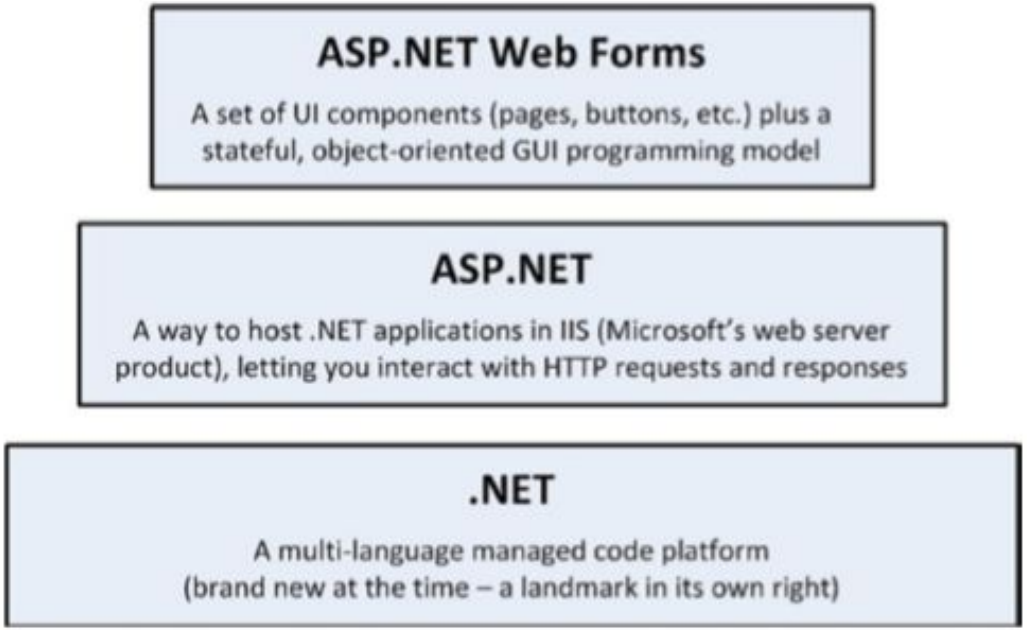
.Net MVC

Unit I

-JalpaPoriya

Understanding the History of ASP.NET

ASP.NET was a huge shift when it first arrived in 2002.
Figure 1-1 illustrates Microsoft's technology stack as it appeared then.



A Quick Introduction To Asp.Net MVC

- ASP.NET MVC is a framework for building web applications that applies the general ModelView-Controller pattern to the ASP.NET framework.

MVC as Applied to Web Frameworks

- The MVC pattern is used frequently in web programming. With ASP.NET MVC, it’s translated roughly as:
- **Models**: These are the classes that represent the domain you are interested in.
- **View**: This is a template to dynamically generate HTML.
- **Controller**: This is a special class that manages the relationship between the View and the Model.

MVC 4 Overview

- The MVC 4 release built on a pretty mature base and is able to focus on some more advanced scenarios. Some top features include:

1. ASP.NET Web API
2. Enhancements to default project templates Mobile project template using jQuery Mobile
3. Display modes
4. Task support for asynchronous controllers
5. Bundling and minification

The Road to MVC 5

- In the five years since ASP.NET MVC 1 was released in March 2009, we've seen five major releases of ASP.NET MVC and several more interim releases. To understand ASP.NET MVC 5, it's important to understand how we got here.

ASP.NET MVC 5 OVERVIEW

- MVC 5 was released along with Visual Studio 2013 in October 2013. The main focus of this release was on a “One ASP.NET” initiative and core enhancements across the ASP.NET frameworks. Some of the top features include:

1. One ASP.NET
2. New Web Project Experience
3. ASP.NET Identity
4. Bootstrap templates
5. Attribute Routing
6. ASP.NET scaffolding
7. Authentication filters
8. Filter overrides

Why MVC?

What Is Wrong with ASP.NET Web Forms?

Traditional ASP.NET Web Forms development was great in principle, but reality proved more complicated:

Problem 1 - View State weight

Problem 2 - Page life cycle

Problem 3 - Separation of pages

Problem 4 - Limited control over HTML

Problem 5 - Leaking abstraction

Problem 6 - Low testability

Problem 1 - View State weight:

- The actual mechanism for maintaining state across requests (known as View State) results in large blocks of data being transferred between the client and server.
- This data can reach hundreds of kilobytes in even modest Web applications, and it goes back and forth with every request, leading to slower response times and increasing the bandwidth demands of the server.

Back↲

Problem 2 - Page life cycle:

- The mechanism for connecting client-side events with server-side event handler code, part of the page life cycle, can be extraordinarily complicated and delicate.
- Few developers have success manipulating the control hierarchy at runtime without getting View State errors or finding that some event handlers mysteriously fail to execute.

Back↲

Problem 3 - False sense of separation of concerns:

- ASP.NET Web Forms' code-behind model provides a means to take application code out of its HTML markup and into a separate code-behind class.
- This has been widely applauded for separating logic and presentation, but, in reality, developers are encouraged to mix presentation code (for example, manipulating the serverside control tree) with their application logic (for example, manipulating database data) in these same monstrous code-behind classes. The end result can be weak and unclear.

Back↲

Problem 4 - Limited control over HTML:

- Server controls render themselves as HTML, but not necessarily the HTML you want.
- In early versions of ASP.NET, the HTML output failed to meet with Web standards or make good use of Cascading Style Sheets (CSS), and server controls generated unpredictable and complex ID attribute values that are hard to access using JavaScript. These problems are much improved in recent Web Forms releases, but it can still be tricky to get the HTML you expect.

