

Learn MVC Project in 7 days – Day 2



Marla Sukesh, 26 Nov 2015

In this article we learn MVC 5 step by step in 7 days – Day 2.

 [Download ViewBag_and_ViewData.zip](#)

 [Download Strongly_Typed_View.zip](#)

 [Download View_Model.zip](#)

 [Download Collections.zip](#)

Introduction

We believe you have successfully completed [day 1](#) before coming here.

Day 1 mainly concentrated on

- Why MVC?
- Asp.Net Web Forms vs. Asp.Net MVC
- Understanding Controllers and Views

Note:

If you have not completed the previous day, please make sure to complete it first. Our target is to create a small MVC project using best practices and modern methodology at the end of the day. With every next lab we will either add new functionality to previous lab or make previous lab better.

Complete Series

- [Day 1](#)
- [Day 2](#)
- [Day 3](#)
- [Day 4](#)
- [Day 5](#)
- [Day 6](#)
- [Day 7](#)
- [Bonus Day 1](#)
- [Bonus Day 2](#)

We are pleased to announce that this article is now available as hard copy book you can get the same from [Amazon](#) and [FlipKart](#)

Day 2 Agenda

[Introduction](#)

[Day 2 Agenda](#)

[Passing Data from Controller to View](#)

[Lab 3 – Using ViewData](#)

Talk on Lab 3

Lab 4 – Using ViewBag

Talk on Lab 4

Problems with ViewData and ViewBag

Lab 5 - Understand strongly typed Views

Talk on Lab 5

Understand View Model in Asp.Net MVC

ViewModel a solution

Lab 6 – Implementing View Model

Talk on Lab 6

Lab 7– View With collection

Talk on Lab 7

Conclusion

Passing Data from Controller to View

View created in the Lab 2 is very much static. In real life scenario it will display some dynamic data. In the next lab we will display some dynamic data in the view.

View will get data from the controller in the form of Model.

Model

In Asp.Net MVC model represent the business data.

Lab 3 – Using ViewData

ViewData is a dictionary, which will contains data to be passed between controller and views. Controller will add items to this dictionary and view reads from it. Let's do a demo.

Step 1 - Create Model class

Create a new class Called Employee inside Model folder as follows.

```
public class Employee
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Salary { get; set; }
}
```

Step 2 -Get Model in controller

Create Employee object inside GetView method as follows

```
Employee emp = new Employee();
emp.FirstName = "Sukesh";
emp.LastName="Marla";
emp.Salary = 20000;
```

Note: Make sure to put using statement in the top or else we have to put fully qualified name of Employee class.

```
using WebApplication1.Models;
```

Step 3 – Create ViewData and return View

Store Employee object in viewdata as follows.

```
ViewData["Employee"] = emp;  
return View("MyView");
```

Step 4 - Display Employee Data in View

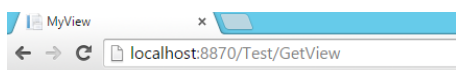
Open MyView.cshtml.

Retrieve the Employee data from the ViewData and display it as follows.

```
<div>  
    @{  
        WebApplication1.Models.Employee emp=(WebApplication1.Models.Employee)  
        ViewData["Employee"];  
    }  
  
<b>Employee Details </b><br />  
    Employee Name : @emp.FirstName@emp.LastName <br />  
    Employee Salary: @emp.Salary.ToString("C")  
</div>
```

Step 5- Test the output

Press F5 and test the application.



Employee Details

Employee Name : Suresh Marla

Employee Salary: \$20,000.00

Talk on Lab 3

What is the difference between writing Razor code with brace brackets (that is “{” and ”)”) and without brace brackets?

In the last lab @emp.FirstName can be replaced with following code snippet.

```
@{  
    Response.Write(emp.FirstName);  
}
```

@ Without brace brackets simply display the value of variable or expression.

Why casting is required?

ViewData holds objects internally. Every time a new value is added into it, it get boxed to object type.

So unboxing is required every time we try to extract value out of it.

What is the meaning of “@emp.FirstName @emp.LastName”?

It means Display First Name followed by a space and then last name.

Can we write same thing with single @ keyword?

Yes, then syntax for this will be @(emp.FirstName+" "+emp.LastName)

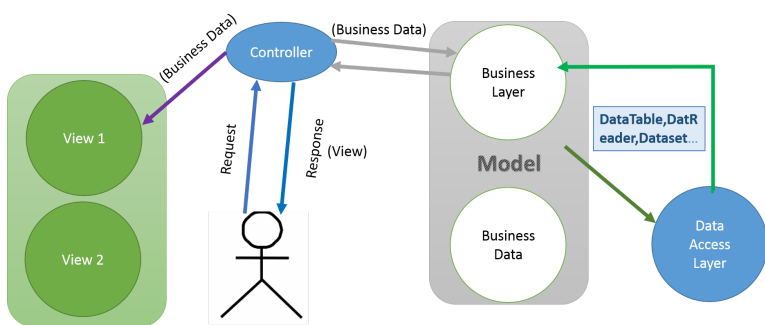
Why hardcoded Employee class is created in Controller?

