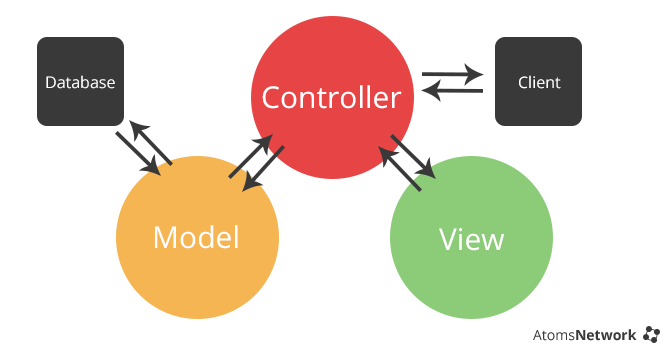
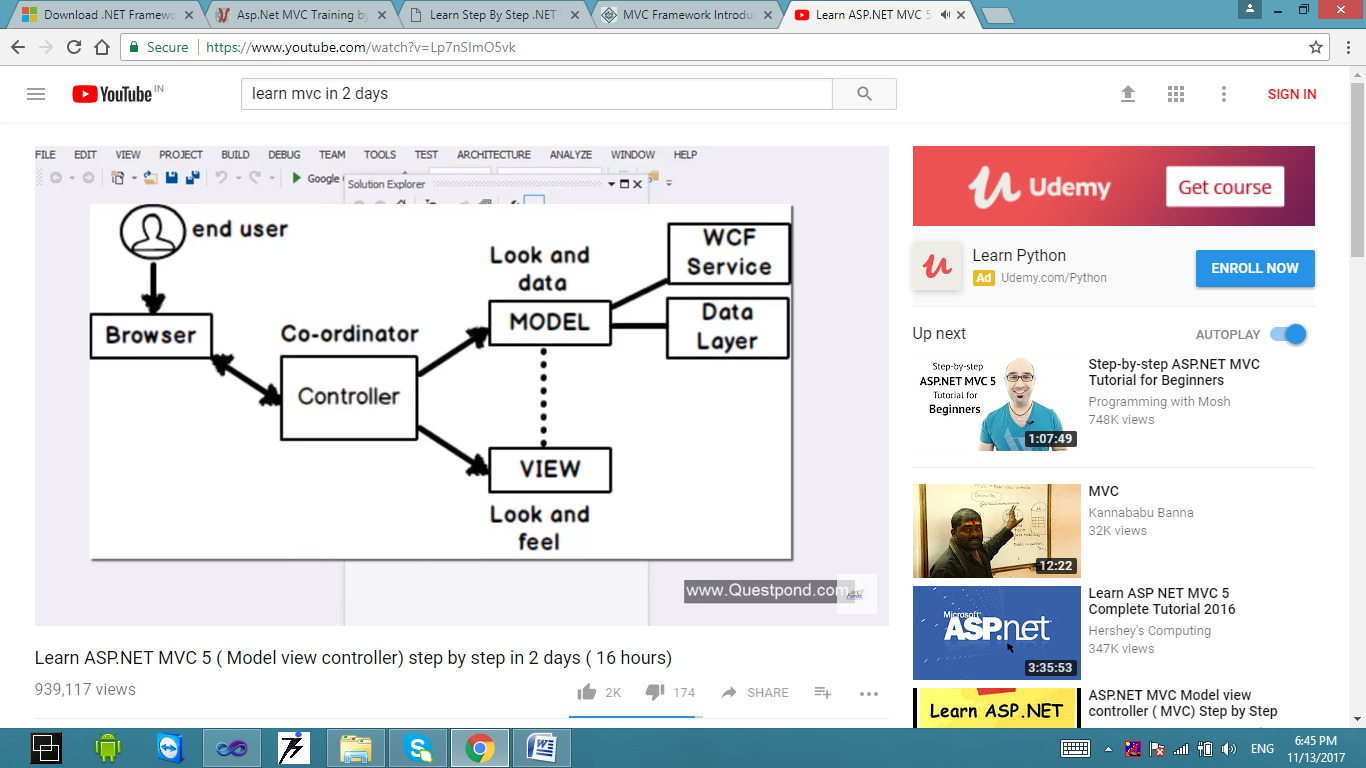
**Topics**