Back↲

Problem 5 - Leaking of abstraction:

- Web Forms tries to hide HTML and HTTP wherever possible. As you try to implement custom behaviors, you frequently fall out of the abstraction, which forces you to reverse-engineer the postback event mechanism or perform obtuse acts to make it generate the desired HTML.
- Plus, all this abstraction can act as a frustrating barrier for competent Web developers.

Back↲

Problem 6 - Low testability:

- The designers of Web Forms could not have anticipated that automated testing would become an essential component of software development.
- Not surprisingly, the tightly coupled architecture they designed is unsuitable for unit testing. Integration testing can be a challenge, too.

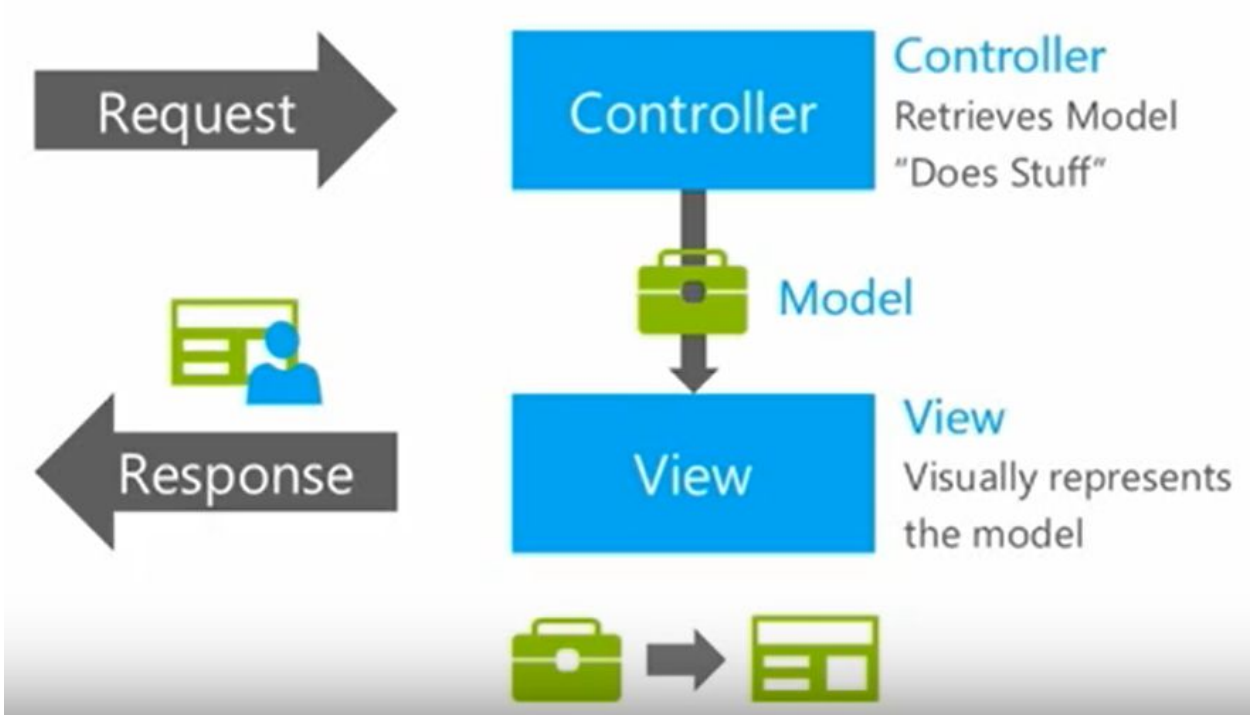
Back↲

Key Benefits of ASP.NET MVC

In October 2007, Microsoft announced a new MVC Web development platform, **built on the core ASP.NET platform**, clearly designed as a direct response to the evolution of technologies such as a reaction to the criticisms of Web Forms.

The following sections describe how this new platform overcame the Web Forms limitations and brought ASP.NET back to the cutting edge.

1) The MVC Pattern (MVC Architecture)



Model View Controller

- Whenever user made request for MVC page.
- Request directly go to **controller**, controller finds proper method and execute that method in this case no events executes in background.
- Request come through a method to class so we are not calling .aspx file we are calling method. So that will provide less response time. Class that handles communication from the user, overall application flow, and application specific logic.

Note : So what “Does Stuff” mean? It can be logic execution, authenticating someone, database activities, creating an image etc.

Model View Controller

- **Model** is a class.
- Model is just a packaging and sending data. Model is not directly connected with database, it can be just a collection, and it could be an array or a string. It is something that controller is passing to View through Model.
- **View** take data sent by Model and turn it in to HTML or other file like JSON.
- View don’t have any logic. It is just a template. It is pure HTML which decides how the UI is going to look.
- View have to rendered by a particular type of View engine.

2) Less Response Time:

