Asp. Net MVC 5

@ankurpathakg

Coverage

Day 1 –

- Define Concept of MVC
- Understand & Implement Controller
  - Action methods
  - Action Parameters
  - Action Results
  - Routing & Attribute Routing
  - Action Filters
- Understand & Implement Views
  - Razor View
  - Razor View Scaffolding
  - HTML Helpers
  - Edit, Update, Delete Data using Views
  - Partial Views
  - Dynamic v. Strongly Typed Views
- Implement Models
- Using Objects as Model

Day 2 –

- Share Data from Controller to View using
  - ViewData
  - ViewBag
  - TempData
  - Session
- Implement Models
  - Using Entity Framework 6 as Model (Code First Approach)
  - Using ViewModel
- Apply Annotation/Validation
  - DataTypeAttribute
  - DisplayAttribute
  - DiplayFormat Attribute
  - RequiredAttribute
  - StringLengthAttribute
- Bundling/Minification Support

Introduction to MVC

MVC

ASP.NET MVC is not meant to replace web forms. It is simply an alternative you can take advantage of based on your specific needs.

MVC Definition

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

This is very complicated.

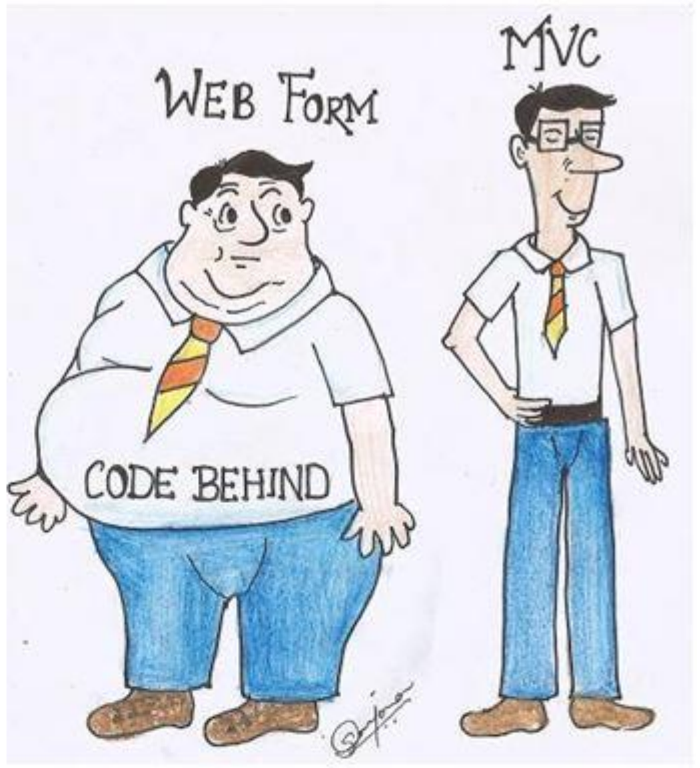
MVC What ?

- The Microsoft ASP.NET model-view-controller (MVC) represents an alternative architecture to the standard ASP.NET web form model.
- This alternative is based on the common MVC software design pattern that separates the layers of your website into user interface (view), data and business logic (model), and input handling (controller).
- This separation through MVC results in loosely coupled objects that are highly testable, can be developed independently from one another, and offer granular control of your application code.
- The model that uses postback and ViewState is nonexistent with ASP.NET MVC.
- MVC uses a routing engine and a set of controllers that provide you with deeper control over your processing and output.
- There are, however, a few familiar items that are preserved in ASP.NET MVC, such as master pages, style sheets, membership, and standard page markup.
- But almost everything else is different, including the process of a page through its life cycle.

MVC Why ?

- ❑ Separation of concerns is achieved as we are moving the code-behind to a separate class file. By moving the binding code to a separate class file we can reuse the code to a great extent.

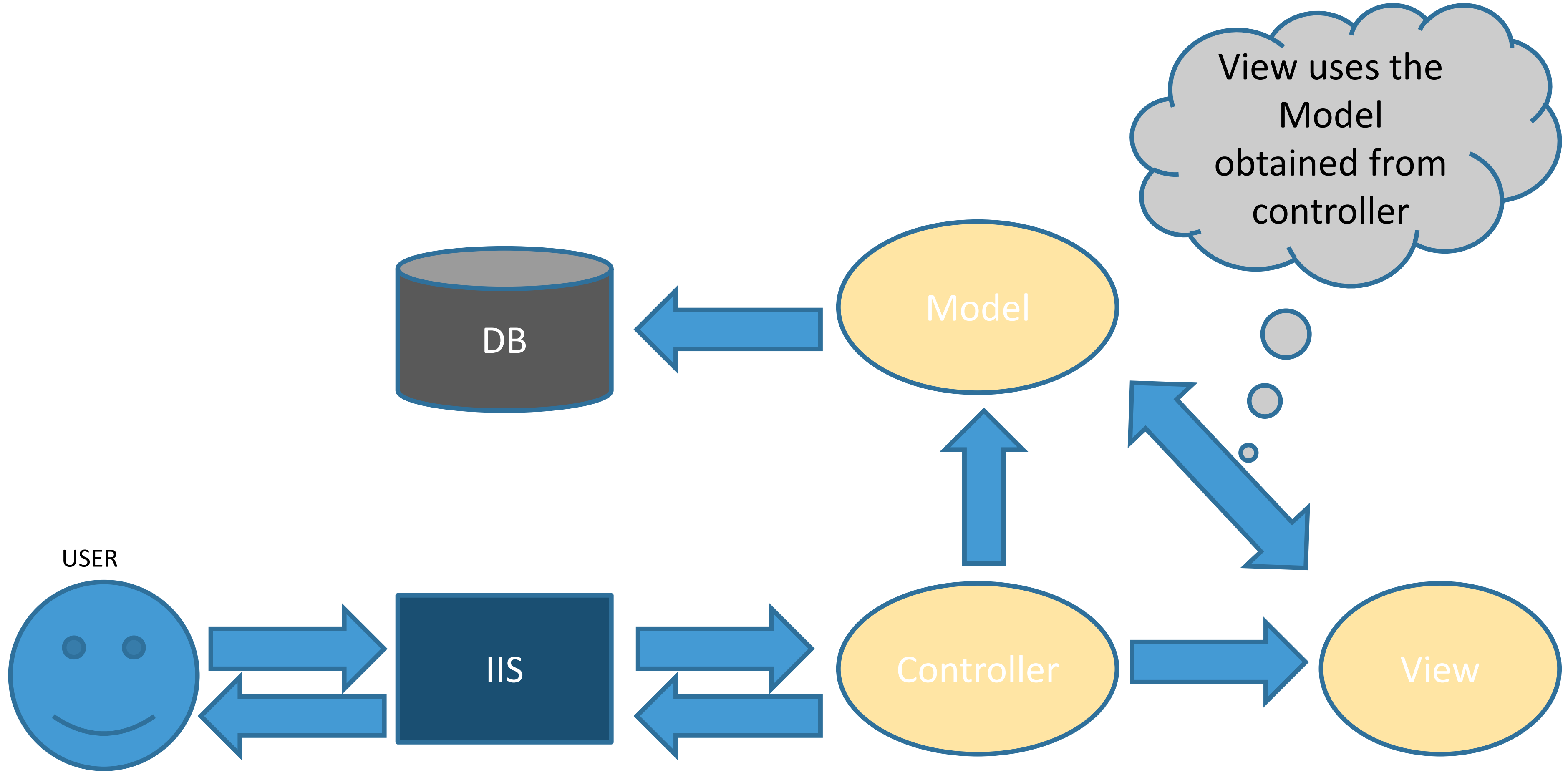
- ❑ Automated UI testing is possible because now the behind code (UI interaction code) has moved to a simple .NET class. This gives us opportunity to write unit tests and automate manual testing.