Just for demo purpose. In real time we will get it from may be database or wcf or web service or may be from somewhere else.

What about the Database Logic/ Data Access Layer and Business Layer?

- Data Access Layer is one of the unspoken layer in Asp.Net MVC. It's always there but never included in MVC definition.
- Business layer as explained prior, it's a part of Model.

Complete MVC structure



Lab 4 – Using ViewBag

ViewBag is just a syntactic sugar for ViewData. ViewBag uses the dynamic feature of C# 4.0 and makes ViewData dynamic.

ViewBag internally uses ViewData.

Step 1 – Create View Bag

Continue with the same Lab 3 and replace Step 3 with following code snippet.

```
ViewBag.Employee = emp;
<span style="font-size: 9pt;">return View("MyView");</span>
```

Step 2 - Display EmployeeData in View

Change Step 4 with following code snippet.

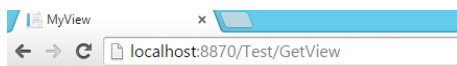
```
@{
    WebApplication1.Models.Employee emp = (WebApplication1.Models.Employee)
        ViewBag.Employee;
}
Employee Details

Employee Name: @emp.FirstName @emp.LastName

Employee Salary: @emp.Salary.ToString("C")
```

Step 3 - Test the output

Press F5 and test the application



Employee Details

Employee Name : Sukesh Marla

Employee Salary: \$20,000.00

Talk on Lab 4

Can we pass ViewData and get it as ViewBag?

Yes, We can. Vice versa is also possible. As I said before, ViewBag is just a syntactic sugar for ViewData,

Problems with ViewData and ViewBag

ViewData and ViewBag is a good option for passing values between Controller and View. But in real time projects it's not a good practice to use any of them. Let's discuss couple of disadvantages of using ViewData and ViewBag.

Performance issues

Values inside the ViewData are of type Object. We have to cast the value to correct type before using it. It adds additional overhead on performance.

No Type safety and no compile time errors

If we try to cast values to wrong type or if we use wrong keys while retrieving the values, we will get runtime error. As a good programming practice, error should be tackled in compiled time.

No Proper connection between Data sent and Data Received

As a developer I personally found this as a major issue.

In MVC, controller and View are loosely connected to each other. Controller is completely unaware about what’s happening in View and View is unaware about what’s happening in Controller.

From Controller we can pass one or more ViewData/ViewBag values. Now when Developer writes a View, he/she have to remember what is coming from the controller. If Controller developer is different from View developer then it becomes even more difficult. Complete unawareness. It leads to many run time issues and inefficiency in development.

Lab 5 - Understand strongly typed Views

Reason for all three problems of ViewData and ViewBag is the data type. Data type of values inside ViewData, which is “Object”.

Somehow if we were able to set the type of data which need to be passed between Controller and View problem will get solved and that’s where strongly typed Views comes to picture.

Let’s do a demo. This time we will take our View requirement to next level. If salary is greater than 15000 then it will be displayed in yellow colour or else green colour.

Step 1 – Make View a strongly typed view

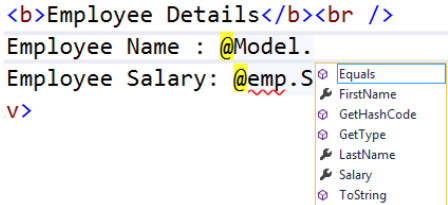
Add following statement in the top of the View

```
@model WebApplication1.Models.Employee
```

Above statement make our View a strongly typed view of type Employee.

Step 2 – Display Data

Now inside View simply type @Model and Dot (.) and in intellisense you will get all the properties of Model (Employee) class.



Write down following code to display the data

```
Employee Details

Employee Name : @Model.FirstName @Model.LastName

@if(Model.Salary>15000)
{
    <span style="background-color:yellow">
        Employee Salary: @Model.Salary.ToString("C")
    </span>
}
else
{
    <span style="background-color:green">
        Employee Salary: @Model.Salary.ToString("C")
    </span>
}
```

Step 3 – Pass Model data from Controller Action method

Change the code in the action method to following.

```
Employee emp = new Employee();  
emp.FirstName = "Sukesh";  
emp.LastName="Marla";  
emp.Salary = 20000;  
return View("MyView",emp);
```

Step 4 – Test the output

Employee Details

Employee Name : Sukesh Marla

Employee Salary: \$20,000.00

Talk on Lab 5

Is it required to type fully qualified class Name (Namespace.ClassName) in View every time?