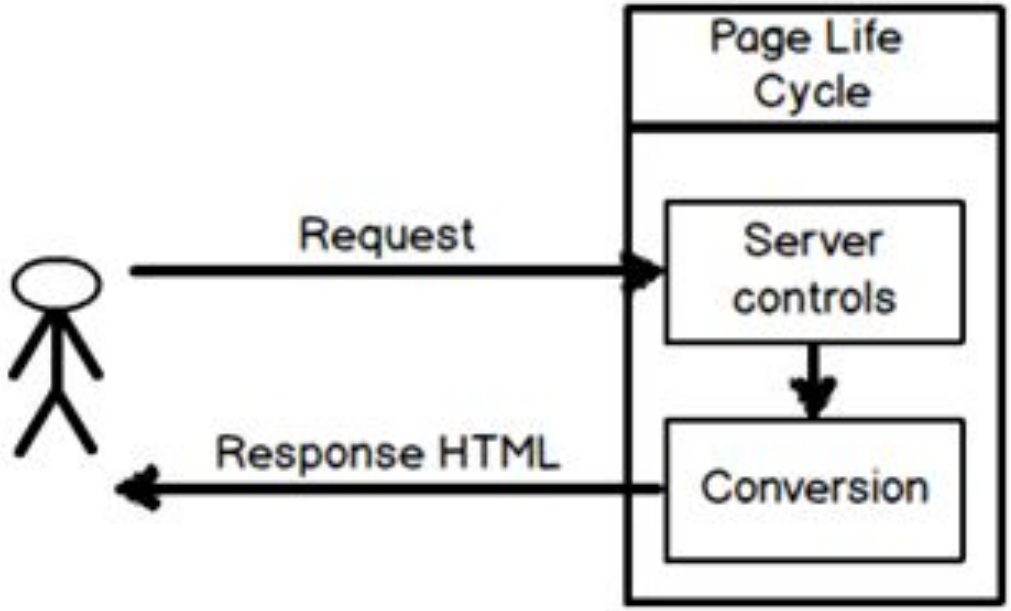
- When you are building an application with asp.net MVC there will be no illusions of state like page init, page load, page unload. There is no such thing as page life cycle.
- No conversion in controls

Server side Text box (ASP):

```
<asp:TextBox ID="TextBox1" runat="server" BackgroudColor =
“Red” Text=”Hello World”>
```

Converts in HTML:

```
<input name="TextBox1" type="text"
style=”background-color:red” Value=”Hello World”>
```



If you see for every request there is a conversion logic which runs and converts the server controls to HTML output.

Control over HTML and HTTP

Drag and drop environment is easy to use but some short of time it will become messy MVC give structure that separate all the stuff right from the beginning.

ASP.NET MVC produces clean, standards-compliant markup. Its built-in HTML helper methods produce standards compliant output, but there is a more significant philosophical change compared with Web Forms.

Testability

Better support for test-driven development (TDD) so unit testing is easier with MVC

Powerful Routing System

The style of URLs has evolved as Web application technology has improved. URLs like this one:

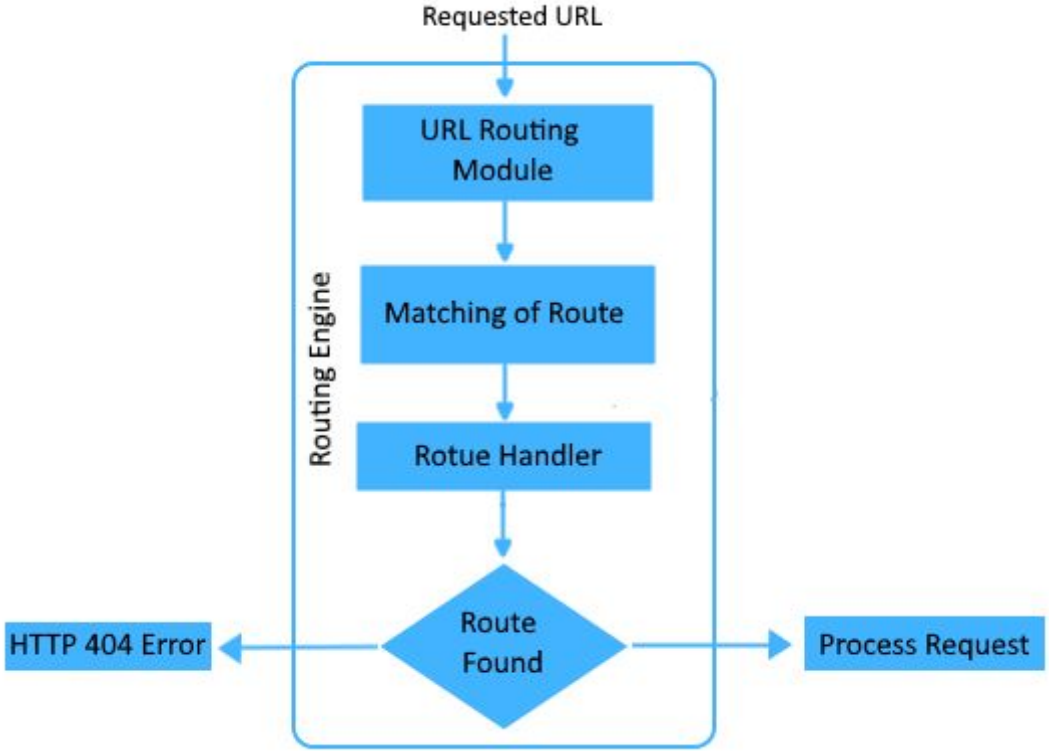
/App_v2/User/Page.aspx?action=show%20prop&prop_id=82742

are increasingly rare, replaced with a simpler, cleaner format like this:

/to-rent/chicago/2303-silver-street

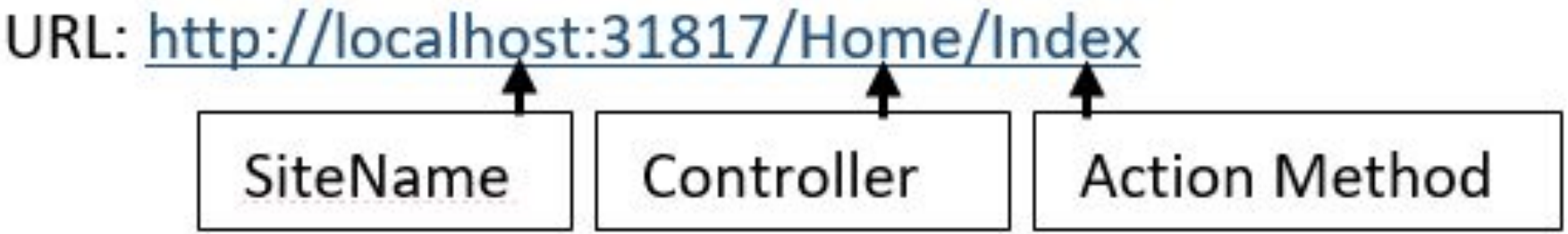
Routing in MVC

Routing is a pattern matching process that monitors the requests and determines what to do with each request. In other words we can say Routing is a mechanism for mapping requests within our MVC application.