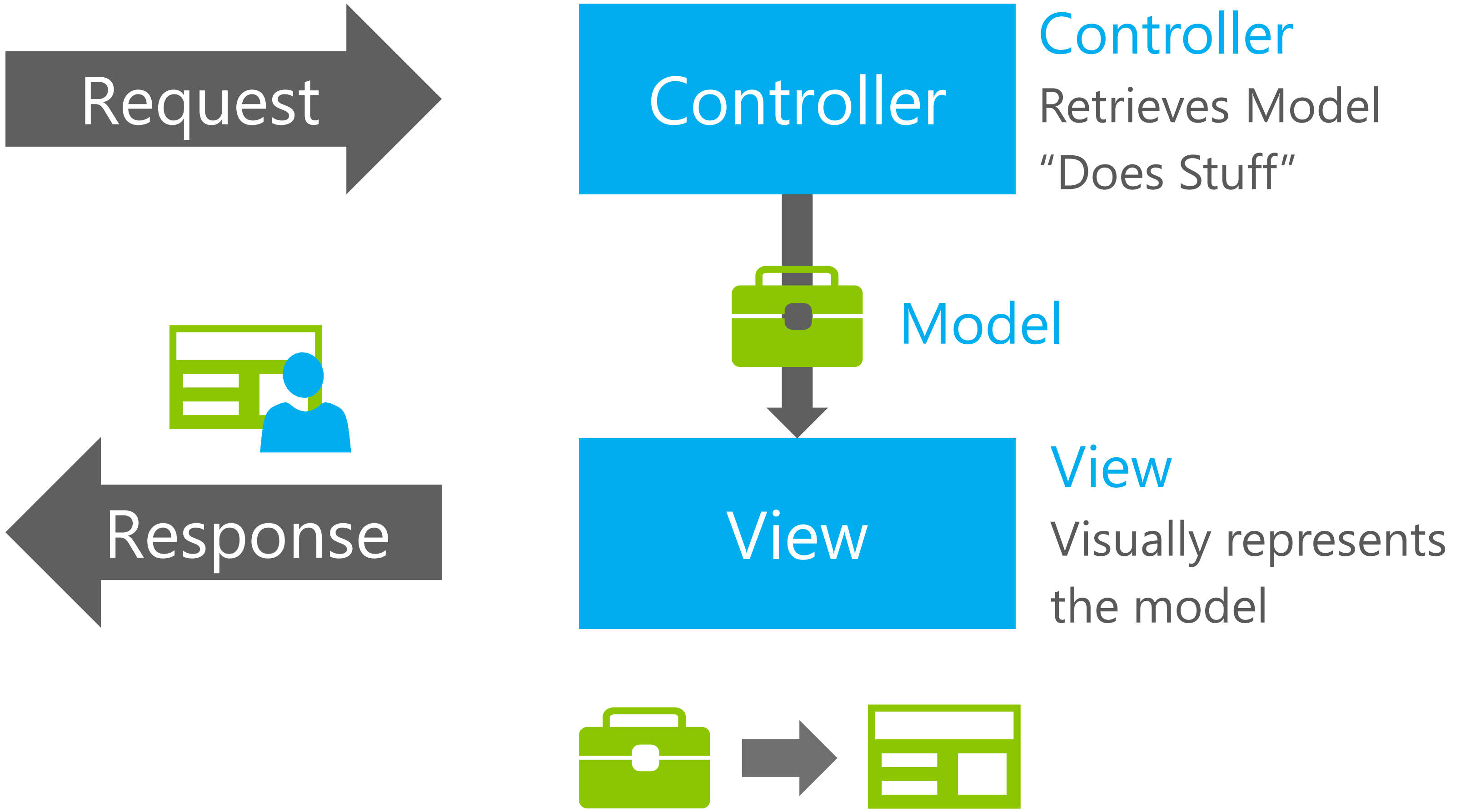
MVC or Web Forms

Asp.Net Web Forms	Asp.Net MVC
Asp.Net Web Form follow a traditional event driven development model.	Asp.Net MVC is a lightweight and follow MVC (Model, View, Controller) pattern based development model.
Asp.Net Web Form has server controls.	Asp.Net MVC has html helpers.
Asp.Net Web Form supports view state for state management at client side.	Asp.Net MVC does not support view state.
Asp.Net Web Form has file-based URLs means file name exist in the URLs must have its physically existence.	Asp.Net MVC has route-based URLs means URLs are divided into controllers and actions and moreover it is based on controller not on physical file.
Asp.Net Web Form follows Web Forms Syntax	Asp.Net MVC follow customizable syntax (Razor as default)
In Asp.Net Web Form, Web Forms(ASPX) i.e. views are tightly coupled to Code behind(ASPX.CS) i.e. logic	In Asp.Net MVC, Views and logic are kept separately
Asp.Net Web Form has Master Pages for consistent look and feels.	Asp.Net MVC has Layouts for consistent look and feels
Asp.Net Web Form has User Controls for code re-usability.	Asp.Net MVC has Partial Views for code re-usability.
Asp.Net Web Form has built-in data controls and best for rapid development with powerful data access.	Asp.Net MVC is lightweight, provide full control over markup and support many features that allow fast & agile development. Hence it is best for developing interactive web application with latest web standards
Asp.Net Web Form is not Open Source.	Asp.Net Web MVC is an Open Source

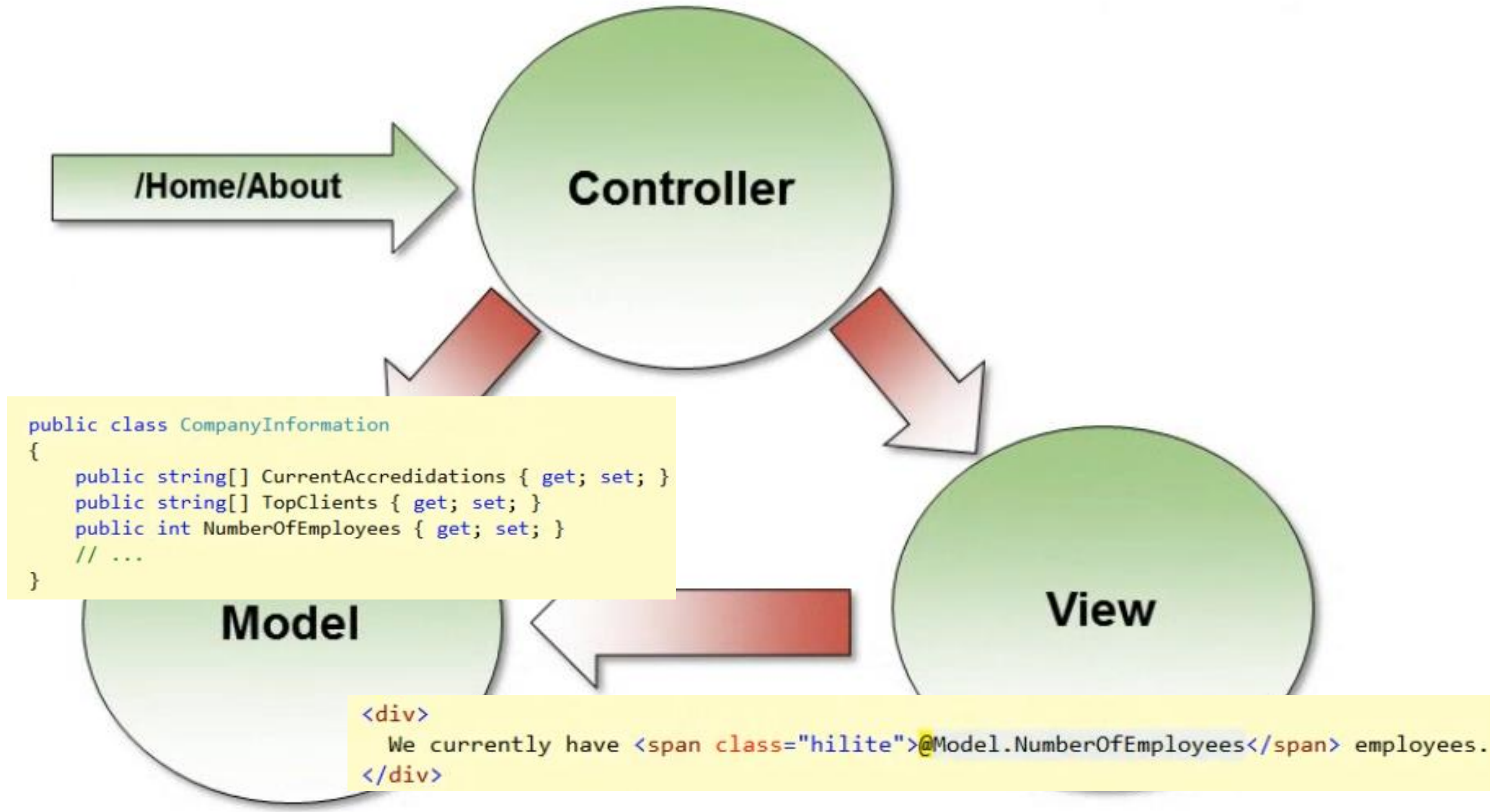
How ASP.NET MVC works



How ASP.NET MVC works



How ASP.NET MVC works



```
public class CompanyInformation
{
    public string[] CurrentAccredidations { get; set; }
    public string[] TopClients { get; set; }
    public int NumberOfEmployees { get; set; }
    // ...
}
```

```
<div>
  We currently have <span class="hilite">@Model.NumberOfEmployees</span> employees.
</div>
```

MVC : What is different than three Layered architecture ?