No, we can put a using statement.

```
@using WebApplication1.Models  
@model Employee
```

Is it must to make View a strongly typed view always or can we go with ViewData or ViewBag sometimes?

As a best practice always make the view a strongly typed view.

Can we make our View a strongly typed view of more than one model?

No, we can't. In real time project we often end up at a point where we want to display multiple models in same view. Solution for this requirement will be discussed in next lab.

Understand View Model in Asp.Net MVC

In Lab 5 we have violated MVC principle. According to MVC, V that is View should be pure UI. It should not contain any kind of logic. We have done following three things which purely violates MVC architecture rules.

- Append First name and Last Name and Display it as Full Name - **Logic**
- Display Salary with Currency – **Logic**
- Display Salary in different colour based on value. In simple words Change appearance of HTML element based on some value. – **Logic**

Other than these three issues, there is one more point worth discussion.

Let say we have situation where we want to display more than one kind of data in the View.

Example – Show Current logged in User's Name along with Employee data

We can implement this in one of the following ways.

1. Add UserName property to Employee class –Every time we want to display new data in the view, adding new property to Employee class seems illogical. This new property may or may not be related to Employee. It also violates SRP of SOLID principle.
2. Use ViewBag or ViewData – We already discussed problems of using this approach.

ViewModel a solution

ViewModel is one of the unspoken layer in the Asp.Net MVC application. It fits between Model and View and act as data container for View

Difference between Model and ViewModel?

Model is Business specific data. It will be created based on Business and Database structure. ViewModel is View specific data. It will be created based on the View.

How it works exactly?

It's simple.

- Controller handle the user interaction logic or in simple words, handles the user's requests.
- Controller get one or more model data.
- Controller will decide which View suits best as response for the correct request.
- Controller will create and initialises ViewModel object from Model data retrieved based on View Requirement
- Controller will pass ViewModel data to View by means of ViewData/ViewBag/Strongly typed View.

- Controller will return the view.

How View and ViewModel will be connected here?

View is going to be a strongly typed view of type ViewModel.

How Model and ViewModel will be connected?

Model and ViewModel should be independent of each other. Controller will create and initialises ViewModel object based on one or more Model object.

Let's do a small lab to understand it better.

Lab 6 – Implementing View Model

Step 1 – Create Folder

Create a new folder called ViewModels in the project

Step 2 – Create EmployeeViewModel

In order to do that, let's list all the requirement on the view

1. First Name and Last Name should be appended before displaying
2. Amount should be displayed with currency
3. Salary should be displayed in different colour (based on value)
4. Current User Name should also be displayed in the view as well

Create a new class called EmployeeViewModel inside ViewModels folder will look like below.

```
public class EmployeeViewModel
{
    public string EmployeeName { get; set; }
    public string Salary { get; set; }
    public string SalaryColor { get; set; }
    public string UserName { get; set; }
}
```

Please note, in View Model class, FirstName and LastName properties are replaced with one single property called EmployeeName, Data type of Salary property is string and two new properties are added called SalaryColor and UserName.

Step 3 – Use View Model in View

In Lab 5 we had made our View a strongly type view of type Employee. Change it to EmployeeViewModel

```
@using WebApplication1.ViewModels
@model EmployeeViewModel
```

Step 4 – Display Data in the View

Replace the contents in View section with following snippet.

```
Hello @Model.UserName
<hr />
<div>
<b>Employee Details</b><br />
    Employee Name : @Model.EmployeeName <br />
<span style="background-color:@Model.SalaryColor">
    Employee Salary: @Model.Salary
</span>
</div>
```

Step 5 – Create and Pass ViewModel

In GetView action method, get the model data and convert it to ViewModel object as follows.

```
public ActionResult GetView()
{
    Employee emp = new Employee();
    emp.FirstName = "Sukesh";
    emp.LastName = "Marla";
    emp.Salary = 20000;
}
```

```

EmployeeViewModel vmEmp = new EmployeeViewModel();
vmEmp.EmployeeName = emp.FirstName + " " + emp.LastName;
vmEmp.Salary = emp.Salary.ToString("C");
if(emp.Salary>15000)
{
    vmEmp.SalaryColor="yellow";
}
else
{
    vmEmp.SalaryColor = "green";
}

vmEmp.UserName = "Admin"

return View("MyView", vmEmp);
}

```

Step 5 – Test the output

Press F5 and Test the output

Hello Admin

Employee Details

Employee Name : Sukesh Marla

Employee Salary: \$20,000.00

Same output as Lab 5 but this time View won't contain any logic.