Understanding Routes in Detail (Mapping)

When Visual Studio creates the MVC project, it adds some default routes to get us started. You can request any of the following URLs, and they will be directed to the Index action on the HomeController:



You can see and edit your routing configuration by opening the RouteConfig.cs file in the App_Start folder.

RouteConfig.cs file

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

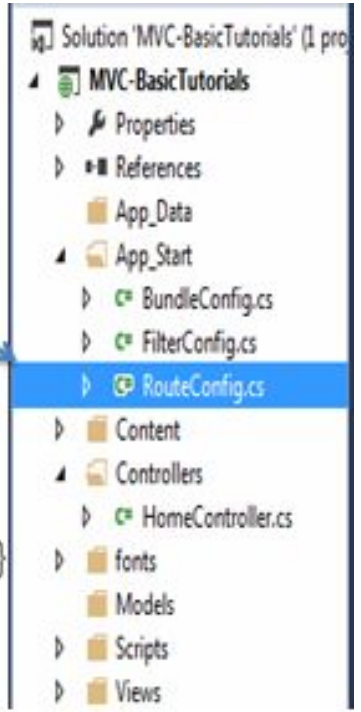
Route to ignore

Route name

URL Pattern

Defaults for Route

RouteConfig.cs



Configuring Routes with Convention Routing

MapRoute Method

```
public static Route MapRoute
    (string name,
    string url,
    object defaults,
    object constraints,
    string[] namespaces);
```

Regular Expression Meta Character

	Metacharacter	Metacharacter name	Meaning
1	^	caret	denote the beginning of a regular expression
2	$	Dollar sign	denote the end of a regular expression or ending of a line
3	[]	Square bracket	check for any single character in the character set specified in []
4	()	Parenthesis	Check for a string. Create and store variables.
5	?	Question mark	check for zero or one occurrence of the preceding character
6	+	Plus sign	check for one or more occurrence of the preceding character
7	*	Multiply sign	check for any number of occurrences (including zero occurrences) of the preceding character.
8	.	Dot	check for a single character which is not the ending of a line
9	|	Pipe symbol	Logical OR
10	\	Escaping character	escape from the normal way a subsequent character is interpreted.
11	!	Exclamation symbol	Logical NOT
12	{}	Curly Brackets	Repeat preceding character

Example

```
routes.MapRoute(
                name: "Home",
                url: "Home/Welcome/{name}/{id}",
        defaults: new { controller = "Home", action = "Welcome",
        name = UrlParameter.Optional, id = UrlParameter.Optional,
        constraint: new {id = @”/d{2}” });

routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional } );
```

Attribute Routing

- Attribute routing is the new type of routing in ASP.NET MVC 5.
- According to the name Attribute Routing, one can suppose that this type of methodology will use Attribute to define routes.

Why do we need Attribute Routing?

- When our application is large enough to have more than two routes, then it will become very complex to make each route separately. For each route, you have to write 5 lines so it will consume your time and disturb your development time.
- Attribute Routing is easy to use and help in mapping the same action method with two routes.
- You don’t have to take care of the routing flow i.e, from most specific to most general. All here is the attribute you use the action method.

Enable Attribute Routing

- If you want to use Attribute Routing, you have to enable it by calling MapMvcAttributeRoutes on the RouteCollection for your app (usually this is done in RouteConfig):

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapMvcAttributeRoutes();
}
```

Add Route Attribute in controller

A [Route] attribute is used at the top of the action method.

File Name : **HomeController.cs**

Syntax:
[Route(*url*)]
Declaration of action method

Example

```
[Route("stud")]
public ActionResult Index()
{
   return View();
}
[Route("stud/detail")]
public string StudDetail()
{
     return "This is Student Detail Page";
}
```

Route with Parameters

Syntax:

```
[Route(url/{parameter_name})]
```

Example:

```
[Route("stud/exam/{sem}/{branch}")]
public string Exam(int sem, string branch)
{
   return "Semester : "+ sem + "<br> Branch : " + branch;
}
```

Optional parameter

Syntax:

```
[Route(url/{parameter_name?})]
```

Example:

```
[Route("stud/exam/{sem?}/{branch?}")]
public string Exam(int sem, string branch)
{
   return "Semester : "+ sem + "<br> Branch : " + branch; }
```

Default value in parameter

Syntax:

```
[Route(url/{parameter_name=value})]
```

```
[Route("stud/exam/{sem=-1}/{branch=no branch}")]
public string Exam(int sem = 1)
{
   return "Semester : "+ sem;
}
```

Route Constraints

Route constraints let you restrict how the parameters in the route template are matched. The general syntax is "{parameter:constraint}".

For example:

```
[Route("users/{id:int}")]
public User GetUserById(int id) { ... }
```

Constraint : alpha
Description : Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)
Example : {x:alpha}

Constraint : length

Description : Matches a string with the specified length or within a specified range of lengths.

Example : {x:length(6)} {x:length(1,20)}

Constraint : min, max

Example : {x:min(1):max(10)}

```
Constraint : regex
Description : Matches a regular expression.
Example : {x:regex(^\d{3}-\d{3}-\d{4}$)}
```