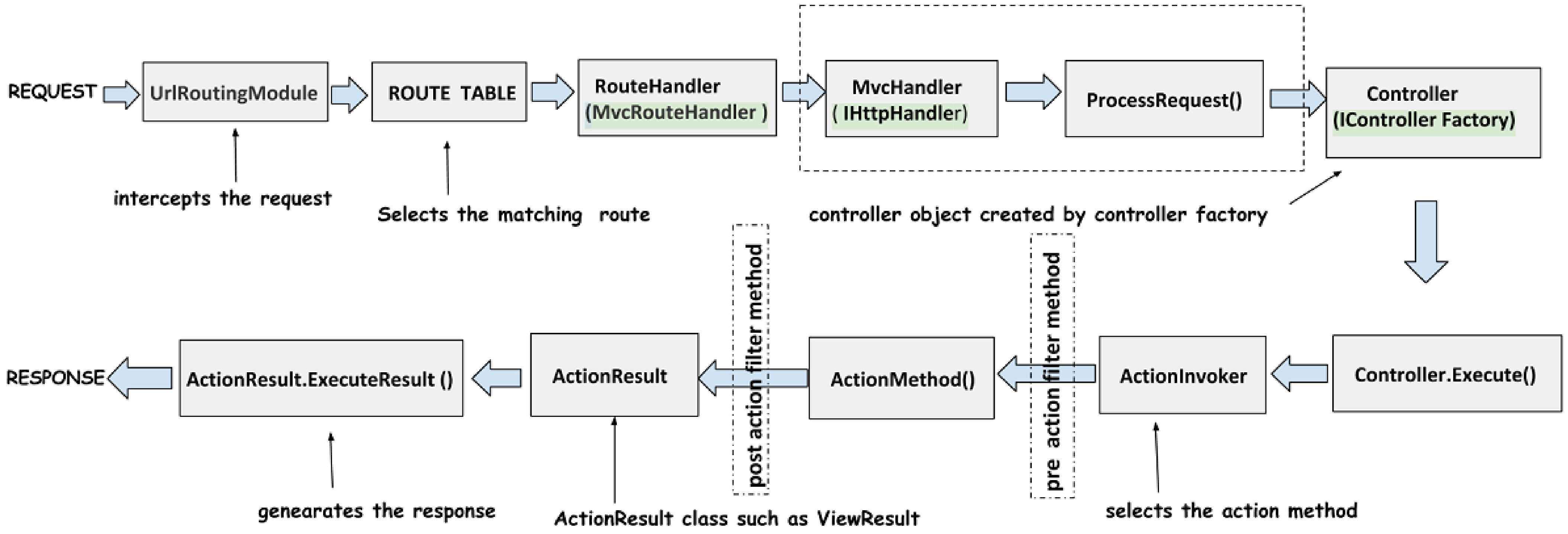
- MVC is an evolution of a three layered traditional architecture. Many components of the three layered architecture are part of MVC.

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

MVC : For Web or Window ?

- The MVC architecture is suited for a web application than Windows.
- For Window applications, MVP, i.e., “Model View Presenter” is more applicable.
- If you are using WPF and Silverlight, MVVM is more suitable due to bindings.

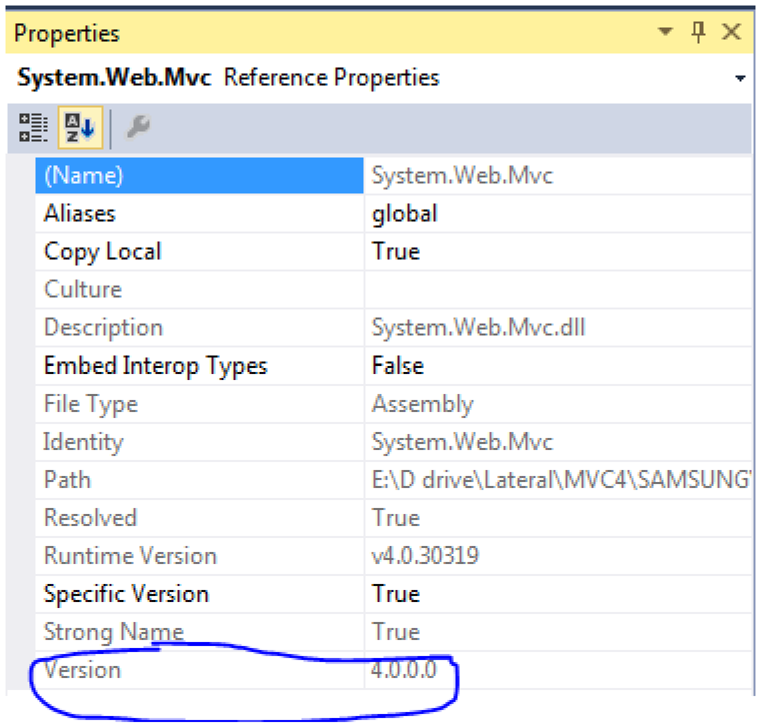
MVC Page Request Processing


REQUEST
UrlRoutingModule
ROUTE TABLE
RouteHandler (MvcRouteHandler)
MvcHandler (IHttpHandler)
ProcessRequest()
Controller (IController Factory)
intercepts the request
Selects the matching route
controller object created by controller factory
Controller.Execute()
ActionInvoker
pre action filter method
ActionMethod()
post action filter method
ActionResult
ActionResult.ExecuteResult ()
RESPONSE
selects the action method
ActionResult class such as ViewResult
genearates the response

MVC – Get Practical

Checking Asp.net Running Version

There are two ways to check version of existing MVC Application



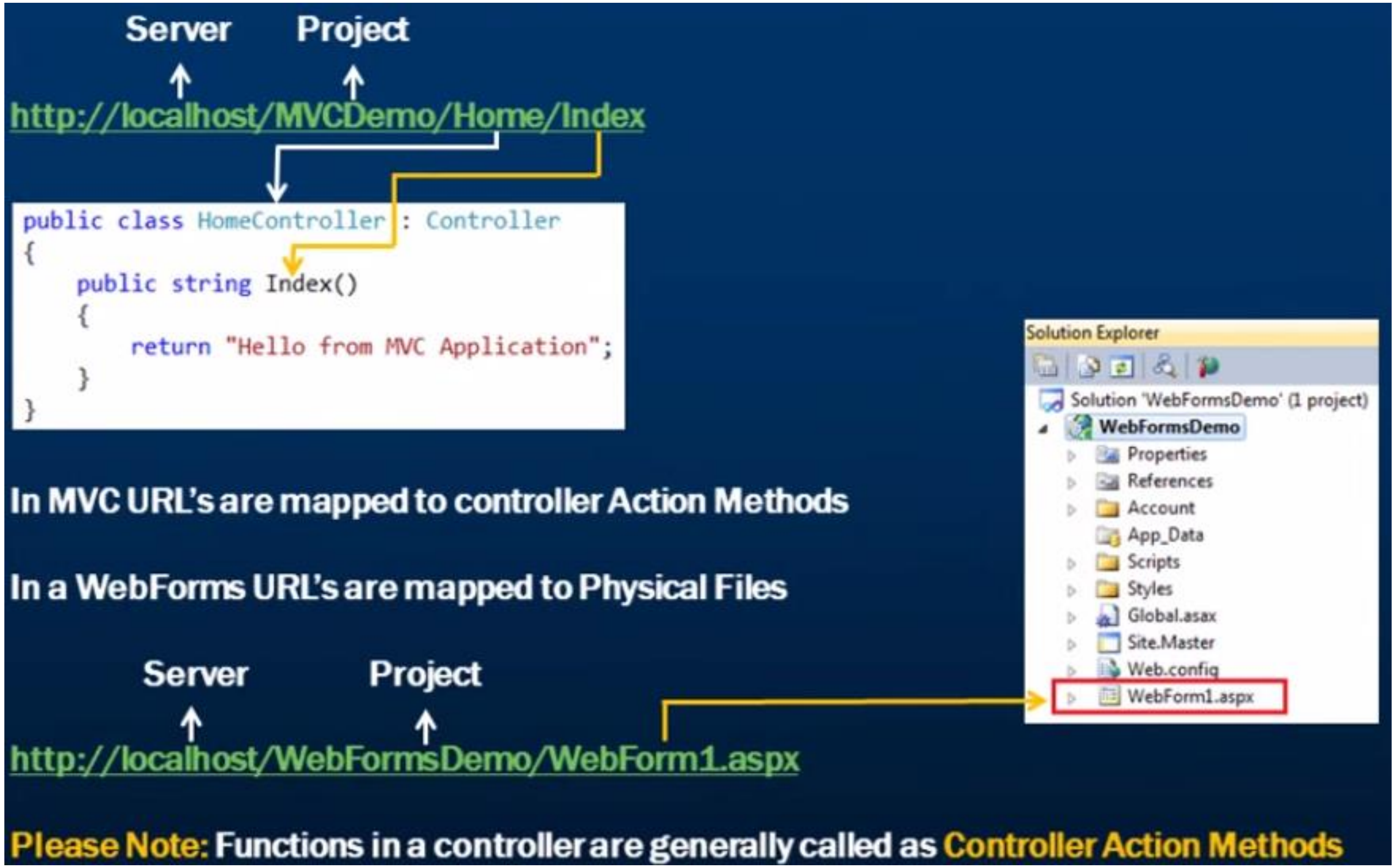
Properties

System.Web.Mvc Reference Properties

(Name)	System.Web.Mvc
Aliases	global
Copy Local	True
Culture	
Description	System.Web.Mvc.dll
Embed Interop Types	False
File Type	Assembly
Identity	System.Web.Mvc
Path	E:\D drive\Lateral\MVC4\SAMSUNG'
Resolved	True
Runtime Version	v4.0.30319
Specific Version	True
Strong Name	True
Version	4.0.0.0

```
public class HomeController : Controller
{
    public string  Index()
    {
        string val = typeof(Controller).Assembly.GetName().Version.ToString();
        return "Current Used Vesion of MVC Is : " + val;
    }
}
```