Talk on Lab 6

Does it means, every model will have one View Model?

No, Every View will have its corresponding ViewModel.

Is it a good practice to have some relationship between Model and ViewModel?

No, as a best practice Model and ViewModel should be independent to each other.

Should we always create ViewModel? What if View won't contain any presentation logic and it want to display Model data as it is?

We should always create ViewModel. Every view should always have its own ViewModel even if ViewModel is going to contain same properties as model.

Let's say we have a situation where View won't contain presentation logic and it want to display Model data as it is. Let's assume we won't create a ViewModel in this situation.

Problem will be, if in future requirement, if we have been asked to show some new data in our UI or if we asked to put some presentation logic we may end with complete new UI creation from the scratch.

So better if we keep a provision from the beginning and Create ViewModel. In this case, in the initial stage ViewModel will be almost same as Model.

Lab 7– View With collection

In this lab we will display list of Employees in the View.

Step 1 – Change EmployeeViewModel class

Remove UserName property from EmployeeViewModel.

```

public class EmployeeViewModel
{
    public string EmployeeName { get; set; }
    public string Salary { get; set; }
    public string SalaryColor { get; set; }
}

```

Step 2– Create Collection View Model

Create a class called EmployeeListViewModel inside ViewModel folder as follows.


```
public class EmployeeListViewModel
{
    public List<EmployeeViewModel><employeeviewmodel> Employees { get; set; }
    public string UserName { get; set; }
}
</employeeviewmodel>
```

Step 3 – Change type of strongly typed view

Make MyView.cshtml a strongly typed view of type EmployeeListViewModel.

```
@using WebApplication1.ViewModels
@model EmployeeListViewModel
```

Step 4– Display all employees in the view

```
<body>
    Hello @Model.UserName
    <hr />
    <div>
        <table>
            <tr>
                <th>Employee Name</th>
                <th>Salary</th>
            </tr>
            @foreach (EmployeeViewModel item in Model.Employees)
            {
                <tr>
                    <td>@item.EmployeeName</td>
                    <td style="background-color:@item.SalaryColor">@item.Salary</td>
                </tr>
            }
        </table>
    </div>
</body>
```

Step 5 – Create Business Layer for Employee

In this lab, we will take our project to next level. We will add Business Layer to our project. Create a new class called EmployeeBusinessLayer inside Model folder with a method called GetEmployees.

```
public class EmployeeBusinessLayer
{
    public List<Employee><employee> GetEmployees()
    {
        List<Employee><employee> employees = new List<Employee><employee>();
        Employee emp = new Employee();
        emp.FirstName = "johnson";
        emp.LastName = "fernandes";
        emp.Salary = 14000;
        employees.Add(emp);</employee></employee></employee>

        emp = new Employee();
        emp.FirstName = "michael";
        emp.LastName = "jackson";
        emp.Salary = 16000;
        employees.Add(emp);

        emp = new Employee();
        emp.FirstName = "robert";
        emp.LastName = "pattinson";
        emp.Salary = 20000;
        employees.Add(emp);

        return employees;
    }
}
```

Step 6– Pass data from Controller

```
public ActionResult GetView()
{
    EmployeeListViewModel employeeListViewModel = new EmployeeListViewModel();

    EmployeeBusinessLayer empBal = new EmployeeBusinessLayer();
    List<employee> employees = empBal.GetEmployees();
```

```
List<EmployeeViewModel><employeeviewmodel> empViewModels = new List<EmployeeViewModel><employeeviewmodel>();

foreach (Employee emp in employees)
{
    EmployeeViewModel empViewModel = new EmployeeViewModel();
    empViewModel.EmployeeName = emp.FirstName + " " + emp.LastName;
    empViewModel.Salary = emp.Salary.ToString("C");
    if (emp.Salary > 15000)
    {
        empViewModel.SalaryColor = "yellow";
    }
    else
    {
        empViewModel.SalaryColor = "green";
    }
    empViewModels.Add(empViewModel);
}
employeeListViewModel.Employees = empViewModels;
employeeListViewModel.UserName = "Admin";
return View("MyView", employeeListViewModel);
}
```

Step 7 – Execute and Test the Output

Press F5 and execute the application.

Hello Admin

Employee Name	Salary
johnson fernandes	\$14,000.00
michael jackson	\$16,000.00
robert pattinson	\$20,000.00

Talk on Lab 7

Can we make View a strongly typed view of List<employeeviewmodel>?