* Basics of MVC
* Why MVC? – 3 tier vs MVC
* ASP.NET MVC Life Cycle
* ASP.NET MVC Versions
* First MVC application Demo with Controller and View
* Controller with ActionMethod and ActionResult
* Routing in MVC
* Razor in MVC
* MVC application Demo to pass data amongst Controller and View
* Comparison between ViewData , ViewBag, TempDate and Session
* MVC application Demo with Model, View and Controller
* HTML Helper in MVC

**MVC:**

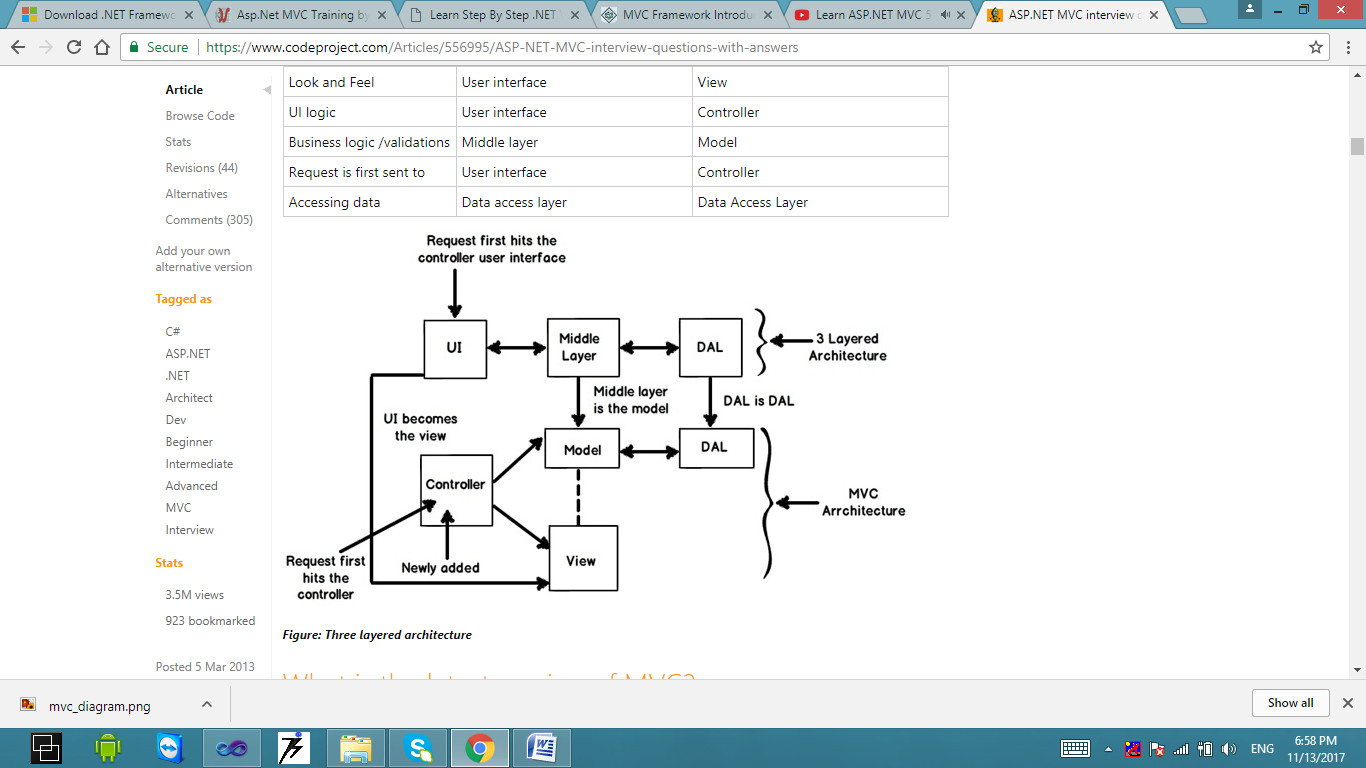
* MVC stands for **Model-View-Controller.**
* The Model-View-Controller (MVC) is an architectural/design pattern that separates an application into three main logical components: the model, the view, and the controller. It is architecture for developing interactive applications where there would be a user interaction involved and event handling would occur.
* **Models:** These are the classes that contain data. They can practically be any class that can be instantiated and can provide some data. It includes all of an application’s validation logic, business logic and data access.
* **Views:** These are simple pages containing HTML and C# code that will use the server side object. It contains logic for rendering Graphical Representation/ HTML output. Typically it create UI with data from model.
* **Controllers:** These are the classes that will be invoked on user requests. The main tasks of these controllers are to generate the model class object and to call view.





