

## **The Razor View Engine**

Def.: Razor is a markup syntax that allows you to embed server-side code (written in c# or vb) in an HTML Markup.

**Note:** the extension of view file is : .cshtml,.vbhtml

### **Razor view Syntax rules:**

- The @ character indicates a transition from markup code
- Razor code statements end with semicolon (;).
- C# code – case sensitive
- Variable should be declared by using the var keyword
- String should be enclosed with quotation mark.

To render a view inside a browser , MVC uses Razor as the default view engine.

### **Characteristics of Razor View Engine:**

- Clean
- Light weight
- Simple view(simple syntax , code)

- Razor allows embedding server side code in a view.

## **How to pass data from Control to View?**

### **1) Code expression inside view:**

<h1> display @mylist.Count or Length items</h1>

<h1> email id : gowthaman@@niit-karur.com</h1>

#### **Implicit code expresion**

<span>pirce : @model.price \* 1.2 </span>

#### **Explicit code expression: (parameter)**

<span> price : @(model.price\*1.2) </span>

### **2) Including code Blocks**

```
@ {  
    .... Add server side codes c# or vb coding  
}
```

### **3) Combining code and markup**

```
@foreach(var items in @products)  
{  
<htmltag>.....@items</htmltag>  
}  
@if(@viewbag.product!=null)  
{  
<htmltag>.....</htmltag>  
}
```

#### 4) Comment line

@\* .....\* @

#### 5) Partial view:

A partial view is a part of a view that can be used on multiple pages.

Ex:1 same as php include,include\_once

Ex:2 Title page is common to all views. That time we can use partial view.

Hint:

Partial views are created in the /Views/Shared folder.

The name of the partial view is prefixed with \_(underscore)

Syntax:

```
<div>@Html.Partial("_partialview")</div>
```

Div tag is must.

### There are three ways to data pass from controller to view.

1) ViewData → Dictionary → key ,value pair

Ex:

`ViewData["Stringvar"]=value;`

2) ViewBag → Object(Dynamic)

Ex:

`ViewBag.variable = value;`

3) TempData → Dictionary → Key,Value Pair.

Ex:

`TempData["variable"] = value;`

#### ViewData

1. ViewData is derived from the ViewDataDictionary class and is basically a Dictionary object i.e. Keys and Values where Keys are String while Values will be objects.
2. Data is stored as Object in ViewData.
3. While retrieving, the data it needs to be Type Casted to its original type as the data is stored as objects and it also requires NULL checks while retrieving.
4. ViewData is used for passing value from Controller to View.
5. ViewData is available only for Current Request. It will be destroyed on redirection.

#### Example

In the below example, a string value is set in the ViewData object in Controller and it is then displayed in View.

#### Controller

```
public class FirstController : Controller
{
    // GET: First
    public ActionResult Index()
    {
        ViewData["Message"] = "Hello MVC!";
        return View();
    }
}
```

#### View

```
<html>
<head>
```

```

        <meta name="viewport" content="width=device-width"/>
        <title>Index</title>
    </head>
    <body>
        <div>
            @ViewData["Message"]
        </div>
    </body>
</html>

```

## **ViewBag**

1. ViewBag is a Wrapper built around ViewData.
2. ViewBag is a dynamic property and it makes use of the C# 4.0 dynamic features.
3. While retrieving, there is no need for Type Casting data.
4. ViewBag is used for passing value from Controller to View.
5. ViewBag is available only for Current Request. It will be destroyed on redirection.

## **Example**

In the below example, a string value is set in the ViewBag object in Controller and it is then displayed in View.

### **Controller**

```

public class FirstController : Controller
{
    // GET: First
    public ActionResult Index()
    {
        ViewBag.Message = "Hello MVC!";
        return View();
    }
}

```

### **View**

```

<html>
<head>
    <meta name="viewport" content="width=device-width"/>
    <title>Index</title>
</head>
<body>
    <div>
        @ViewBag.Message
    </div>
</body>
</html>

```

## **TempData**

1. TempData is derived from the TempDataDictionary class and is basically a Dictionary object i.e. Keys and Values where Keys are String while Values will be objects.
2. Data is stored as Object in TempData.
3. While retrieving, the data it needs to be Type Casted to its original type as the data is stored as objects and it also requires NULL checks while retrieving.
4. TempData can be used for passing value from Controller to View and also from Controller to Controller.
5. TempData is available for Current and Subsequent Requests. It will not be destroyed on redirection.