```
Example : {x:bool}, {x:datetime}, {x:decimal}, {x:double},
{x:float}, {x:int}, {x:long}
```

Route Prefix

- **What is RoutePrefix?**
  You may see that many routes have the same portion from its start, it means their prefixes are the same. For example:
- stud/detail
- stud/exam

Both the above URLs have the same prefix which is stud.
So, rather than repeatedly typing the same prefix, again and again, we use RoutePrefix attribute. This attribute will be set at the controller level.

Example

```
[RoutePrefix("stud")]
public class StudentController : Controller
{
   [Route("detail")]
   //Route: stud/detail
       public string StudDetail()
       {     return "This is Student Detail Page"; }
   [Route(“fees”)]
   //Route: stud/Fees
       public string Fees()
        { return "This is Fees detail action method”  }
}
```

Action Results

- It is very important to note that it is the controller's job to tell the ASP.NET MVC Framework what it should do next, but not how to do it.
- This communication occurs through the use of ActionResults, the return values which every controller action is expected to provide.
- Despite the fact that every controller action needs to return an ActionResult, you will rarely be creating them manually.

System.Web.Mvc.Controller base class

Helper Methods

1. **Content()**

- Returns a **ContentResult** that renders arbitrary text, e.g., “Hello, world!”. Content result returns different content's format to view.
- MVC returns different format using content return like **HTML format, Java Script** format and any other format.
- Example:

```
return Content(
        "<script>alert('This is ContentResult Action
        Result');</script>");
```

System.Web.Mvc.Controller base class

Helper Methods

2. **File()**
- Returns a **FileResult** that renders the contents of a file, e.g., a PDF, docx, jpg etc.

3. **HttpNotFound()**
- Returns an **HttpNotFoundResult** that renders a 404 HTTP status code response.
- Example:
  ```
  return HttpNotFound();
  ```

System.Web.Mvc.Controller base class

Helper Methods

4. **PartialView()**
- Returns a PartialViewResult that renders only the content of a view (i.e., a view without its layout).

5. **Redirect()**
- Returns a RedirectResult that renders a 302 (temporary) status code to redirect the user to a given URL, e.g., “302 http://www.ebuy.com/auctions/recent”.

6. **View()**
- Returns a ViewResult that renders a view.

If any of the available ActionResult works for you, you are free to create your own!

- Filter
- Authentication Filters

Understanding ASP.NET MVC Filters and Attributes

- ASP.NET MVC provides a simple way to inject your piece of code or logic either before or after an action is executed.
- This is achieved by decorating the controllers or actions with ASP.NET MVC attributes or custom attributes.
- An attribute or custom attribute implements the ASP.NET MVC filters(filter interface) and can contain your piece of code or logic.

When to use Filters?

1. Custom Authentication
2. Error handling or logging
3. User Activity Logging
4. Data Caching
5. Data Compression

Types of Filters

1. Action filters
2. Exception filters
3. Authentication filters and Authorization filters (New in ASP.NET MVC5)
4. Result filters

ASP.NET MVC - Action Filters

Action filter executes before and after an action method executes.

Action filter attributes can be applied to an individual action method or to a controller.

When action filter applied to controller then it will be applied to all the action methods in that controller.

Output Catch (Action Filter)

OutputCache is a built-in action filter attribute that can be apply to an action method for which we want to cache the output.

Syntax: [OutputCatch(int DurationValue)]

For example, output of the following action method will be cached for 100 seconds.

Example:

```
[OutputCache(Duration=100)]

public ActionResult Index()

{      return View();  }
```

Handle Error (Exception Filter)

- As per the C# we can use try catch to handle exception.
- MVC framework provide in build filter to handle exception this filter known as HandleError
- HandleError is type of ExceptionFilter.

Steps to enable HandleError

- Enable custom error in web.config file
  <System.Web>
    <CustomError mode="On"></CustomError>
  </System.Web>
- Create Error.cshtml file in shared folder
- Use HandelError attribute in Controller/Action/Globally.

Example

```
[HandleError]
Public ActionResult Index()
{
   throw new Exception(“Something is worng”);
}
```

- Validation Attributes
- How viewbag falls short
- Understanding Viewbag, viewdata, and view data dictionary, viewmodels,
- Samples layouts View state
- Specifying a partial view

Implement Data Validation in MVC

ASP.NET MVC uses DataAnnotations attributes to implement validations. **DataAnnotations** includes built-in validation attributes for different validation rules, which can be applied to the properties of model class.

ASP.NET MVC framework will automatically enforce these validation rules and display **validation messages in the view.**

Namespace :

```
using System.ComponentModel.DataAnnotations;
```

DataAnnotations Attributes of Validation

- **Required**
- **Indicates that the property is a required field**
  - **Example:**
  - **(in model)**

```
[Required(ErrorMessage="Please enter First Name")]
public string FirstName { get; set; }
```

  - **(in view)**

