }
        });
}
```

4. Modify `getStudentTable` function as written below:

Hide Copy Code

```
    function getStudentTable() {
    $.ajax({
        // Get Student PartialView
        url: "/Home/StudentsToPVDemo",
        type: 'Get',
        data: { courseName: selectedCourseName },
        success: function (data) {
            $("#studentDetailTable").empty().append(data);
        },
        error: function () {
            alert("something seems wrong");
        }
    });
}
```

5. Add a new Action method in `HomeController` as `StudentsToPVDemo` and add the following code in `StudentsToPVDemo` action method.

Hide Copy Code

```
public ActionResult StudentsToPVDemo(string courseName)
{
    IEnumerable <Course> allCourses = _repository.GetCourses();
    var selectedCourseId = (from c in allCourses where
    c.CourseName == courseName select c.CourseId).FirstOrDefault();

    IEnumerable <Student> allStudents = _repository.GetStudents();
    var studentsInCourse = allStudents.Where(s =>
    s.EnrolledCourses.Any(c => c.CourseId == selectedCourseId)).ToList();

    return PartialView("StudentPV", studentsInCourse);
}
```

6. Add a `PartialView` to the Shared folder by right clicking on `StudentsToPVDemo` action method, give it name as `StudentPV`. `StudentsToPVDemo` action will return `StudentPV PartialView` having the list of students studying in a particular course.

7. Add the following code in *StudentPV.cshtml* file.

Hide Copy Code

```
@model  IEnumerable <MultipleModelDemo.Models.Student>
 <table id="tblFacultyDetail" class="tableStyle">
         <tr>
         <th class="tableHeader" style="width:60px;">Roll No </th>
         <th class="tableHeader">Student Name </th>
         </tr>
    @foreach (var item in Model)
    {
          <tr>
             <td  class="tableCell" >@item.EnrollmentNo </td>
             <td class="tableCell">@item.StudentName </td>
         </tr>
    }
 </table>
```

8. Add a new action method to call `PatialView` for faculties in `HomeController`. Name it as `FacultiesToPVDemo` and add the following code:

Hide Copy Code

```
public ActionResult FacultiesToPVDemo(string courseName)
{
    IEnumerable <Course> allCourses = _repository.GetCourses();
    var selectedCourseId = (from c in allCourses where
    c.CourseName == courseName select c.CourseId).FirstOrDefault();

    IEnumerable <Faculty> allFaculties = _repository.GetFaculties();
    var facultiesForCourse = allFaculties.Where(f =>
    f.AllotedCourses.Any(c =>  c.CourseId == selectedCourseId)).ToList();
    return PartialView("FacultyPV", facultiesForCourse);
  }
```

9. Add a `PartialView` named as `FacultyPV` as we did for student `PartialView`, write the same code as you have written in *StudentPV.cshtml* file. Just replace one line of code as:

Hide Copy Code

```
@model  IEnumerable<MultipleModelDemo.Models.Faculty>
```

10. `FacultiesToPVDemo` action will return `FacultyPV` `PartialView` having the list of faculties who teach a particular course.

Passing Multiple Models using TempData

`TempData` is a dictionary of objects that is a type of `TempDataDictionary` class. `TempData` is defined as property in `ControllerBase` class. Values stored in `TempData` require typecasting to `datatype` in view. The values in `TempData` are accessible using a key. It is similar to `ViewData` but the difference is that it allow us to send and receive the data from one controller to another controller and from one action to another action. It's all about possible because it internally uses session variables.

For more information about `TempData`, please visit here. Here, we will use `TempData` only to hold (at server side `HomeController`) and pass (to the *TempDataDemo.cshtml*) List of Courses.

1. In home controller, add the following code:

Hide Copy Code

```
public ActionResult TempDataDemo()
{
    // TempData demo uses repository to get List<courses> only one time
    // for subsequent request to get List<courses> it will use TempData
    TempData["Courses"] = _repository.GetCourses();

    // This will keep Courses data untill next request is served
    TempData.Keep("Courses");
    return View();
}
```

2. Using `TempData`, we can pass values from one action to another action. `TempData["Courses"]` having the list of course. We will access this list in `FacultiesToTempDataDemo` action as shown below:

Hide Shrink ▲ Copy Code