**3-tier vs MVC**

MVC is an evolution of a three layered traditional architecture. Many components of the three layered architecture are part of MVC.

****

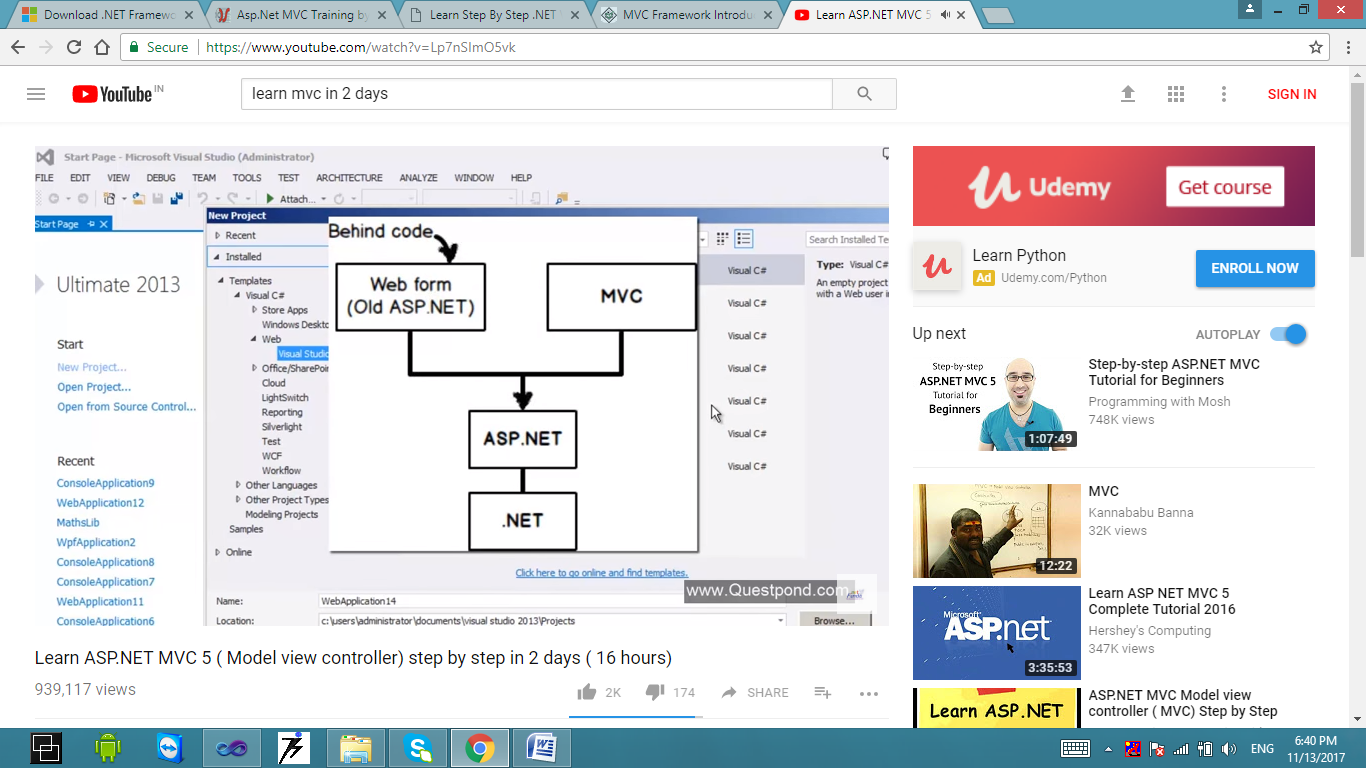
|  |  |  |
| --- | --- | --- |
| **Functionality** | **Three layered / tiered architecture** | **Model view controller architecture** |
| Look and Feel | User interface | View |
| UI logic | User interface | Controller |
| Business logic /validations | Middle layer | Model |
| Request is first sent to | User interface | Controller |
| Accessing data | Data access layer | Data Access Layer |

**Why MVC over Web Form?**

|  |  |
| --- | --- |
|  | **Disadvantages of Web Form**   * View Based Architecture * Behind code is not reusable * Fixed response type * Fix combination of View and Data * Separate Unit Testing Program |

**ASP.NET MVC Framework**

ASP.NET MVC is architecture to develop ASP.NET web applications in a different manner. Web applications developed with ASP.NET MVC are even more SEO (Search Engine) friendly.