### Example

In the below example, a string value is set in the TempData object in Controller and it is redirected to another Controller and finally it is displayed in View.

#### First Controller

```
public class FirstController : Controller
{
    // GET: First
    public ActionResult Index()
    {
        TempData["Message"] = "Hello MVC!";
        return new RedirectResult(@"~\Second\");
    }
}
```

#### Second Controller

```
public class SecondController : Controller
{
    // GET: Second
    public ActionResult Index()
    {
        return View();
    }
}
```

#### View of Second Controller

```
<html>
<head>
    <meta name="viewport" content="width=device-width"/>
    <title>Index</title>
</head>
<body>
    <div>
        @TempData["Message"];
    </div>
</body>
</html>
```

## ViewData VS ViewBag VS TempData

ViewData	ViewBag	TempData
It is Key-Value Dictionary collection	It is a type object	It is Key-Value Dictionary collection
ViewData is a dictionary object and it is property of ControllerBase class	ViewBag is Dynamic property of ControllerBase class.	TempData is a dictionary object and it is property of ControllerBase class.
ViewData is Faster than ViewBag	ViewBag is slower than ViewData	NA
ViewData is introduced in MVC 1.0 and available in MVC 1.0 and above	ViewBag is introduced in MVC 3.0 and available in MVC 3.0 and above	TempData is also introduced in MVC1.0 and available in MVC 1.0 and above.
ViewData also works with .net framework 3.5 and above	ViewBag only works with .net framework 4.0 and above	TempData also works with .net framework 3.5 and above
Type Conversion code is required while enumerating	In depth, ViewBag is used dynamic, so there is no need to type conversion while enumerating.	Type Conversion code is required while enumerating
Its value becomes null if redirection has occurred.	Same as ViewData	TempData is used to pass data between two consecutive requests.
It lies only during the current request.	Same as ViewData	TempData only works during the current and subsequent request

**Example combine code & markup:**

**(pass List<> data from controller to view)**

**Controller file(using Generic Class → List<>)**


```
}  
public ActionResult GView()  
{  
    List<string> Student = new List<string>();  
    Student.Add("Jignesh");  
    Student.Add("Tejas");  
    Student.Add("Rakesh");  
    Student.Add("Gowthaman");  
  
    TempData["Student"] = Student;  
  
    ViewBag.Student = Student;  
    ViewBag.totalcount = Student.Count;  
  
    ViewData["Student"]=Student;  
    return View("~/Views/MyFolder/GView1.cshtml");  
}
```





# View file

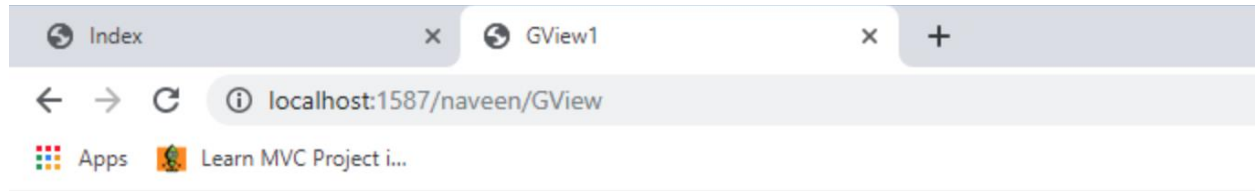
```
@*@ViewData["name"]*@
<ul>
    <h1> Total count of list : @ViewBag.totalcount </h1>
    <h1> Total count of list : @ViewBag.Student.Count </h1>
    @foreach (var item in @TempData["Student"] as List<String>)
    {
        <li> The item name is @item</li>
    }
    @foreach (var item in @ViewData["Student"] as List<String>)
    {
        <li> The item name is @item</li>
    }
    @foreach (var item in @ViewBag.Student as List<String>)
    {
        <li> The item name is @item</li>
    }
</div>
</body>
```