Creating First MVC Application



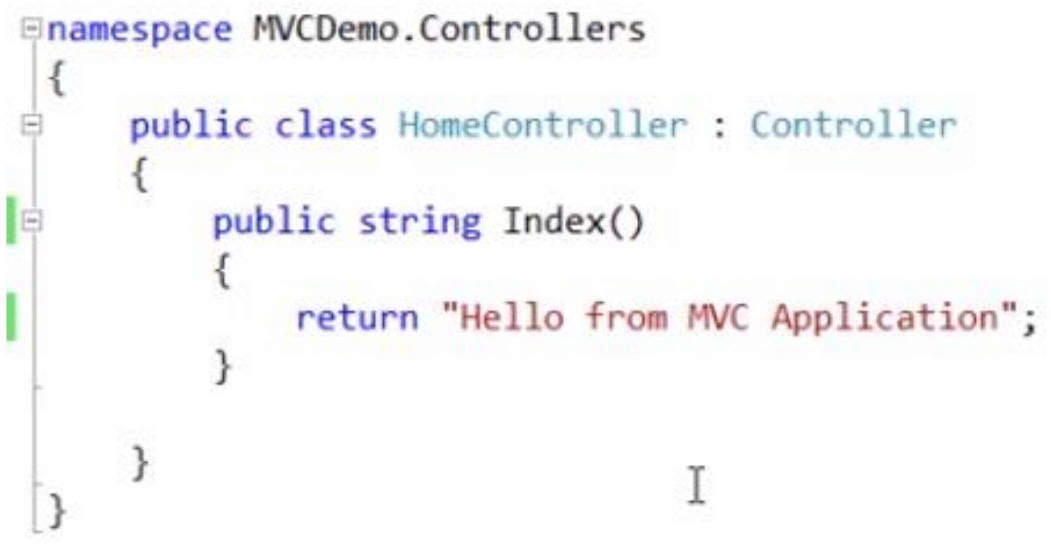
```
public class HomeController : Controller
{
    public string Index()
    {
        return "Hello from MVC Application";
    }
}
```

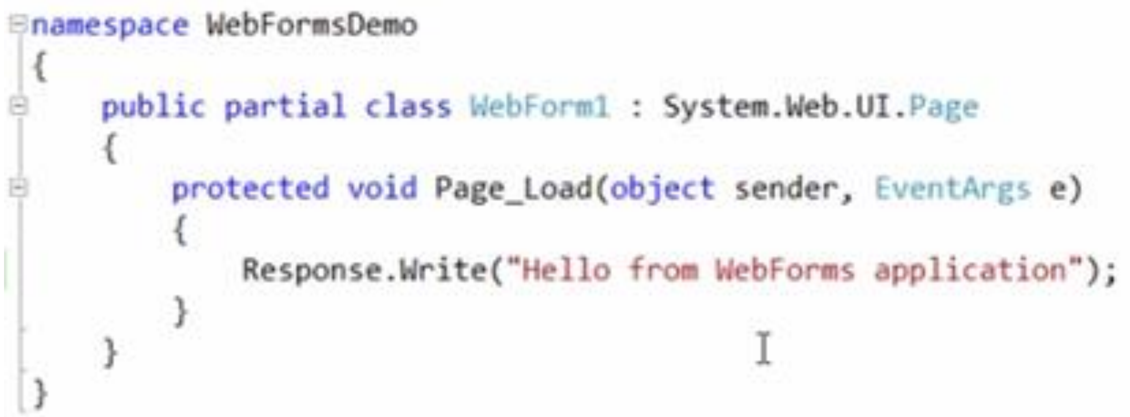
In MVC URL's are mapped to controller Action Methods

In a WebForms URL's are mapped to Physical Files

Please Note: Functions in a controller are generally called as Controller Action Methods

Creating First MVC Application


```
namespace MVCDemo.Controllers
{
    public class HomeController : Controller
    {
        public string Index()
        {
            return "Hello from MVC Application";
        }

    }
}
```

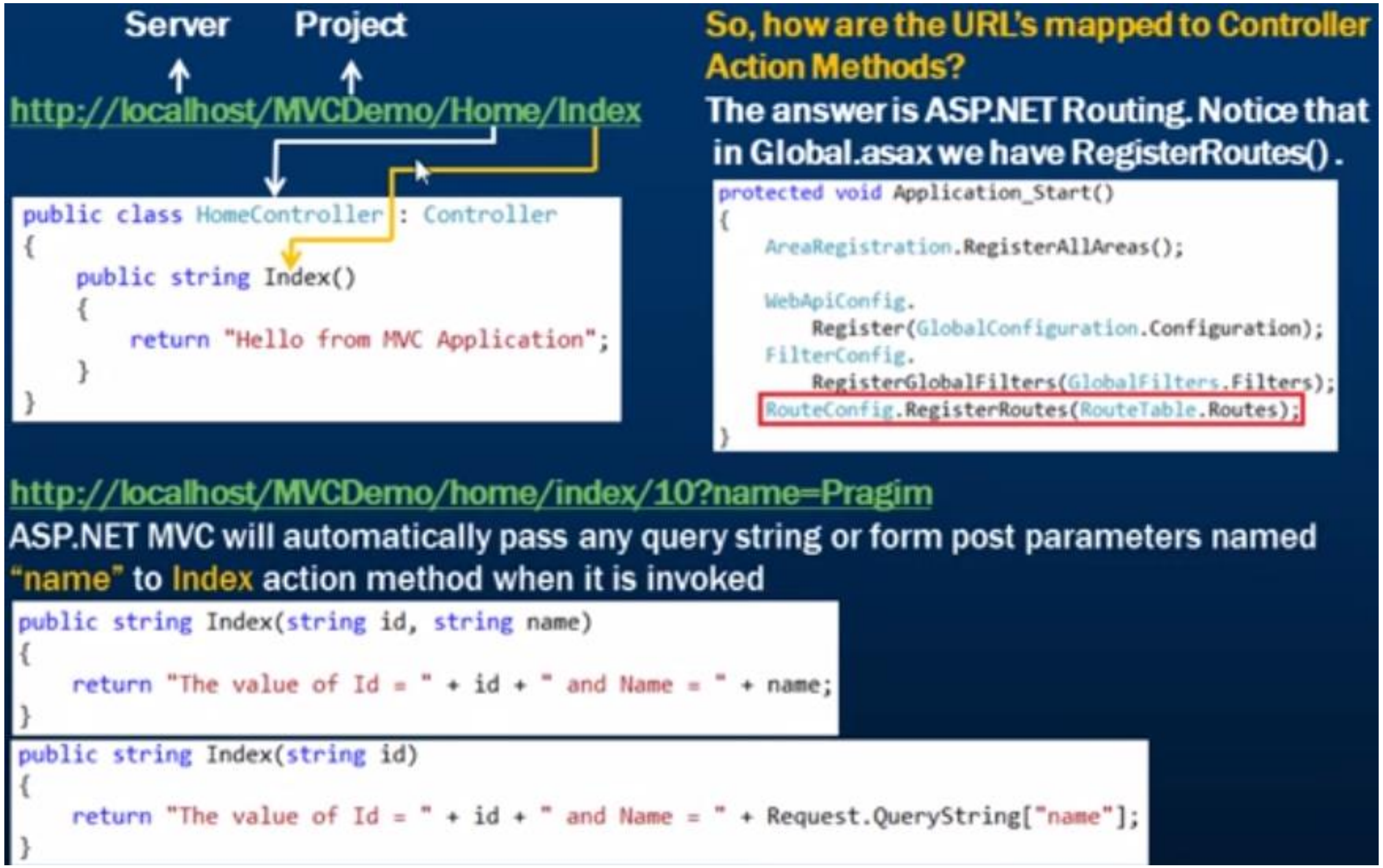

```
namespace WebFormsDemo
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write("Hello from WebForms application");
        }
    }
}
```

Controller & Action

Controller

- The ASP.NET MVC framework maps URLs to classes that are referred to as controllers.
- Controllers process incoming requests, handle user input and interactions, and execute appropriate application logic.
- A controller class typically calls a separate view component to generate the HTML markup for the request.
- The base class for all controllers is the ControllerBase class, which provides general MVC handling.
- The Controller class inherits from ControllerBase and is the default implement of a controller.
- The Controller class is responsible for the following processing stages:
  - Locating the appropriate action method to call and validating that it can be called.
  - Getting the values to use as the action method's arguments.
  - Handling all errors that might occur during the execution of the action method.
  - Providing the default WebFormViewEngine class for rendering ASP.NET page types (views)

Controller Contd.



http://localhost/MVCDemo/Home/Index

```
public class HomeController : Controller
{
    public string Index()
    {
        return "Hello from MVC Application";
    }
}
```

So, how are the URL's mapped to Controller Action Methods?

The answer is ASP.NET Routing. Notice that in Global.asax we have RegisterRoutes() .