**Model:**

* MVC Model is basically a C# or VB.NET class.
* A model is accessible by controller and view.
* A model can be used to pass data from Controller to View.

**View:**

* View is CSHTML page without having code behind file.
* All page specific HTML generation and formatting can be done inside view.
* One can use inline (server tag) to develop dynamic page.
* A request to view can be made only from a controller’s action method.

**Controller:**

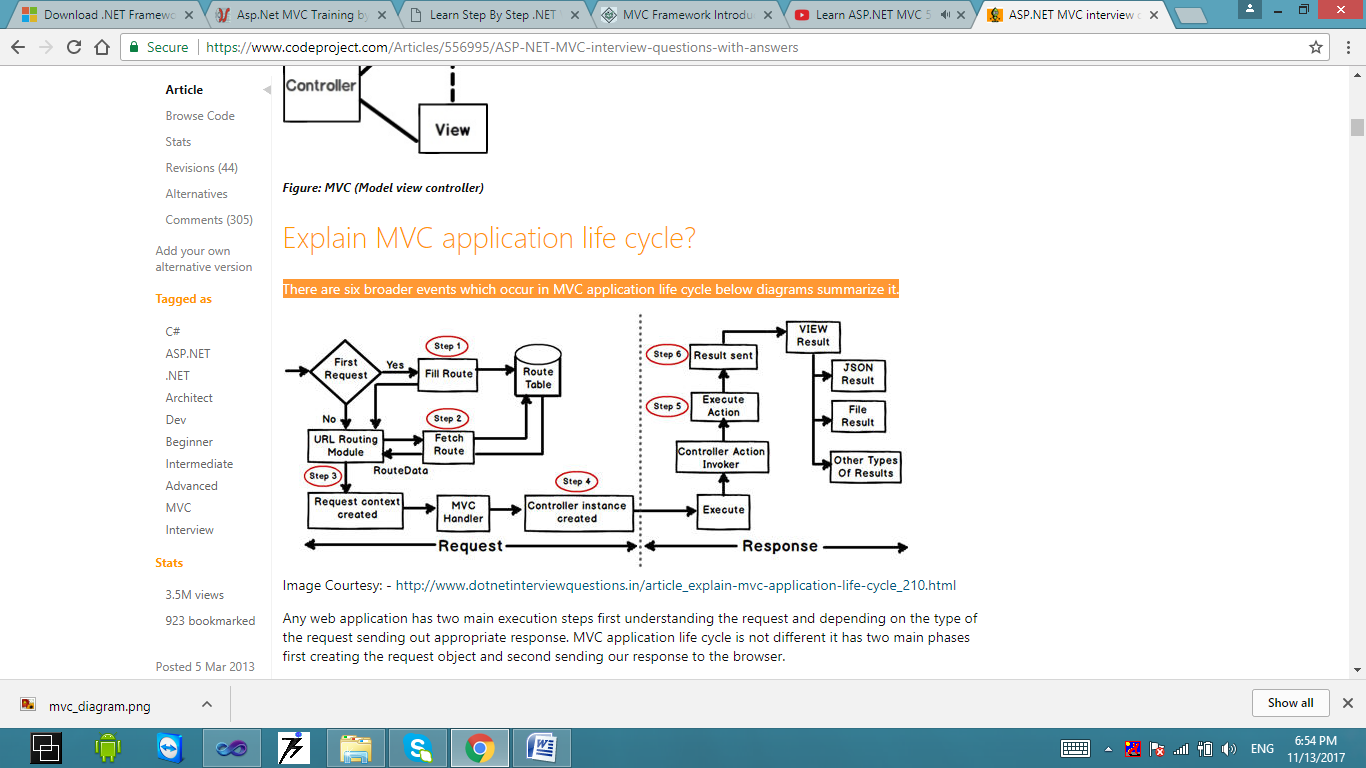
* Controller is basically a C# or VB.NET class which inherits System.Web.Mvc.Controller.
* Controller is heart/coordinator of the MVC architecture.
* Inside controller’s class action methods can be implemented which are responsible for responding to browser or calling View.
* Controller can access and use model class to pass data to views.