```
public ActionResult FacultiesToTempDataDemo(string courseName)
{
    var allCourses = TempData["Courses"] as IEnumerable <Course>;

    // Since there are two AJAX call on TempDataPage
    // So we keep to keep Courses data for the next one
    TempData.Keep("Courses");
    var selectedCourseId = (from c in allCourses where
    c.CourseName == courseName select c.CourseId).FirstOrDefault();

    IEnumerable <Faculty> allFaculties = _repository.GetFaculties();
    var facultiesForCourse = allFaculties.Where(f =>
    f.AllotedCourses.Any(c => c.CourseId == selectedCourseId)).ToList();

    return PartialView("FacultyPV", facultiesForCourse);
}

public ActionResult StudentsToTempDataDemo(string courseName)
{
    var allCourses = TempData["Courses"] as IEnumerable <Course>;

    // If there is change in course again...it may cause more AJAX calls
    // So we need to keep Courses data
    TempData.Keep("Courses");

     var selectedCourseId = (from c in allCourses where
     c.CourseName == courseName select c.CourseId).FirstOrDefault();

    IEnumerable <Student> allStudents = _repository.GetStudents();
    var studentsInCourse = allStudents.Where(s =>
    s.EnrolledCourses.Any(c => c.CourseId == selectedCourseId)).ToList();

    return PartialView("StudentPV", studentsInCourse);
}
```

3. Add a view by right clicking on `TempDataDemo` action method. Write the same code as you have written in *PartialViewDemo.cshtml*. Only one line of code needs to modify as written below:

Hide Copy Code

```
@*Iterating Course model using TempData["Courses"] *@
@foreach (var item in TempData["Courses"]
    as List <MultipleModelDemo.Models.Course>)
{
    <option>@item.CourseName </option>
}
```

Modify `url` in `getFacultyTable` and `getStudentTable` function respectively as written below:

Hide Copy Code

```
url: "/Home/FacultiesToTempDataDemo",
url: "/Home/StudentsToTempDataDemo",
```

Both Action methods `FacultiesToTempDataDemo` and `StudentsToTempDataDemo` will return the same `PartialView` which we used for `PartialView` demo.

Passing Multiple Models using ViewModel

`ViewModel` is a pattern that allow us to have multiple models as a single class. It contains properties of entities exactly need to be used in a view. `ViewModel` should not have methods. It should be a collection of properties needed for a view.

1. We have three models (classes) named as `Student`, `Course` and `Faculty`. All are different models but we need to use all three models in a view. We will create a `ViewModel` by adding a new folder called *ViewModels* and add a class file called *ViewModelDemoVM.cs* and write the following code as shown below:

   Hide Copy Code

   ```
   public class ViewModelDemoVM
   {
       public List <Course> allCourses { get; set; }
       public List <Student> allStudents { get; set; }
       public List <Faculty> allFaculties { get; set; }
   }
   ```

   Note: As a good practice when you create any `ViewModel`, it should have suffix as VM or `ViewModel`.

2. In the `HomeController`, add the following code:

   Hide Copy Code

   ```
   public ActionResult ViewModelDemo()
   {
       ViewModelDemoVM vm = new ViewModelDemoVM();
       vm.allCourses = _repository.GetCourses();
       vm.allFaculties = _repository.GetFaculties();
       vm.allStudents = _repository.GetStudents();