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();

    WebApiConfig.
        Register(GlobalConfiguration.Configuration);
    FilterConfig.
        RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
}
```

http://localhost/MVCDemo/home/index/10?name=Pragim

ASP.NET MVC will automatically pass any query string or form post parameters named "name" to Index action method when it is invoked

```
public string Index(string id, string name)
{
    return "The value of Id = " + id + " and Name = " + name;
}
```

```
public string Index(string id)
{
    return "The value of Id = " + id + " and Name = " + Request.QueryString["name"];
}
```

Adding Actions

- Controllers are classes
- Actions are methods
- Creating an action involves adding a method to a class

Action Signature

- Return Types
  - ActionResult
    - FileResult
    - JsonResult
    - ViewResult
- Parameters
  - Normal parameters
  - MVC model binding

Action Results

- Actions typically return an ActionResult

Name	Framework Behavior	Producing Method
ContentResult	Returns a string literal	Content
EmptyResult	No response	
FileContentResult / FilePathResult /FileStreamResult	Returns the contents of a file	File
HttpUnauthorizedResult	Returns and HTTP 403 status	
JavaScriptResult	Returns a script to execute	JavaScript
JsonResult	Returns data in JSON format	Json
RedirectResult	Redirect the client to a new URL	Redirect
RedirectToRouteResult	Redirect to another action , or another controller's action	RedirectToRoute / RedirectToAction
ViewResult PartialViewResult	Response is the responsibility of a view engine	View/PrtialView

Action Method Parameters

- By default, the values for action method parameters are retrieved from the request's data collection.

- The data collection includes name/values pairs for form data, query string values.

- The controller class locates the action method and determines any parameter values for the action method.

- If the parameter value cannot be parsed, and if the type of the parameter is a reference type or a nullable value type, **null** is passed as the parameter value. Otherwise, an exception is thrown.

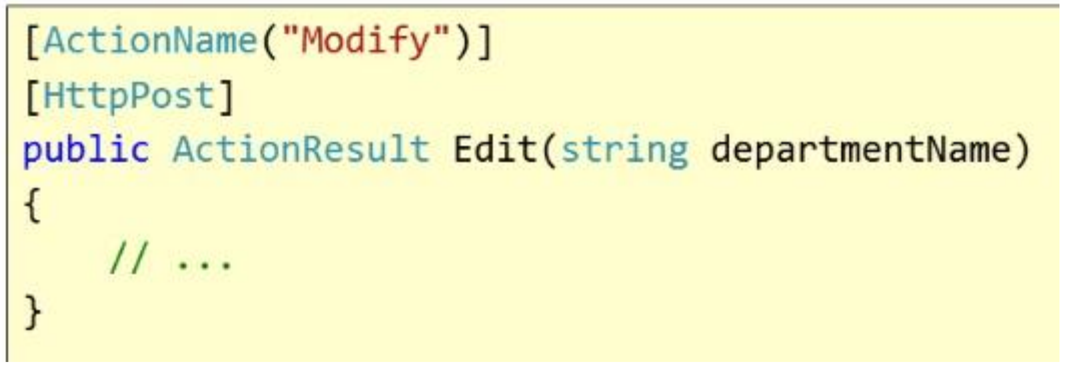
- There are several ways to access URL parameter values in the action methods of controller classes.

Get and Post

- Create/Update/Delete are typically two step operations
  - Present the form
  - Accept the input
- Create two actions
  - Form presentation via HttpGet (default)
  - Accept data via HttpPost

Action Selectors

- ActionName
- Accept Verbs
  - HttpPost
  - HttpGet

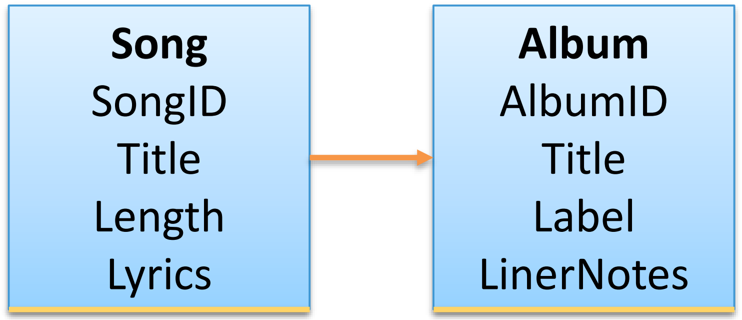


```
[ActionName("Modify")]
[HttpPost]
public ActionResult Edit(string departmentName)
{
    // ...
}
```

Default Model Binder

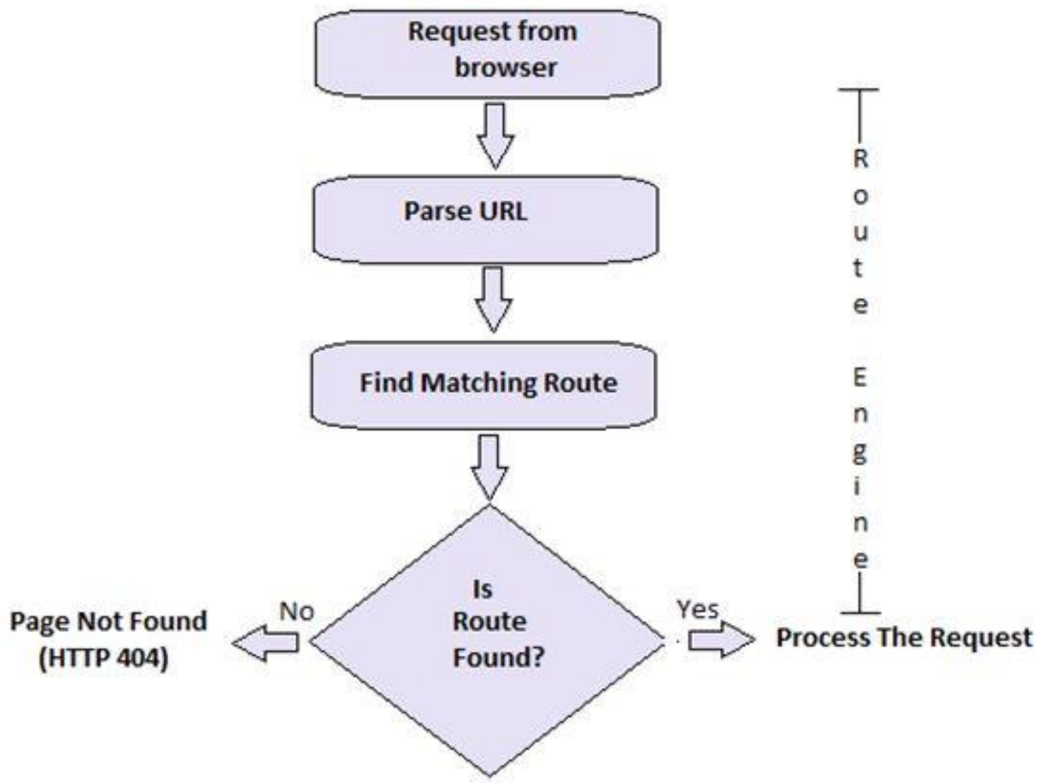
- "It just works"
- Uses the name attribute of input elements
  - Automatically matches parameter names for simple data types
  - Complex objects are mapped by property name
    - Complex properties use dotted notation

      - `<input type="text" name="Album.LinerNotes" />`

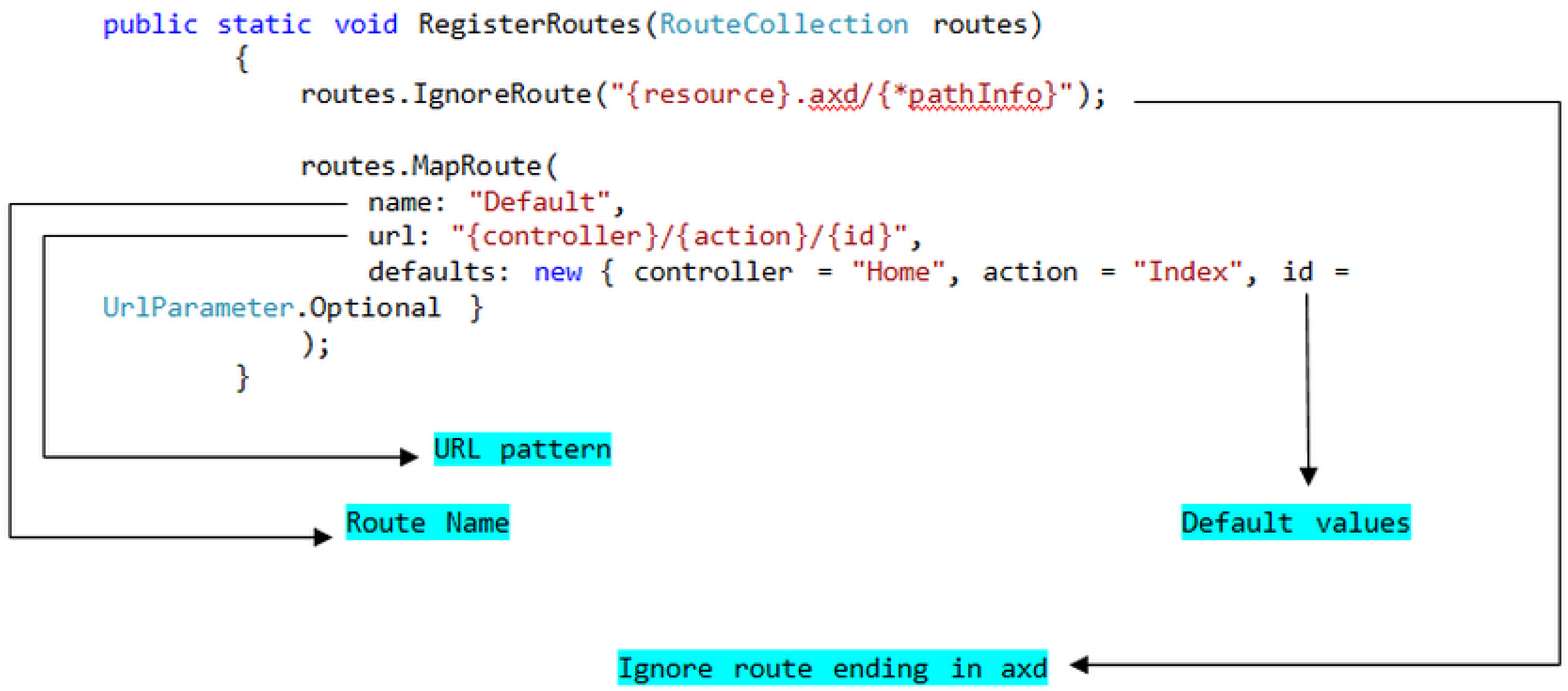


Routing

- A route is a URL pattern.
- Routing is a pattern matching process that monitors the requests and determines what to do with each request.
- Routing is a mechanism for mapping requests within our MVC application.
- The Routing mechanism passes the request to the handler.
- A handler may be a physical path such as .aspx or may be a class capable of processing our request such as controller in MVC.



Routing Contd.