**Advantages:**

* It manages the complexity by dividing an application into the model, the view and the controller.
* It does not use view state or server-base forms or server control.
* It uses a front controller pattern that processes web application requests through single controller. This enables you to design an application that supports a rich routing infrastructure.
* RESTful/ User friendly URL’s and this enables SEO.
* Clean HTML and easy integration with Javascript and JQuery.
* It provides better support for test-driven development.

**ASP.NET MVC Application Life Cycle**

There are six broader events which occur in MVC application life cycle below diagrams summarize it.



Any web application has two main execution steps first understanding the request and depending on the type of the request sending out appropriate response. MVC application life cycle is not different it has two main phases first creating the request object and second sending our response to the browser.

**Creating the request object: -**The request object creation has four major steps. Below is the detail explanation of the same.

* **Step 1 Fill route: -** MVC requests are mapped to route tables which in turn specify which controller and action to be invoked. So if the request is the first request the first thing is to fill the route table with routes collection. This filling of route table happens in the global.asax file.
* **Step 2 Fetch route: -** Depending on the URL sent “UrlRoutingModule” searches the route table to create “RouteData” object which has the details of which controller and action to invoke.
* **Step 3 Request context created: -** The “RouteData” object is used to create the “RequestContext” object.
* **Step 4 Controller instance created: -** This request object is sent to “MvcHandler” instance to create the controller class instance. Once the controller class object is created it calls the “Execute” method of the controller class.

**Creating Response object: -** This phase has two steps executing the action and finally sending the response as a result to the view.

**ASP.NET MVC Versions**

|  |  |
| --- | --- |
| **Release Date** | **ASP.NET MVC Version** |
| 13 March 2009 | ASP.NET MVC 1.0 |
| 10 March 2010 | ASP.NET MVC 2.0 |
| 13 January 2011 | ASP.NET MVC 3 |
| 15 August 2012 | ASP.NET MVC 4 |
| 30 May 2013 | ASP.NET MVC 4 4.0.30506.0 (4.5) |
| 17 October 2013 | ASP.NET MVC 5 |
| 17 January 2014 | ASP.NET MVC 5.1 |
| 10 February 2014 | ASP.NET MVC 5.1.1 |
| 4 April 2014 | ASP.NET MVC 5.1.2 |
| 22 June 2014 | ASP.NET MVC 5.1.3 |
| 1 July 2014 | ASP.NET MVC 5.2 |
| 28 August 2014 | ASP.NET MVC 5.2.2 |
| 9 January 2015 | ASP.NET MVC 5.2.3 |

**MVC 2**

* Client-Side Validation
* Templated Helpers
* Areas
* Asynchronous Controllers
* Html.ValidationSummary Helper Method
* DefaultValueAttribute in Action-Method Parameters
* Binding Binary Data with Model Binders
* DataAnnotations Attributes
* Model-Validator Providers
* New RequireHttpsAttribute Action Filter
* Templated Helpers
* Display Model-Level Errors

**MVC 3**

* Razor
* Readymade project templates
* HTML 5 enabled templates
* Support for Multiple View Engines
* JavaScript and Ajax
* Model Validation Improvements

**MVC 4**

* ASP.NET Web API
* Refreshed and modernized default project templates
* New mobile project template
* Many new features to support mobile apps
* Enhanced support for asynchronous methods

**MVC 5**

* One ASP.NET
* Attribute based routing
* Asp.Net Identity
* Bootstrap in the MVC template
* Authentication Filters
* Filter overrides

**MVC 6**

* ASP.NET MVC and Web API have been merged in to one.
* Dependency injection is inbuilt and part of MVC.
* Side by side - deploy the runtime and framework with your application
* Everything packaged with NuGet, Including the .NET runtime itself.
* New JSON based project structure.
* No need to recompile for every change. Just hit save and refresh the browser.
* Compilation done with the new Roslyn real-time compiler.
* vNext is Open Source via the .NET Foundation and is taking public contributions.
* vNext (and Rosyln) also runs on Mono, on both Mac and Linux today.

**MVC Core**

* .NET Core functionality

**Controller Overview**

The ASP.NET MVC framework maps URLs to classes that are referred to controllers.

Controllers

* Process incoming requests
* Handle user input and interactions
* Execute appropriate application logic

The Controller class inherits from Controllerbase and is the default implementation of a controller.