```
@Html.TextBoxFor(m => m.FirstName)
@Html.ValidationMessageFor(m => m.FirstName)
```

- **StringLength**
- Defines a maximum length for string field.
- Specifies how many number of characters are allowed for that particular property. Example:
  - [StringLength(10,MinimumLength = 1,ErrorMessage ="")]
  - public string FirstName { get; set; }
- **MaxLength and MinLength**
  - Example
  - [MaxLength(3, ErrorMessage = “”)]
  - [MinLength(1 , ErrorMessage = “”)]
  - publicint Marks { get; set; }

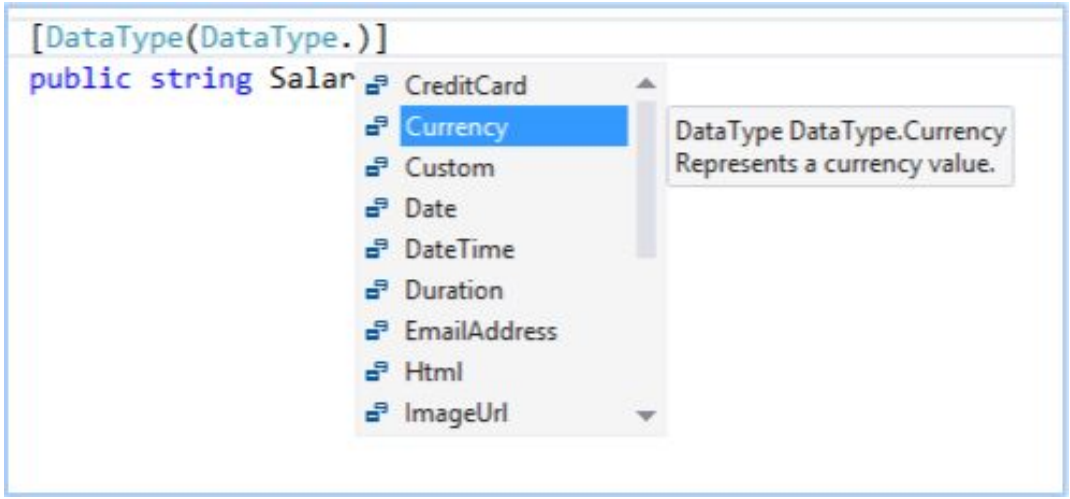
- **Range**
- **Defines a maximum and minimum value for a numeric field. Example:**

```
[Range(1,100, ErrorMessage = “”)]

public int Age { get; set; }
```

- **RegularExpression**
- **Specifies that the field value must match with specified Regular Expression. Example:**

```
[RegularExpression(@"\d{10}", ErrorMessage="Please enter valid mobile number")]

public string Mobile { get; set; }
```

- **DataType**
- This attribute specifies the type of the property like emailId OR phoneNumber



- [DataType(DataType.EmailAddress)]
- public string Email { get; set; }

  or
- [EMailAddress]
- public string Email { get; set; }

- **Compare**
- We have to set the compare attribute in the property which we want to be compared to any other property. We pass the property name to this compare attribute.

- Example:
- [Compare("confirmPassword",ErrorMessage = "Password mismatch")
- public string Password {get; set;}

- public string confirmPassword {get; set; }

Attributes of MVC

- **Display**
- This attribute is used to enhance the current property for which the attribute needs to be applied.
- Example:

```
[Display(Name = "First Name",
Description="First Name of the person")]
public string FirstName { get; set; }
```

- **DisplayFormat**
- This attribute is used to specify how the fields should be displayed and formatted
- [DisplayFormat(DataFormatString = "{0:C}")]
- public Decimal ListPrice { get; set; }

Example of Validation Attributes

- Model Properties:
- **Name** (should be minimum 5 char or maximum 50 character)
- **Age** (should be in between 18 to 60)
- **Mobile** (should be in 10 digit)
- **Email** (format checking)
- **WebSite** (ulr of web site)

```
Class ValidationDemo
{
   [StringLength(50,MinimumLength = 5,ErrorMessage = "Name
length   should    be 5 to 50)]
   public string Name {get;set;}

   [Range(18,60,ErrorMessage = "Age should be in between 18 to
60")]
   public int Age {get; set; }

   [RegularExpression(@"/d{10}",ErrorMessage = "Invalid Mobile
Number")]
   public string Mobile { get; set;}
```

```
   [RegularExpression(@"^([a-zA-Z0-9\-\.]+)@([a-zA-Z0-9\-\.]
+)\.([a- zA-Z]{2,5})$",ErrorMessage = "Invalid e-mail id")]
   public string EMail {get; set; }

   [DataType(DataType.Url)]
   [RegularExpression(@"^((https?|ftp|smtp):\/\/)?(www.)?[a-
z0-9]+\.[a-  z]+(\/[a-zA-Z0-9#]+\/?)*$",ErrorMessage =
"Invalid Web site  address")]
   public string WebSite { get; set; }
}
```

ASP.NET MVC - ViewBag

- ViewBag can be useful when you want to transfer temporary data (which is not included in model) from the controller to the view.


@ViewBag.Name
ViewBag
ViewBag.Name = "Bill"
View
Controller

ASP.NET MVC - ViewData

- ViewData is similar to ViewBag. It is useful in transferring data from Controller to View.
- ViewData is a dictionary which can contain key-value pairs where each key must be string.
- The following figure illustrates the ViewData.


@ViewData["Name"]
ViewData
ViewData["Name"] = "Bill"
View
Controller

Example

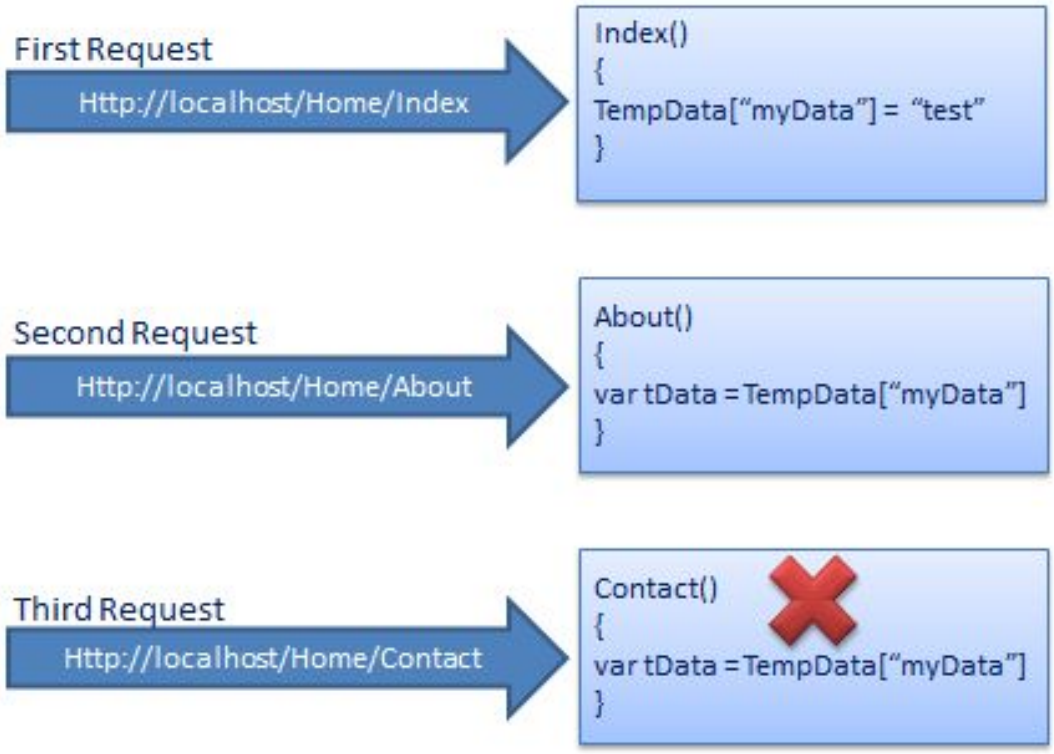
```
ViewData.Add("Id", 1);