       return View(vm);
   }
   ```

   `ViewModelDemo` is an action method that will return a view having `ViewModelDemoVM` which has lists of all `Courses`, `Faculties` and `Students`.

3. Right click on `ViewModelDemo` to add a view is called `ViewModelDemo`. *ViewModelDemo.cshtml* view will use `ViewModelDemoVM` as `Model` as shown below:

   Hide Copy Code

   ```
   @model MultipleModelDemo.ViewModel.ViewModelDemoVM
   ```

4. Modify `foreach` function.

   Hide Copy Code

   ```
   @*Iterating Course ViewModel *@
   @foreach (var item in Model.allCourses)
       {
           <option>@item.CourseName</option>
       }
   ```

5. Replace the following line of code in `getFacultyTable` function:

   Hide Copy Code

   ```
   //Get all faculties with the help of viewmodel
   //(Model.allFaculties), and convert data into json format.
   var allFaculties = @Html.Raw(Json.Encode(Model.allFaculties));
   ```

6. Replace the following line of code in `getStudentTable` function:

   Hide Copy Code

   ```
   //Get all students with the help of viewmodel(Model.allStudents),
   //and convert data into json format.
   var allStudents = @Html.Raw(Json.Encode(Model.allStudents));
   ```

Passing Multiple Models using Tuple

`Tuple` is a new class introduced in .NET Framework 4.0. It is an ordered sequence, immutable, fixed-size collection of heterogeneous (allows us to group multiple kind of data types) objects.

1. Add the following code in Home controller.

   Hide Copy Code
   ```
   public ActionResult TupleDemo()
   {
       var allModels = new Tuple <List <Course>,
       List <Faculty>,  List <Student>>
       (_repository.GetCourses(), _repository.GetFaculties(),  _repository.GetStudents()) {
   };
        return View(allModels);
   }
   ```

   Here, we are defining a `tuple` which is having lists of courses, faculties and students. We are passing this `tuple` to the `View`.

2. Add a `View` named as `TupleDemo`. Write code as you have written in *ViewModelDemo.cshtml* file. Just need to replace few lines of code. Replace model declarations as shown below. Here, we will use the namespace used to avoid long fully qualified class names.

   Hide Copy Code
   ```
   @using MultipleModelDemo.Models;
   @model Tuple <List <Course>, List <Faculty>, List <Student>>
   ```

3. Modify `foreach` function:

   Hide Copy Code
   ```
   @*Iterating Course model using tuple *@
   @foreach (var item in Model.Item1)
       {
            <option>@item.CourseName </option>
       }
   ```

   Here `Model.Item1` is mapped to `Course` model.

4. Inside the JavaScript function called `getFacultyTable` replace the following line of code:

   Hide Copy Code
   ```
   var allFaculties = @Html.Raw(Json.Encode(Model.Item2));
   ```

   Here `Model.Item2` is mapped with `Faculty` model.

5. Inside the JavaScript function called `getStudentTable`, replace the following line of code:

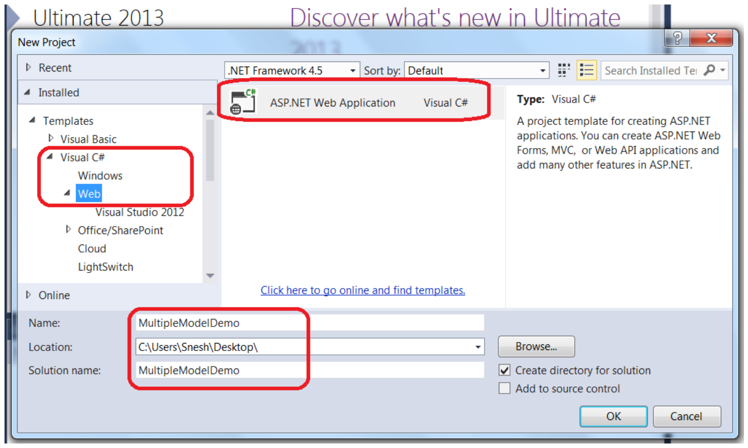
   Hide Copy Code
   ```
   var allStudents = @Html.Raw(Json.Encode(Model.Item3));
   ```