The controller class is responsible for the following processing stage:

* Locating the appropriate action method to call and validate that it can be called.
* Getting the values to be used as the action method’s parameters.
* Handling all errors that might occur during the execution of the action method.
* Providing the View for rendering ASP.NET pages to browser.

**ActionMethods and ActionResult object**

ASP.NET MVC application is organized around controllers and action methods. The controller defines action methods. Controller can include as many action methods as needed.

* Action methods typically have a one-to-one mapping with user interactions. Example of user interaction includes entering a URL into the browser, clicking a link, and submitting a form. Each of these user interactions cause a request to be sent to the server. In each case, the URL of the request included information that the MVC framework uses to invoke an action method.
* Most action methods return and instance of a class that derives from ActionResult. The ActionResult class is the base for all action results. However, there are different action result types, depending on the task that the action method is performing. For example, the most common action is call the View Method. The View method returns an instance of the ViewResult class, which is derived from ActionResult.
* We can create action methods that return an object of any type, such as a string, an integer or a Boolean value. These return types are wrapped in an appropriate ActionResult type before they are rendered to the response stream.

|  |  |  |
| --- | --- | --- |
| **Action Result** | **Action Method** | **Description** |
| ViewResult | View | Render a view as a Web Page. |
| PartialViewResult | PartialView | Render a partial view, which defines a section of view that can be rendered inside another view. |
| RedirectResult | Redirect  RedirectPermanent | Redirect to another action method by using its URL.  return Redirect(“/Home/Contact”); |
| RedirectToResultResult | RedirectToAction  RedirectToRoute | Redirect to another action method.  Return RedirectToAction(“Contact”)  Return RedirectToAction(“sayHello”,”Demo”, new fname=”a”, lname=”b”)  Return RedirectToRoute(“sayHelloRoute”) |
| ContentResult | Content | Return a user-defined content type.  return Content(“content detail”);  return Content(“<name><fname>ABC</fname><lname>XYZ</lname></name>”,”text/xml”); |
| FileContentResult  FilePathResult  FileStreamResult | File | Returns binary output to write to the response.  Return File(“filename.txt”,”text/plain”)  Return File(“filename.txt”,”text/plain”,”dwnName.txt”) |
| JavascriptResult | JavaScript | Return JavaScript content  Return JavaScript(“alert(‘js demo’)”)  <script src=”Home/Contact”></script> |
| JsonResult | Json | Return a serialized JSON object  Person p = new Person(){s=”n”};  Return Json(p,JsonRequestBehavior.Allowget) |
| EmptyResult | Null | Returns Empty Page  Return null |
| HttpNotFoundResult | HttpNotFound | Returns 404 error  Return HttpNotFounf() |
| HttpStatusCodeResult | -- | Return the specified status code and description  Return new HttpStatusCodeResult(System.Net.HttpStatusCode.BadRequest,”Bad Request” ) |
| HttpUnauthorizedResult | -- | Return Unauthorized Http Response |

**Routing in MVC**

Routing helps you to define a URL structure and map the URL with the controller.

The route mapping code is written in "RouteConfig.cs" file and registered using "global.asax" application start event.

This RegisterRoutes method is called by the Global.ascx when the application is started. The Application\_Start method under Global.ascx calls this MapRoute function which sets the default Controller and its action (method inside the Controller class).

For instance let’s say we want that when a user types “*http://localhost/View/ViewCustomer/*”, it goes to the “Customer” Controller and invokes the DisplayCustomer action. This is defined by adding an entry in to the routes collection using the maproute function. Below is the underlined code which shows how the URL structure and mapping with controller and action is defined.

routes.MapRoute(

"View", *// Route name*

"View/ViewCustomer/{id}", *// URL with parameters*

new { controller = "Customer", action = "DisplayCustomer",

id = UrlParameter.Optional }); *// Parameter defaults*

In MVC 5 route mapping is also done by attribute which is known as attribute based routing.

public class HomeController : Controller

{

[Route("Users/about")]

[Route("Users/WhoareWe")]

[Route("Users/OurTeam")]

[Route("Users/aboutCompany")]

public ActionResult GotoAbout()

{

return View();

}

}

**Advantage of attribute based routing**

Most of the time developers code in the action methods. Developers can see the URL structure right upfront rather than going to the “routeconfig.cs” and see the lengthy codes. For instance in the below code the developer can see right upfront that the “GotoAbout” action can be invoked by four different URL structure.