```
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 }
            );
        }
```

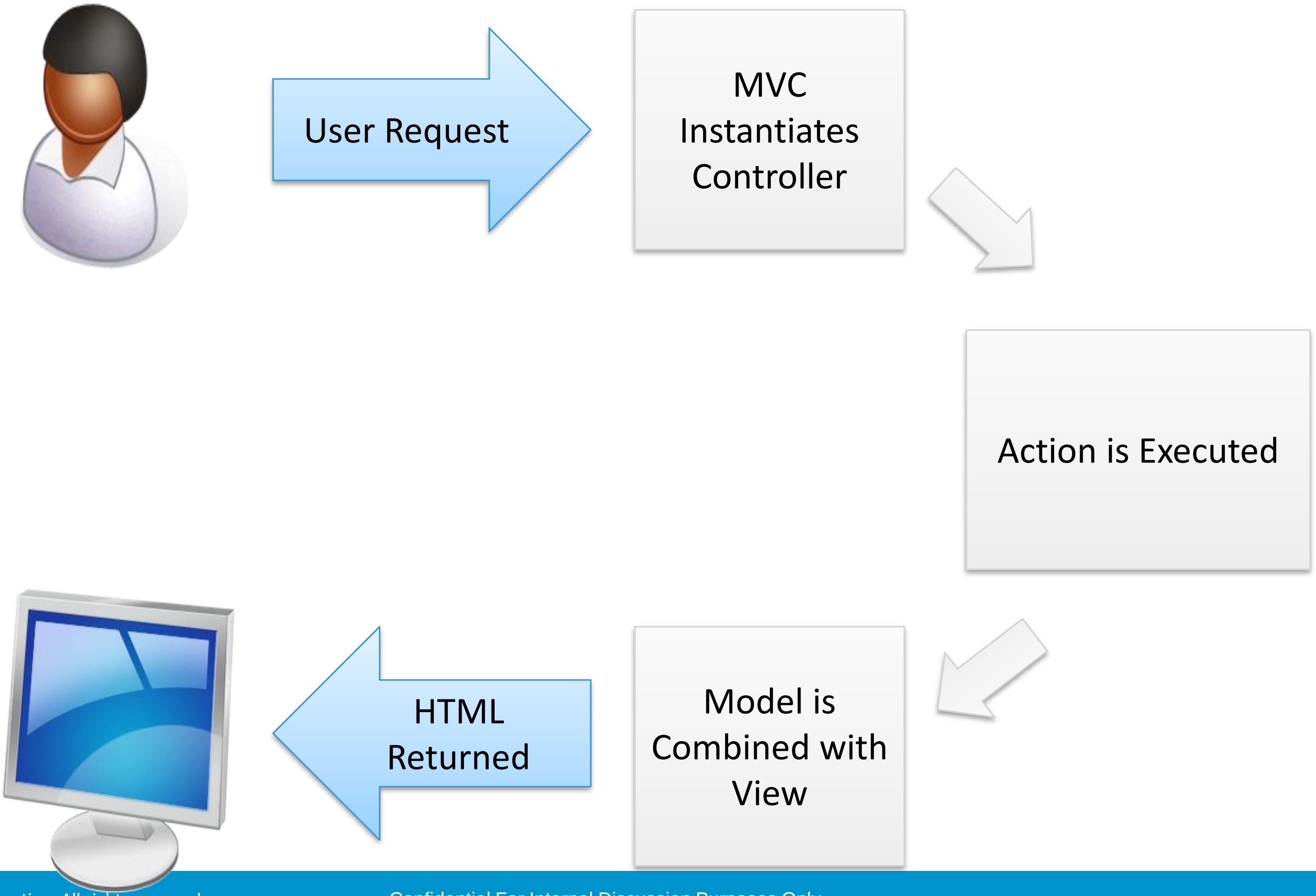
Action Filters

- Sometimes you want to perform logic either before an action method is called or after an action method runs. To support this, ASP.NET MVC provides action filters.

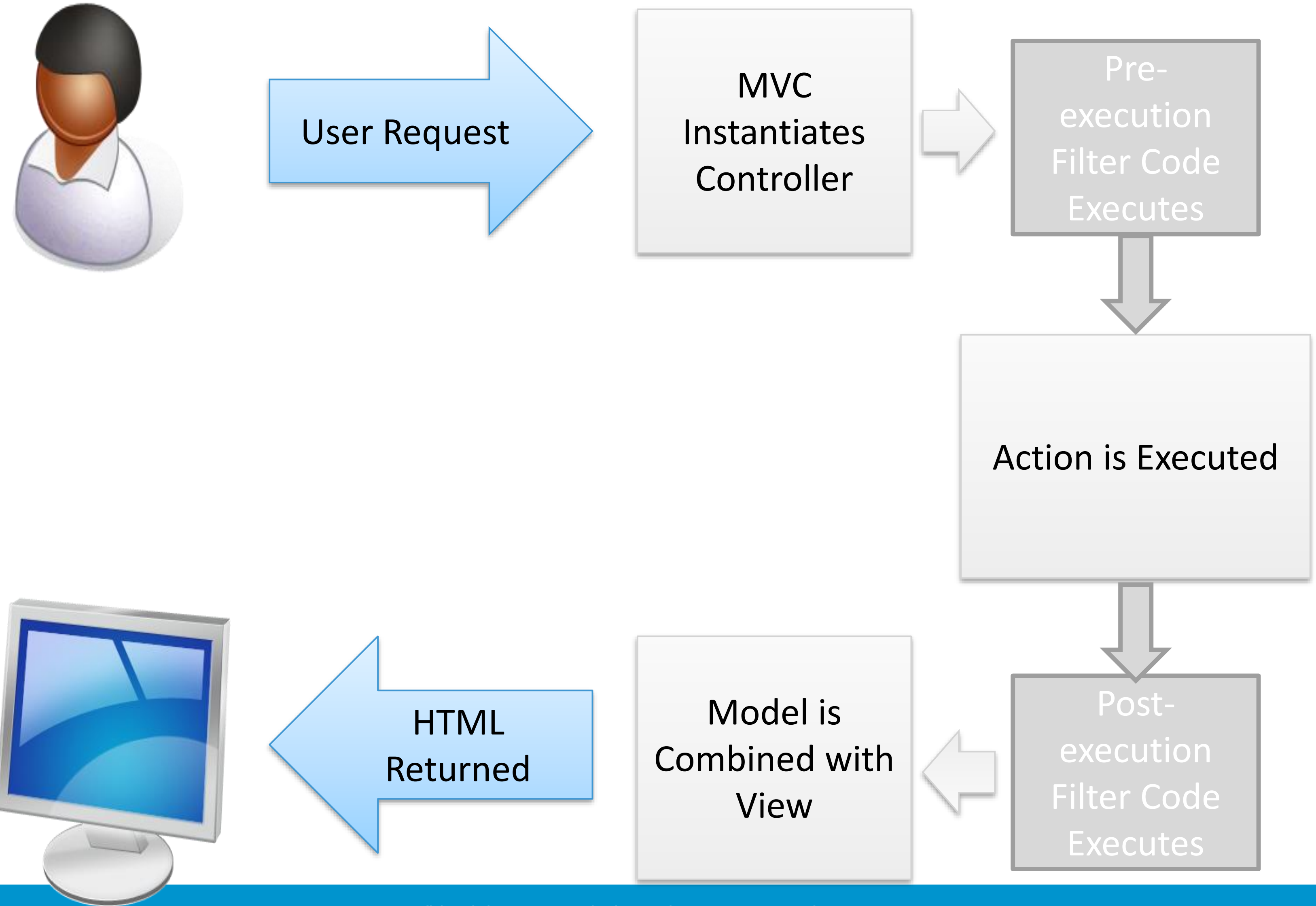
- Action filters are custom attributes that provide a declarative means to add pre-action and post-action behavior to controller action methods.

- Few Commonly used action filters
  - Non Action
  - Authorize & Allow Anonymous
  - Child Action Only
  - Output Cache
  - Require Https
  - Validate Inputs

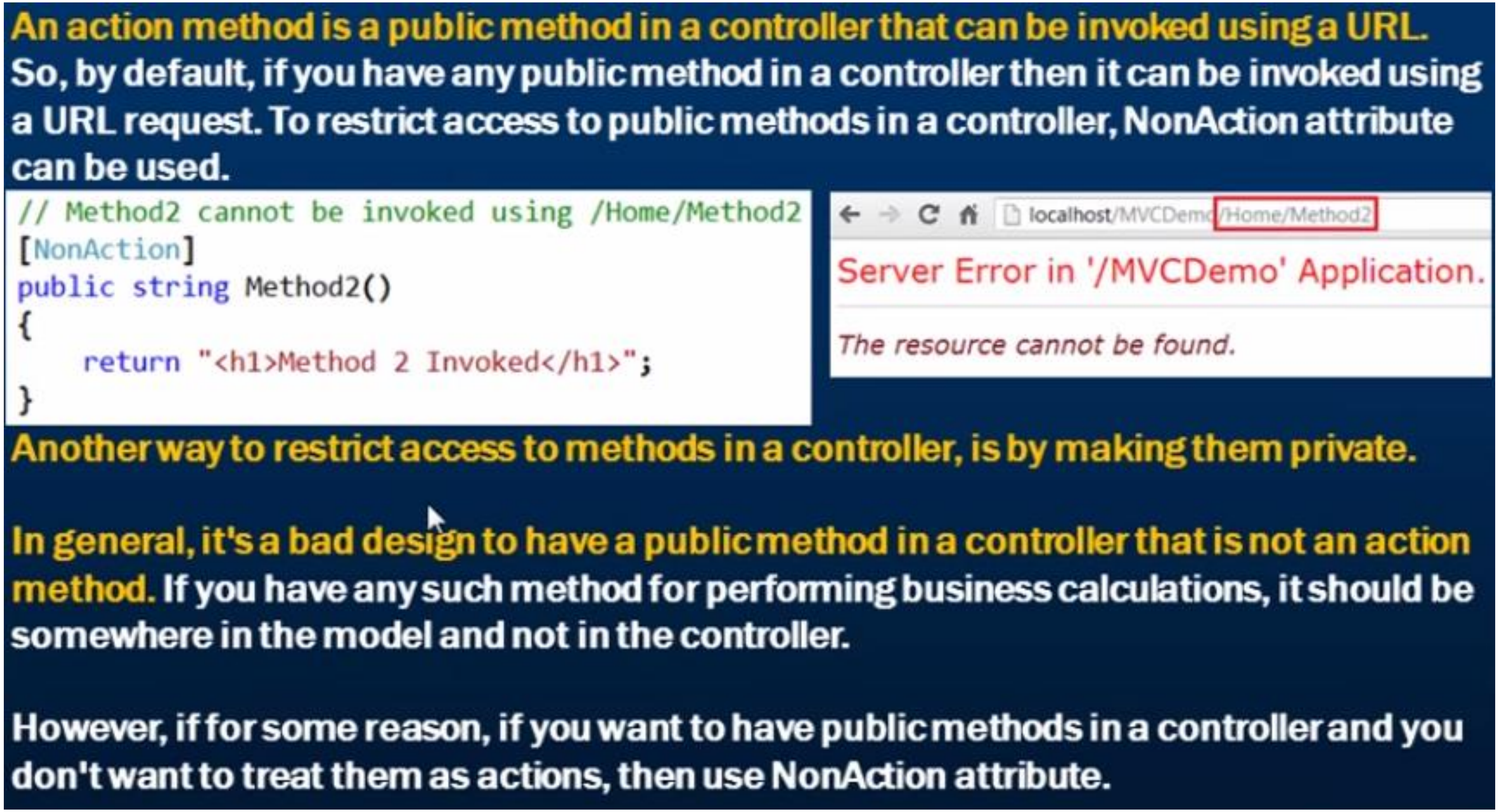
Normal Action Execution



Actions with Filters


User Request
MVC Instantiates Controller
Pre-execution Filter Code Executes
Action is Executed
Post-execution Filter Code Executes
Model is Combined with View
HTML Returned

Action Filters : Non Action



An action method is a public method in a controller that can be invoked using a URL. So, by default, if you have any public method in a controller then it can be invoked using a URL request. To restrict access to public methods in a controller, NonAction attribute can be used.