   Here `Model.Item3` is mapped with `Student` model.

Update for VS 2013 / MVC 5

If you would like to use Visual Studio 2013 / MVC 5, Step 1 and 2 as given for MVC 4 would be different, So please follow the steps as given below and then start from steps 3 as given for MVC 4 above.

1. Open Visual Studio 2013, Click on "New Project" and select ASP.NET Web Application template and give it project name as `MultipleModelDemo` and click OK.

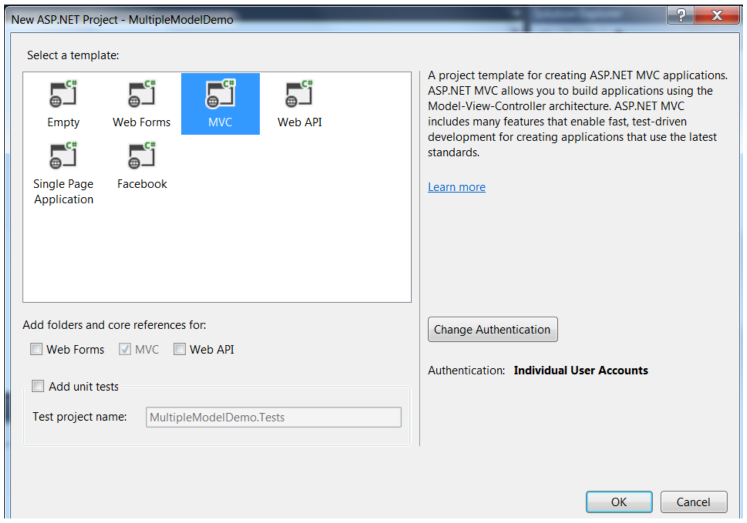


Type: Visual C#

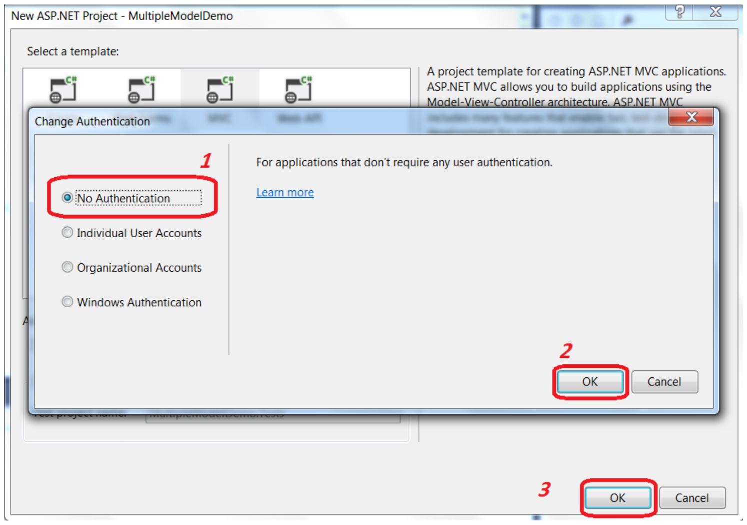
A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: MultipleModelDemo
Location: C:\Users\Snesh\Desktop\
Solution name: MultipleModelDemo
[✓] Create directory for solution
[ ] Add to source control

2. Select a template as MVC and make sure that all check box given in lower section of popup are unchecked other than MVC. Before clicking OK, we need to change Authentication (since for this demo we are not using either Authentication or Unit Tests to keep it simple and to the point). Please find screen shots below for the same:


New ASP.NET Project - MultipleModelDemo
Select a template:
Empty
Web Forms
MVC
Web API
Single Page Application
Facebook
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.
Learn more
Add folders and core references for:
Web Forms
MVC
Web API
Add unit tests
Test project name: MultipleModelDemo.Tests
Change Authentication
Authentication: Individual User Accounts
OK
Cancel

Changing the Authentication from "Individual User Account" to "No Authentication":



Visual Studio adds a `MultipleModelDemo` project in solution. Now you can follow the step 3 and further as given for MVC 4.

Conclusion

In this article, we learned how to use multiple models in a view. In the article ***How to Choose the Best Way to Pass Multiple Models***, I have shared my findings about when and where to pick a particular way to use multiple models in a view. Your inputs and queries are most welcome. Thanks.

License

Share

TWITTER FACEBOOK

About the Author

Snesh Prajapati
Software Developer
India

I am a Software Developer working on Microsoft technologies. My interest is exploring and sharing the awesomeness of emerging technologies.

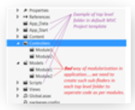
You may also be interested in...

The Offine-First Approach to Mobile App Development

SAPrefs - Netscape-like Preferences Dialog

10 ways to Bind Multiple Models on a View in MVC

Window Tabs (WndTabs) Add-In for DevStudio



Areas in ASP.NET MVC 4

Introduction to D3DImage

Comments and Discussions

Add a Comment or Question

Search Comments

Spacing Relaxed Layout Normal Per page 50 Update

First Prev Next

Very easy to understand	**anuverma033@gmail.com**	**15-Sep-17 0:03**
Re: Very easy to understand	Snesh Prajapati	15-Sep-17 9:07
Best article with nice demonstration.	**jyoti-kumari-5**	**7-Aug-17 14:42**
QUESTION	**Member 13135478**	**19-Apr-17 6:40**
hopes	**Member 13135478**	**18-Apr-17 11:16**
thanks	**Member 13135478**	**18-Apr-17 11:05**
Re: thanks	Snesh Prajapati	18-Apr-17 11:10
thank you	**Member 13135478**	**18-Apr-17 7:18**