This is much user friendly as compared to scrolling through the “routeconfig.cs” file and going through the length line of code to figure out which URL structure is mapped to which action.

**Razor in MVC**

Razor is a markup syntax that lets you embed server-based code (Visual Basic and C#) into web pages.

Server-based code can create dynamic web content on the fly, while a web page is written to the browser. When a web page is called, the server executes the server-based code inside the page before it returns the page to the browser. By running on the server, the code can perform complex tasks, like accessing databases.

Razor is based on ASP.NET, and designed for creating web applications. It has the power of traditional ASP.NET markup, but it is easier to use, and easier to learn.

<ul>  
@for (int i = 0; i < 10; i++) {  
<li>@i</li>  
}  
</ul>

As per Microsoft, Razor is more preferred because Razor is clean, lightweight, and syntaxes are easy as compared to ASPX.

For example, in ASPX to display simple time, we need to write:

<%=DateTime.Now%>

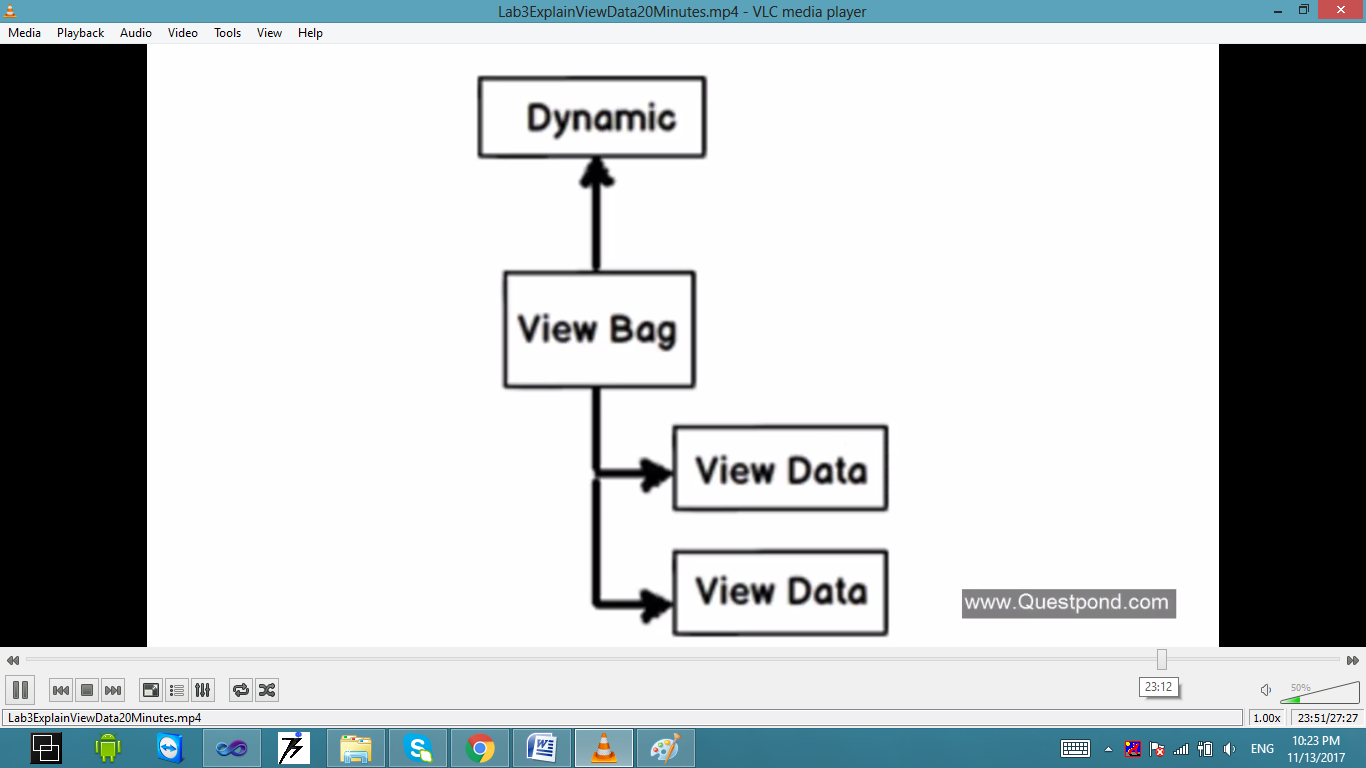
In Razor, it’s just one line of code:

@DateTime.Now

**Maintain Data [State Management] in MVC**

**ViewData:** Helps to maintain data when you move from controller to view.

**ViewBag:** It’s a dynamic wrapper around view data. When you use Viewbag type, casting is not required. It uses the dynamic keyword internally.