100 %

Source <html> <body> <div> <ul> <h1>

## Output:



**Total count of list : 4**

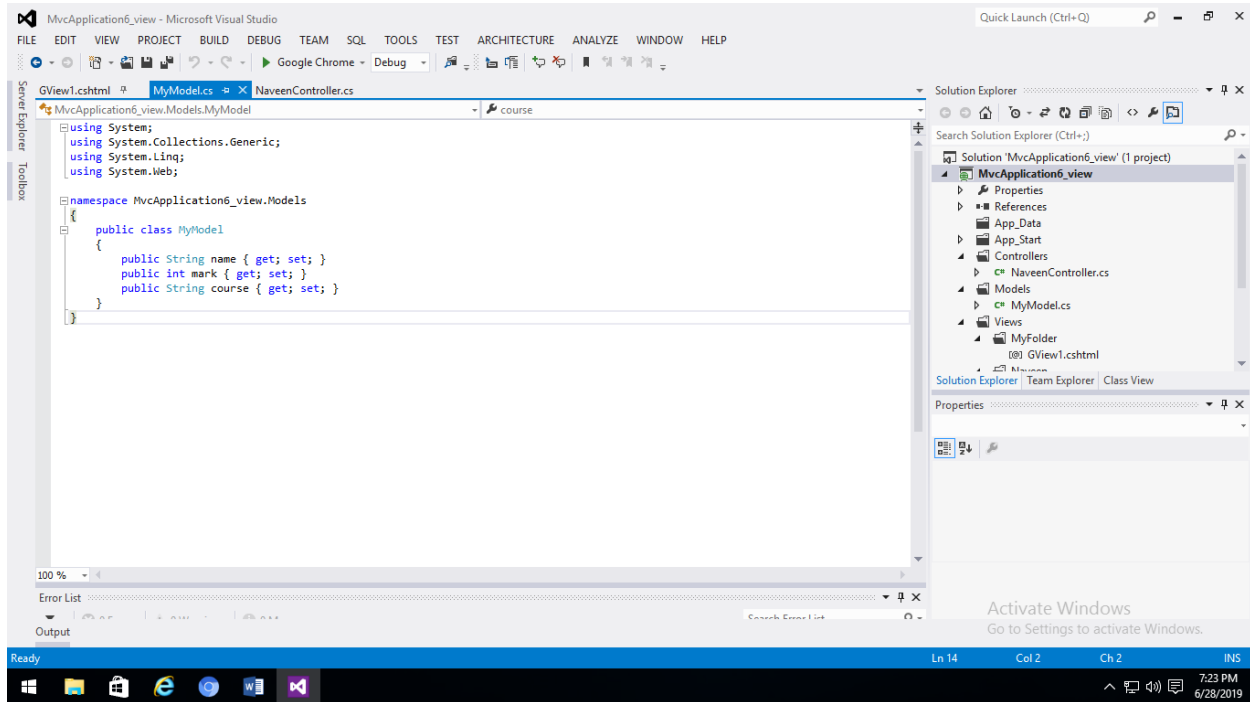
**Total count of list : 4**

- The item name is Jignesh
- The item name is Tejas
- The item name is Rakesh
- The item name is Gowthaman
- The item name is Jignesh
- The item name is Tejas
- The item name is Rakesh
- The item name is Gowthaman