ViewData.Add(new KeyValuePair<string,
object>("Name", "Bill"));
```

ASP.NET MVC - TempData

- TempData is useful when you want to transfer non-sensitive data from one action method to another action method of the same or a different controller as well as redirects.

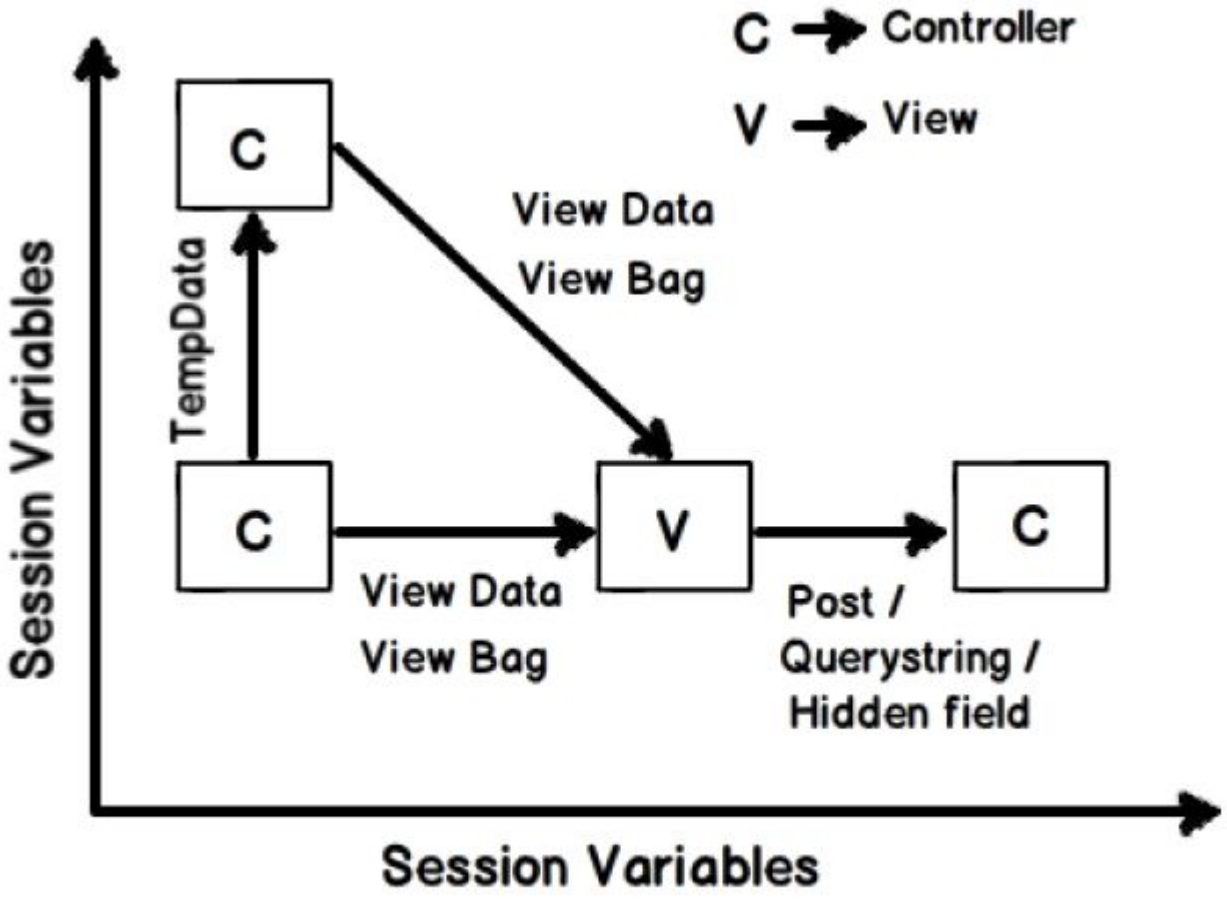
- You can add a key-value pair in TempData.

- TempData will be cleared out after second request.
- Call TempData.Keep() to retain TempData values in a third consecutive request.



Example

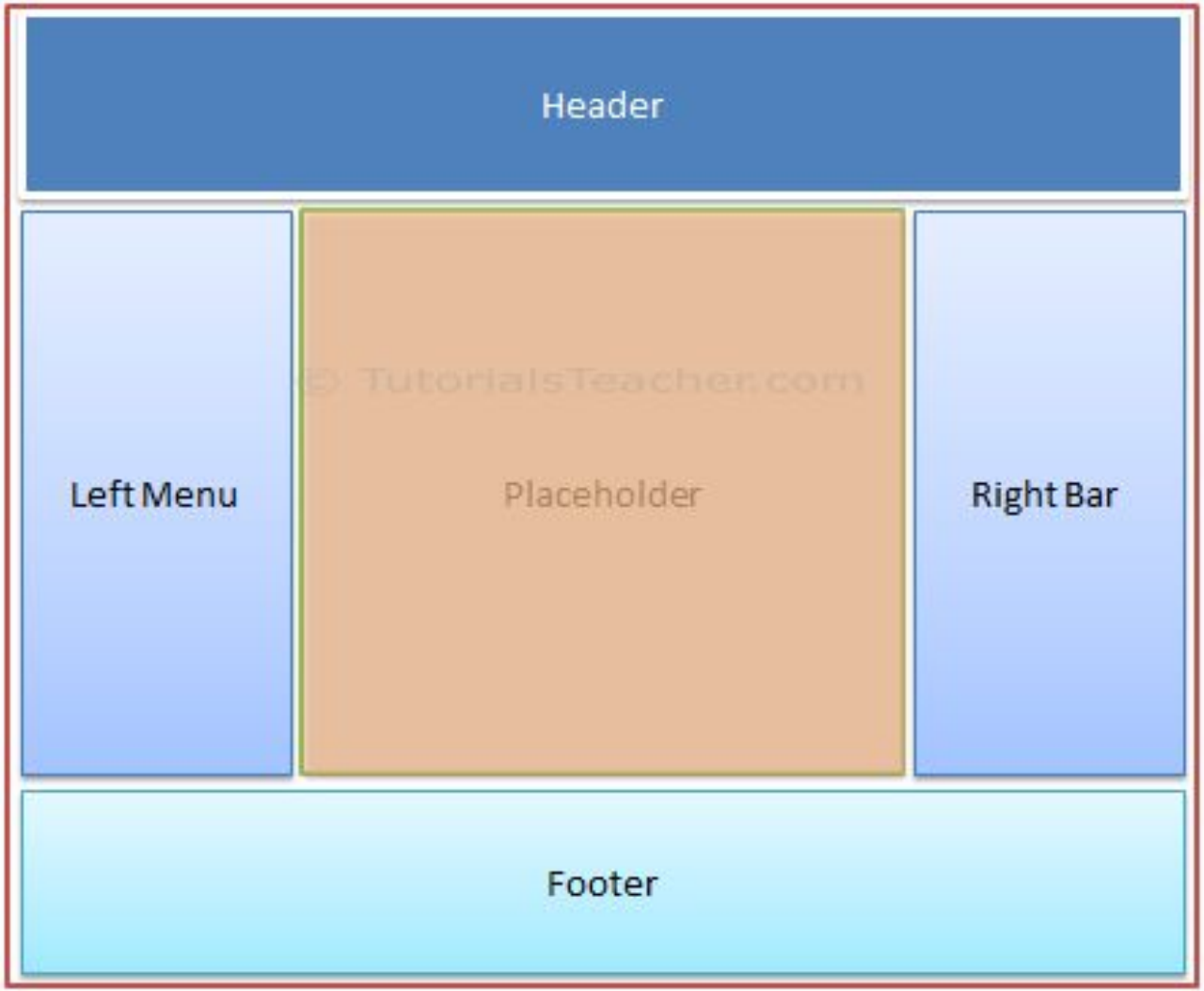
```
public ActionResult Index()
{
   TempData["name"] = "Test data";
   TempData["age"] = 30;
   return View();
}
public ActionResult About()
{
   string userName;
   userName = TempData["name"].ToString();
}
```



difference between tempdata ,viewdata and viewbag

Layout in MVC (Bootstrap overview)

- An application may contain common parts in the UI which remains the same throughout the application such as the **logo, header, left navigation bar, right** bar or **footer section**.
- ASP.NET MVC introduced a Layout view which contains these common UI parts, so that we don't have to write the same code in every page.
- The layout view is same as the master page of the ASP.NET webform application.


Header
Left Menu
Placeholder
Right Bar
Footer

- The razor layout view has same extension as other views, **.cshtml** or **.vbhtml**.
- Layout views are shared with multiple views, so it must be stored in the Shared folder.