****

**TempData:** Helps to maintain data when you move from one controller to another controller or from one action to another action. In other words when you redirect, tempdata helps to maintain data between those redirects. It internally uses session variables.

TempData value will not be persisted for the next request

* **Condition1:** If value of TempData is read normally.

TempData value will be persisted for the next request

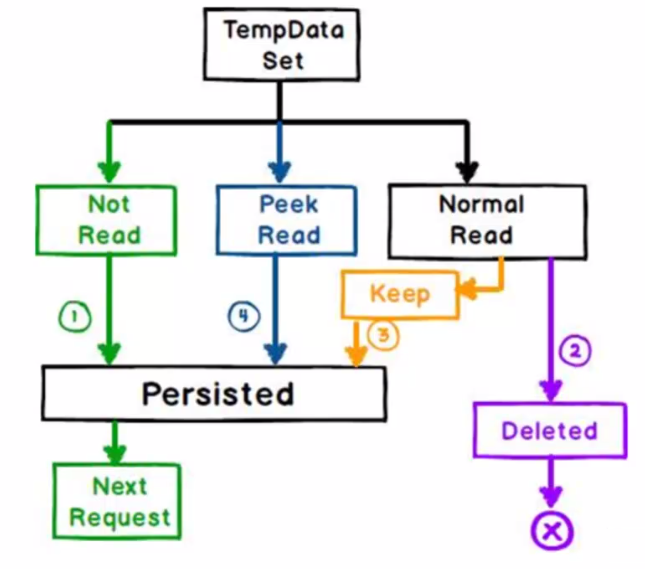
* **Condition1:** If value of TempData is not read.
* **Condition2:** If value of TempData is read and call the Keep method of TempData.

@TempData["MyData"];

TempData.Keep("MyData");

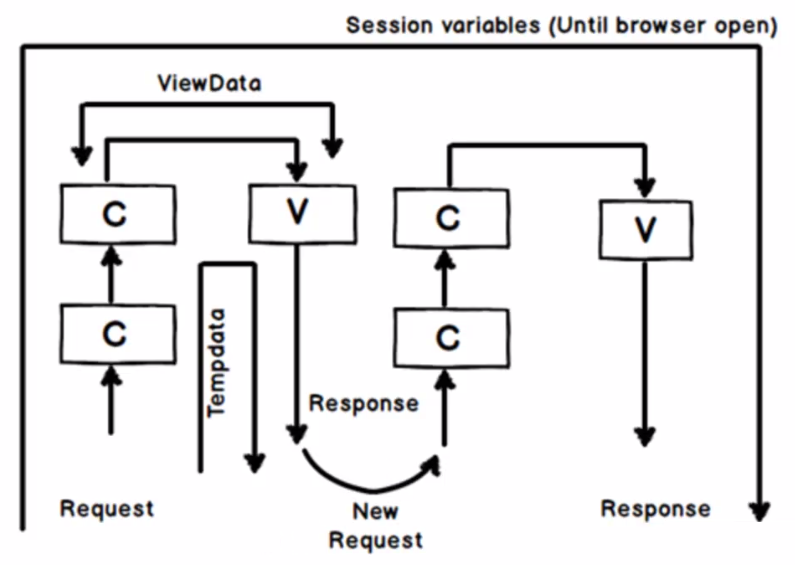
* **Condition3:** If value of TempData is read by Peek method of TempData.

string str = TempData.Peek("Td").ToString();

****

**Session:** By using session variables we can maintain data from any entity to any entity and for multiple requests.

**When to use which State management technique?**



**HTML Helper in MVC**

HTML helpers help you to render HTML controls in the view. For instance if you want to display a HTML textbox on the view , below is the HTML helper code.

@Html.TextBox("CustomerCode")

For checkbox below is the HTML helper code. In this way we have HTML helper methods for every HTML control that exists.

@Html.CheckBox("Married")

[**http://www.tutorialsteacher.com/mvc/html-helpers**](http://www.tutorialsteacher.com/mvc/html-helpers) **[HTML Helper]**