Re: thank you	Snesh Prajapati	18-Apr-17 9:49

Good	**Member 12933551**	**2-Mar-17 1:46**
more model object inside tempdata MVC	**Abhilash.J.A**	**22-Feb-17 13:08**
Awesome tutorial	**Jim 817**	**28-Dec-16 10:49**
Re: Awesome tutorial	Snesh Prajapati	28-Dec-16 11:07
This is the way how a technical article should be written	**RAVI143**	**25-Nov-16 4:09**
Re: This is the way how a technical article should be written	Snesh Prajapati	25-Nov-16 12:15
Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	**Rod at work**	**5-Oct-16 17:18**
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Snesh Prajapati	5-Oct-16 21:14
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Rod at work	6-Oct-16 10:37
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Snesh Prajapati	6-Oct-16 20:30
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Rod at work	24-Oct-16 12:50
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Snesh Prajapati	24-Oct-16 20:54
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Rod at work	25-Oct-16 9:25
Re: Getting a "has no key defined" error message when trying to define a view based upon the ViewModel	Snesh Prajapati	25-Oct-16 10:11
My vote of 5	**Rod at work**	**3-Oct-16 17:04**
Re: My vote of 5	Snesh Prajapati	3-Oct-16 21:33
Good examples	**hafizmahmed**	**31-Aug-16 8:00**
Re: Good examples	Snesh Prajapati	31-Aug-16 11:23
Did you have tried CUD using this ideas?.	**Brian Rey Baril**	**12-Aug-16 4:31**
Re: Did you have tried CUD using this ideas?.	Snesh Prajapati	12-Aug-16 8:55
Re: Did you have tried CUD using this ideas?.	Brian Rey Baril	15-Aug-16 5:34
Great	**Bhuvanesh Mohankumar**	**27-Apr-16 12:05**
Re: Great	Snesh Prajapati	27-Apr-16 12:27
What is ExpandoObject (the System.Dynamic namespace)?	**Bhuvanesh Mohankumar**	**27-Apr-16 11:58**
Re: What is ExpandoObject (the System.Dynamic namespace)?	Snesh Prajapati	27-Apr-16 12:32
Very Comprehensive and simple post	**Ali Saeed Khan**	**18-Apr-16 8:08**
Re: Very Comprehensive and simple post	Snesh Prajapati	18-Apr-16 10:26
Single create action for multiple models	**Azamat Ismagulov**	**7-Apr-16 6:05**
Re: Single create action for multiple models	Pankaj9624263	24-Aug-16 1:40

Re: Single create action for multiple models	Snesh Prajapati	24-Aug-16 12:05
Really very good article	**niizgz**	**24-Mar-16 13:03**
Re: Really very good article	Snesh Prajapati	24-Mar-16 22:39
Nice and Simple	**Mahesh Pratap Singh**	**20-Mar-16 20:34**
Re: Nice and Simple	Snesh Prajapati	20-Mar-16 21:28
Good summary	**celli29**	**19-Mar-16 15:39**
Re: Good summary	Snesh Prajapati	19-Mar-16 21:08
Very nice	**NaibedyaKar**	**20-Feb-16 8:10**
Re: Very nice	Snesh Prajapati	20-Feb-16 8:22
Thanks	**Rathod Jaydev**	**10-Feb-16 5:25**
Re: Thanks	Snesh Prajapati	10-Feb-16 6:39
My vote of 5	**Member 12251195**	**8-Jan-16 1:14**

Last Visit: 26-Oct-17 13:49 Last Update: 26-Oct-17 13:49 Refresh 1 2 3 4 5 6 7 Next »

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 | Terms of Use | Mobile
Web02 | 2.8.171020.1 | Last Updated 3 Nov 2014

Layout: fixed | fluid