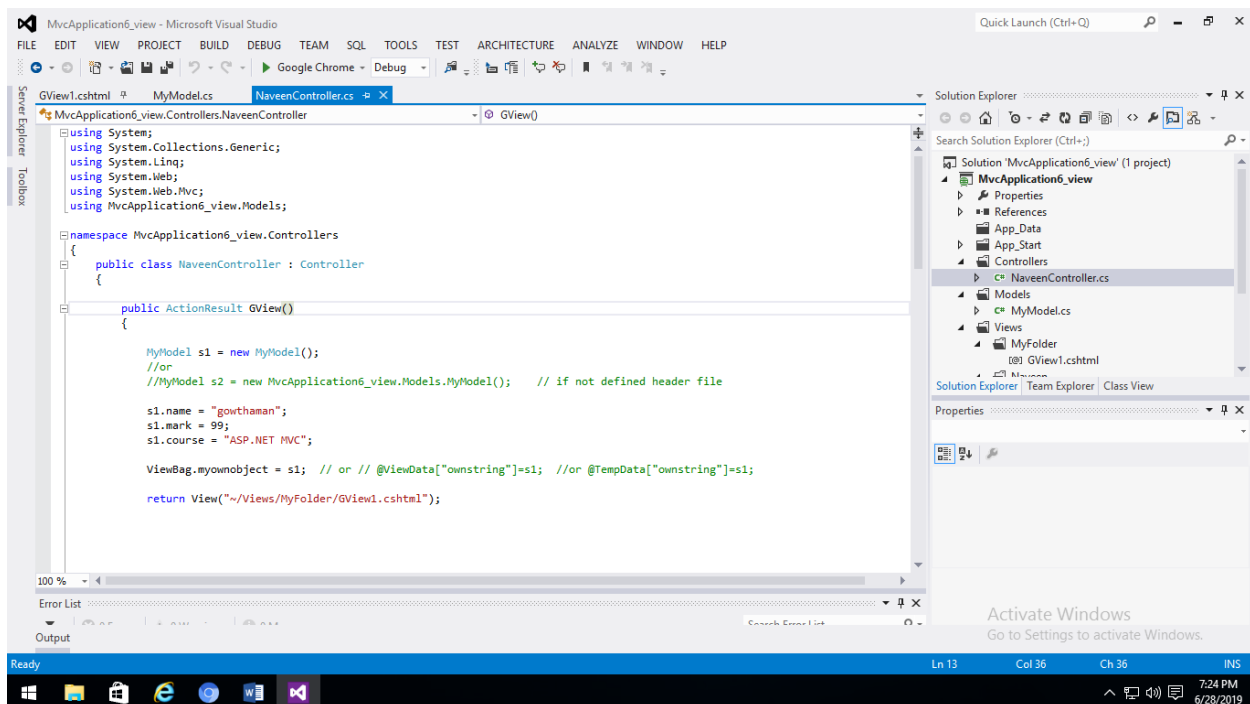


# Model Data from Controller to View

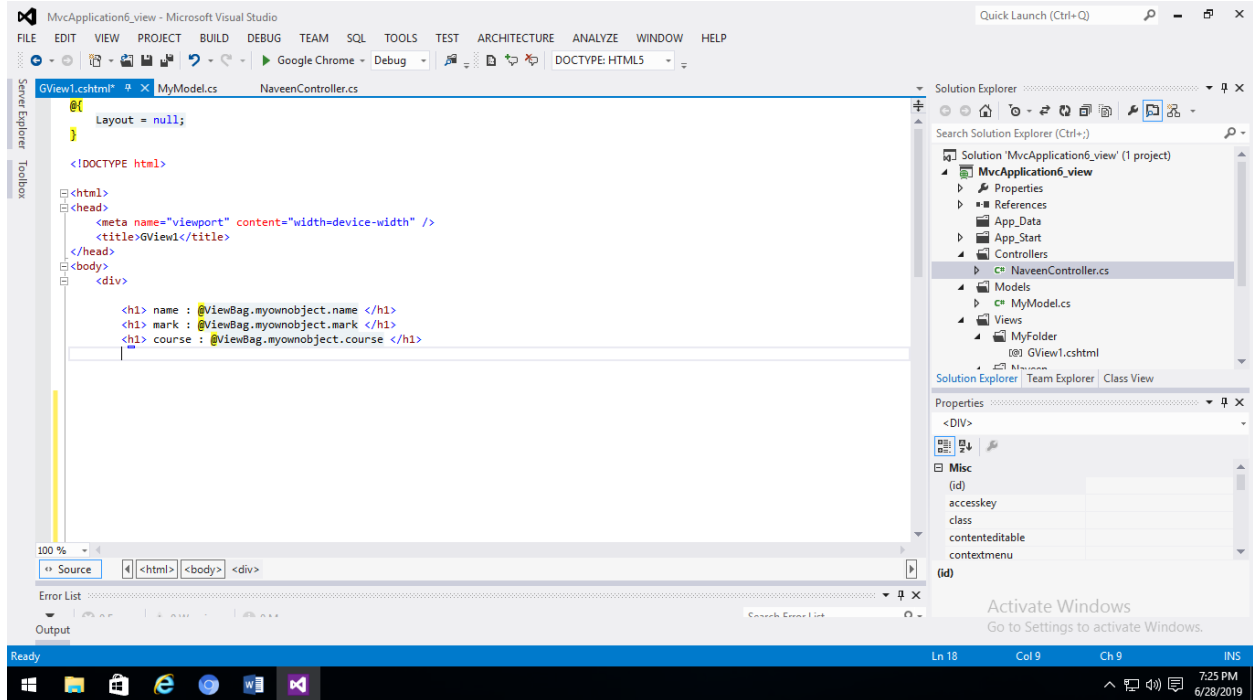
## model



## controller



# View:



# Output:



**name : gowthaman**

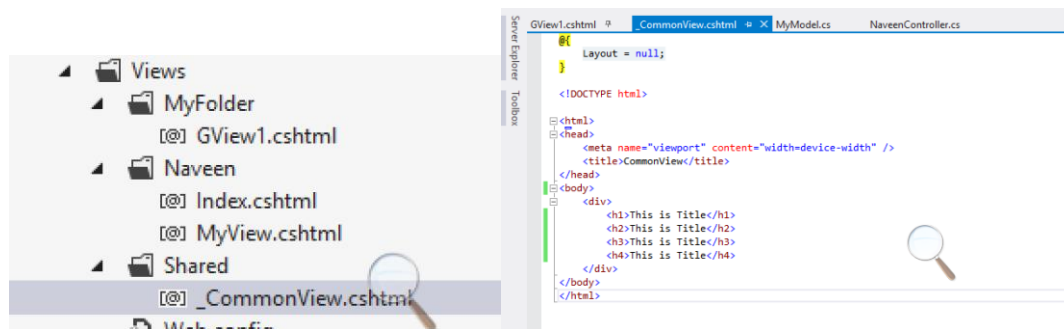
**mark : 99**

**course : ASP.NET MVC**

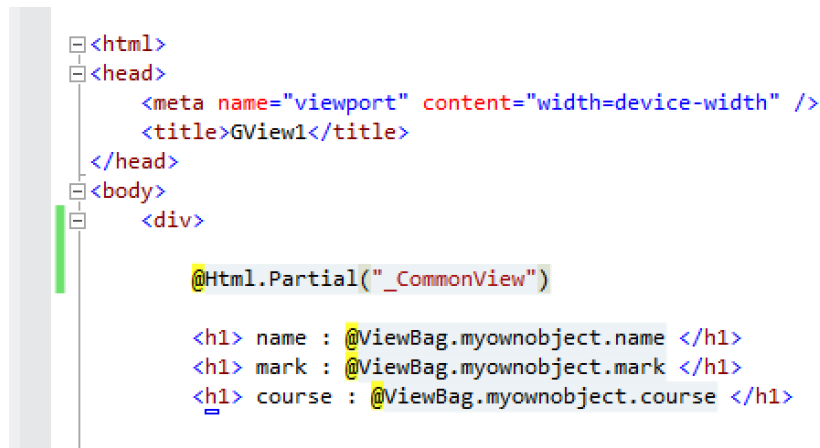
## Example partial view

Step 1: create on folder named Shared in Views

Step 2: add view → view name must start with \_(underscore) ex:  
\_CommonView.cshtml



Step 3:



Result:

# **This is Title**

**This is Title**

**This is Title**

**This is Title**

**name : gowthaman**

**mark : 99**

**course : ASP.NET MVC**