- For example, when we created our first MVC application in the previous section, it also created **_Layout.cshtml** in the **Shared folder**.

Razor Syntax in _Layout.cshtml

- **@ViewBag.Title** - The page title will be inserted here.
- **@Url.Content():** Content() method is a method of UrlHelper class. It converts a virtual (relative) path to an application absolute path.
- **@Html.ActionLink():** The easiest way to render an HTML link in is to use the HTML.ActionLink() helper.
- **@RenderBody()** - The page content will be rendered here.

Partial View

- Partial view is a reusable view, which can be used as a child view in multiple other views.
- It eliminates duplicate coding by reusing same partial view in multiple places. You can use the partial view in the layout view, as well as other content views.

Render Partial View

- You can render the partial view in the parent view using html helper methods:
- Html.Partial() and Html.RenderPartial(): helper method renders the specified partial view.
  Example: Html.Partial(string partialViewName)
  Example: Html.RenderPartial(String partialViewName)
- RenderAction()
  Example: Html.RenderAction("Category","Home")

References

Web Site:
https://docs.microsoft.com/en-us/aspnet/mvc/
https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm
https://www.guru99.com/mvc-tutorial.html
https://en.wikipedia.org/wiki/ModelViewController
https://www.guru99.com/mvc-tutorial.html
https://www.geeksforgeeks.org/mvc-design-pattern/

Book:
Pro ASP.NET MVC 5.0