```
// Method2 cannot be invoked using /Home/Method2
[NonAction]
public string Method2()
{
    return "<h1>Method 2 Invoked</h1>";
}
```

localhost/MVCDemo/Home/Method2

Server Error in '/MVCDemo' Application.

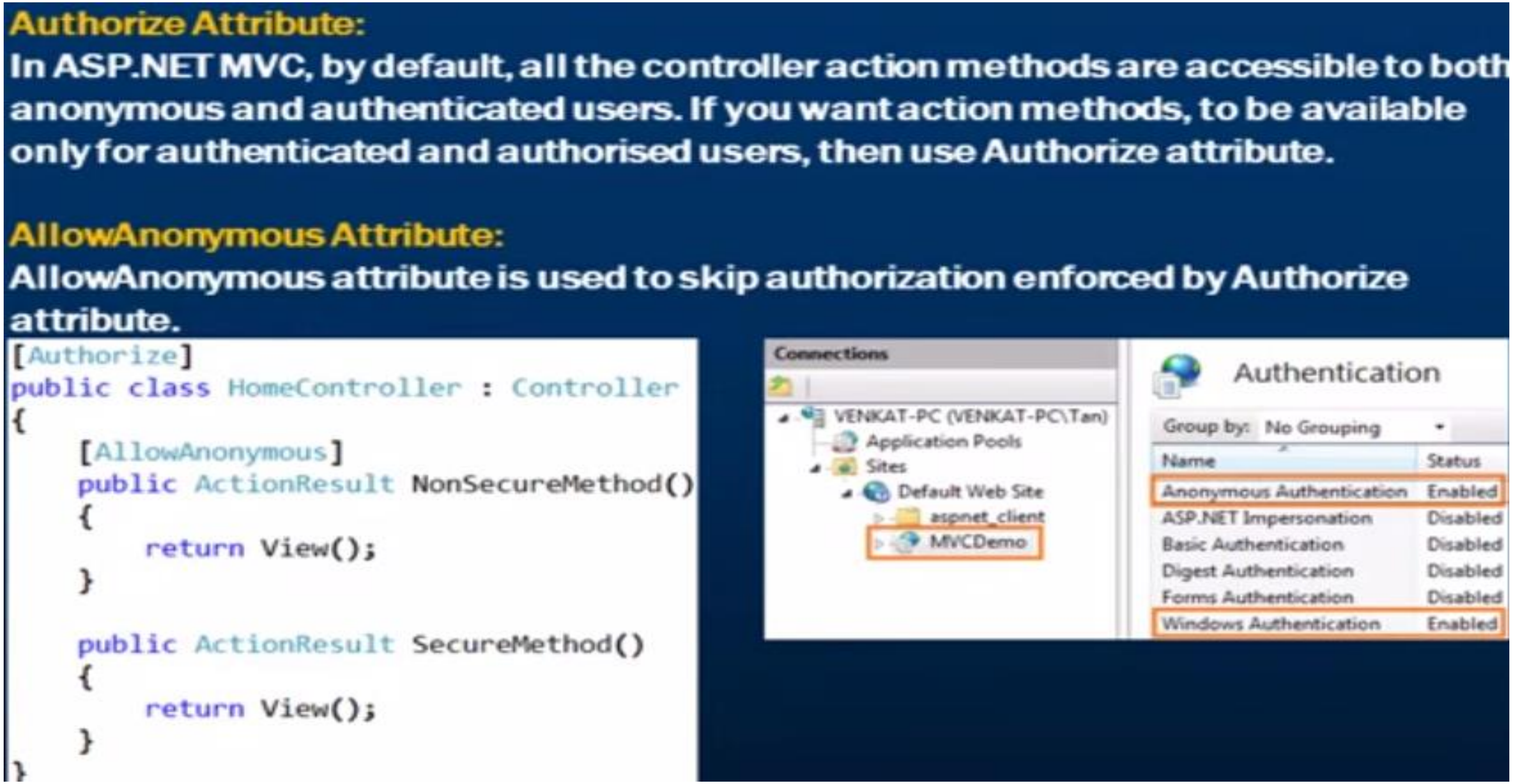
The resource cannot be found.

Another way to restrict access to methods in a controller, is by making them private.

In general, it's a bad design to have a public method in a controller that is not an action method. If you have any such method for performing business calculations, it should be somewhere in the model and not in the controller.

However, if for some reason, if you want to have public methods in a controller and you don't want to treat them as actions, then use NonAction attribute.

Action Filters : Authorize & Allow Anonymous



Authorize Attribute:

In ASP.NET MVC, by default, all the controller action methods are accessible to both anonymous and authenticated users. If you want action methods, to be available only for authenticated and authorised users, then use Authorize attribute.

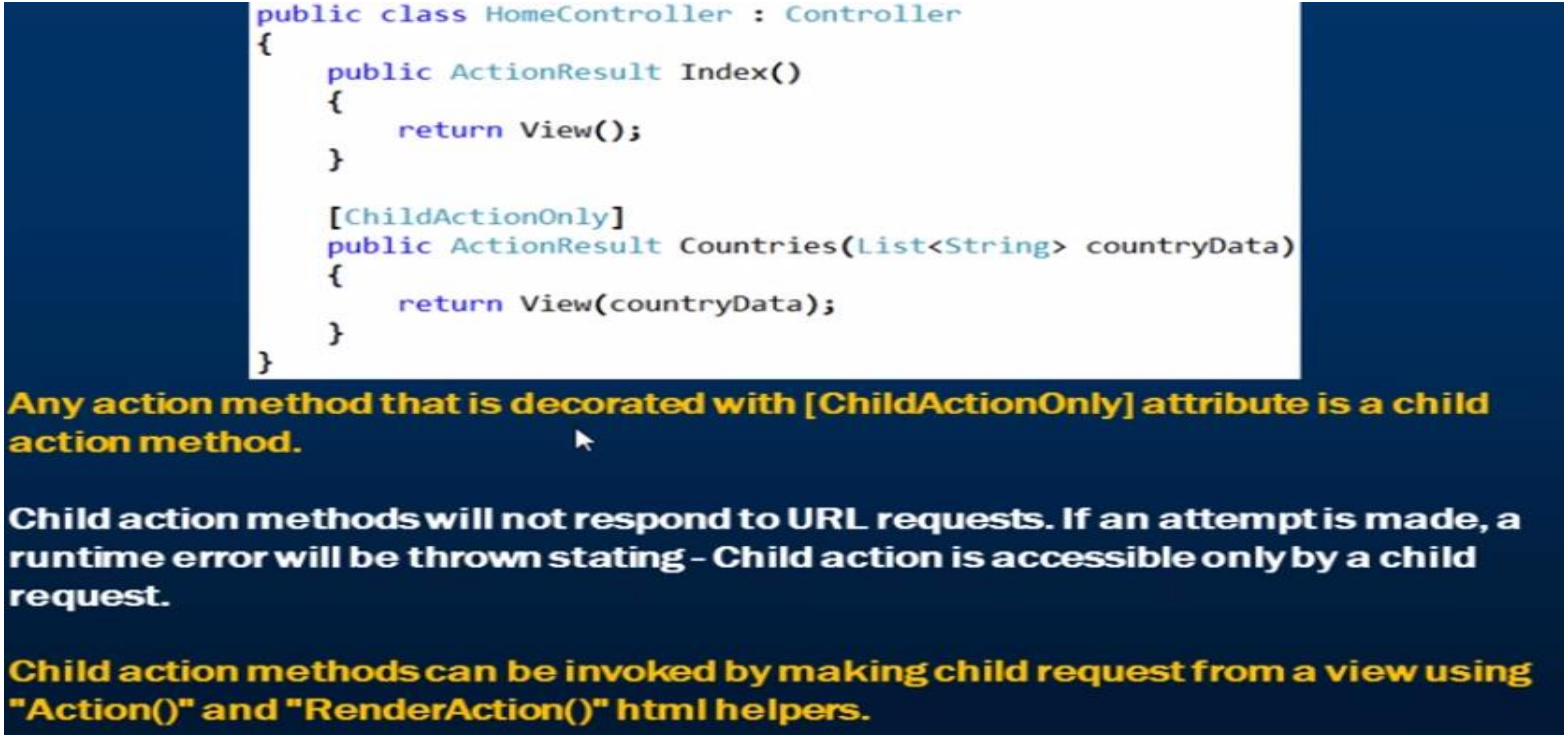
AllowAnonymous Attribute:

AllowAnonymous attribute is used to skip authorization enforced by Authorize attribute.