Yes, we can.

Why we create a separate class called EmployeeListViewModel, why didn't we made View a strongly typed view of type List<EmployeeListViewModel><employeeviewmodel>?

If we use List<employeeviewmodel> directly instead of EmployeeListViewModel then there will be two problems.

- 1. Managing future presentation logic.
- 2. Secondly UserName property. UserName is not associated with individual employees. It is associated with complete View.

Why we removed UserName property from EmployeeViewModel and made it part of EmployeeListViewModel?

UserName is going to be same for all the employees. Keeping UserName property inside EmployeeViewModel just increase the redundant code and also increases the overall memoty requirement for data.

Conclusion

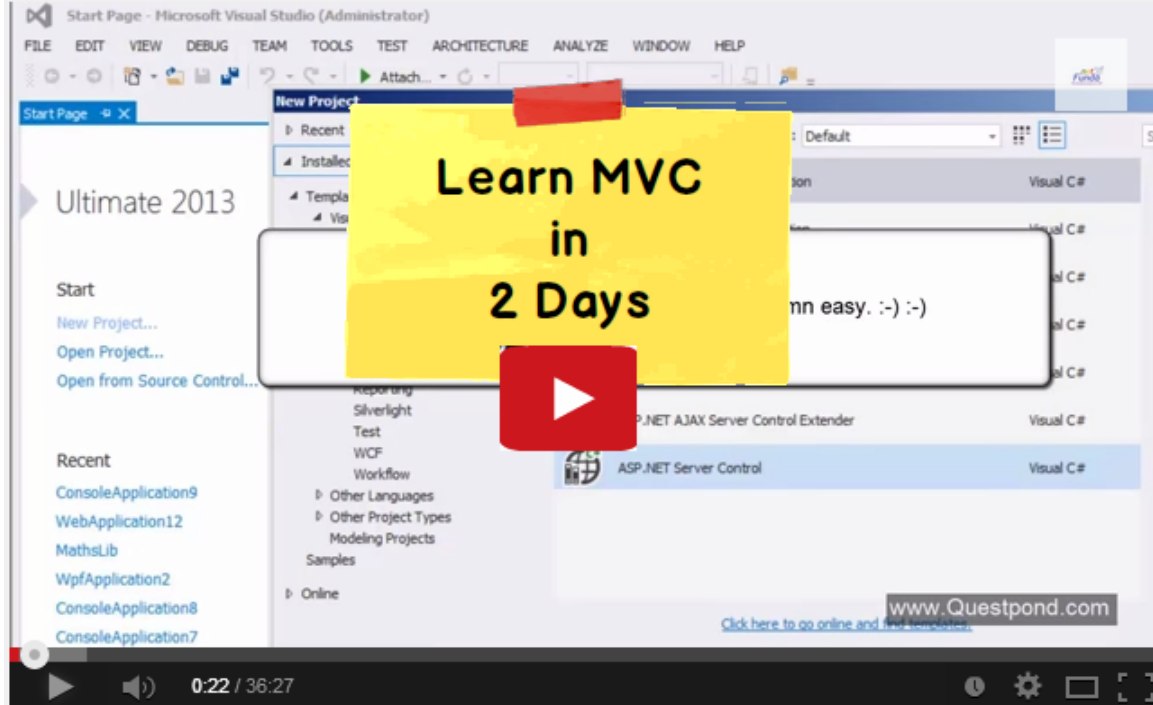
Here we complete our day 2. In Day 3 we will take our project into next version.

Connect us on [Facebook](#), [LinkedIn](#) or [twitter](#) to stay updated about new releases.

For Offline Technical trainings in Mumbai visit [StepByStepSchools.Net](#)

For Online Trainings visit [JustCompile.com](#)

In case you want to start with MVC 5 start with the below video [Learn MVC 5 in 2 days](#).



License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

About the Author



Marla Sukesh

   Instructor / Trainer Train IT
India 

Learning is fun but teaching is awesome.

Code re-usability is my passion ,Teaching and learning is my hobby, Becoming an successful entrepreneur is my goal.

By profession I am a Corporate Trainer.


I do trainings on WCF, MVC, Business Intelligence, Design Patterns, HTML 5, jQuery, JSON and many more Microsoft and non-Microsoft technologies.

Find my profile [here](#)

My sites

- [justcompile.com](#)
- [www.sukesh-marla.com](#)

Comments and Discussions

 **204 messages** have been posted for this article Visit <https://www.codeproject.com/Articles/897559/Learn-MVC-Project-in-days-Day> to post and view comments on this article, or click [here](#) to get a print view with messages.