```
[Authorize]
public class HomeController : Controller
{
    [AllowAnonymous]
    public ActionResult NonSecureMethod()
    {
        return View();
    }

    public ActionResult SecureMethod()
    {
        return View();
    }
}
```

Name	Status
Anonymous Authentication	Enabled
ASP.NET Impersonation	Disabled
Basic Authentication	Disabled
Digest Authentication	Disabled
Forms Authentication	Disabled
Windows Authentication	Enabled

Action Filters : Child Action Only



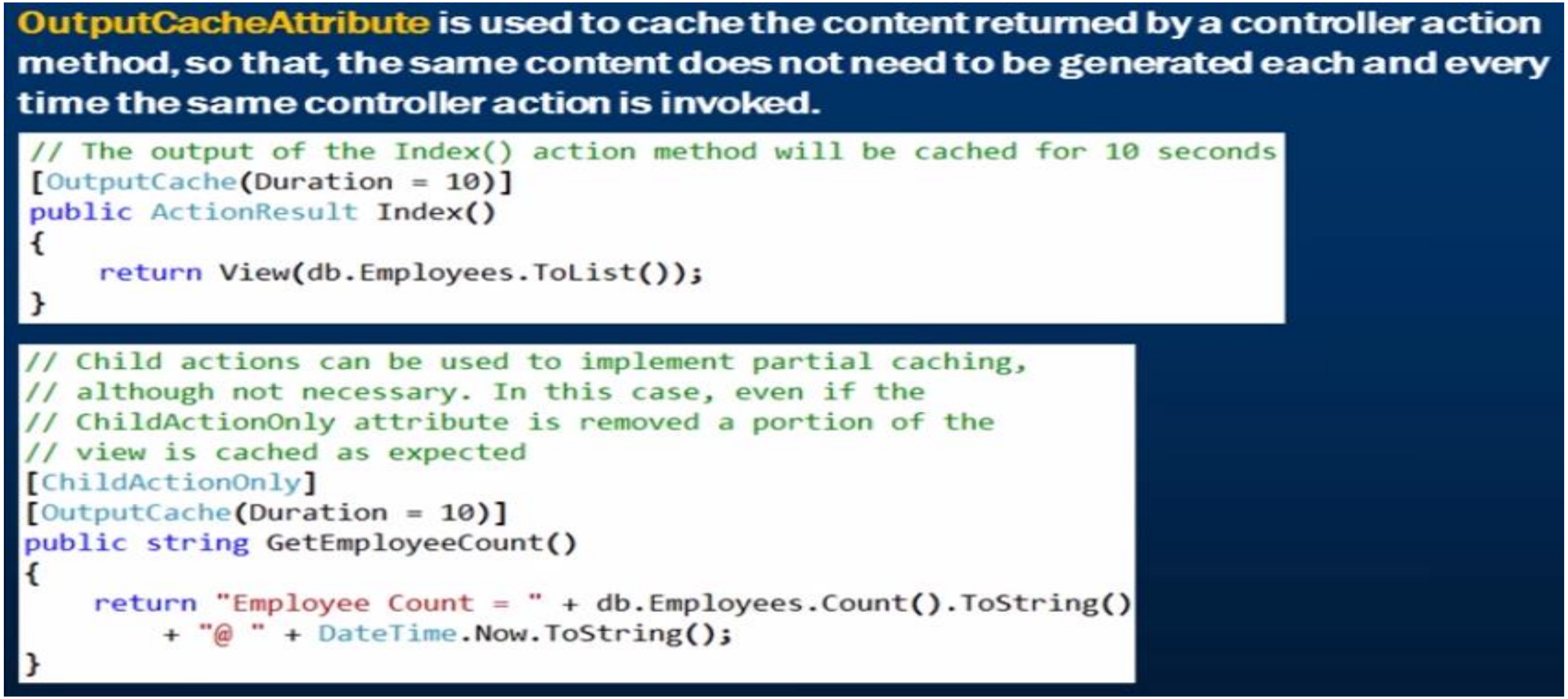
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    [ChildActionOnly]
    public ActionResult Countries(List<String> countryData)
    {
        return View(countryData);
    }
}
```

Any action method that is decorated with [ChildActionOnly] attribute is a child action method.

Child action methods will not respond to URL requests. If an attempt is made, a runtime error will be thrown stating - Child action is accessible only by a child request.

Child action methods can be invoked by making child request from a view using "Action()" and "RenderAction()" html helpers.

Action Filters : Output Cache



OutputCacheAttribute is used to cache the content returned by a controller action method, so that, the same content does not need to be generated each and every time the same controller action is invoked.

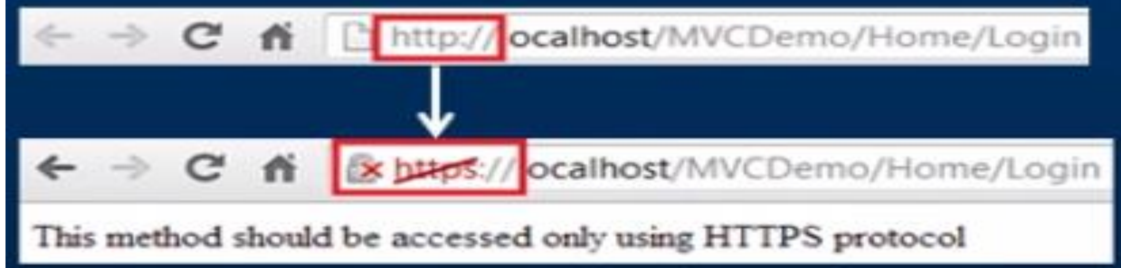
```
// The output of the Index() action method will be cached for 10 seconds
[OutputCache(Duration = 10)]
public ActionResult Index()
{
    return View(db.Employees.ToList());
}
```

```
// Child actions can be used to implement partial caching,
// although not necessary. In this case, even if the
// ChildActionOnly attribute is removed a portion of the
// view is cached as expected
[ChildActionOnly]
[OutputCache(Duration = 10)]
public string GetEmployeeCount()
{
    return "Employee Count = " + db.Employees.Count().ToString()
        + "@ " + DateTime.Now.ToString();
}
```

Action Filters : Require Https

[RequireHttps] attribute forces an unsecured HTTP request to be re-sent over HTTPS.

```
[RequireHttps]
public string Login()
{
    return "This method should be accessed only using HTTPS protocol";
}
```



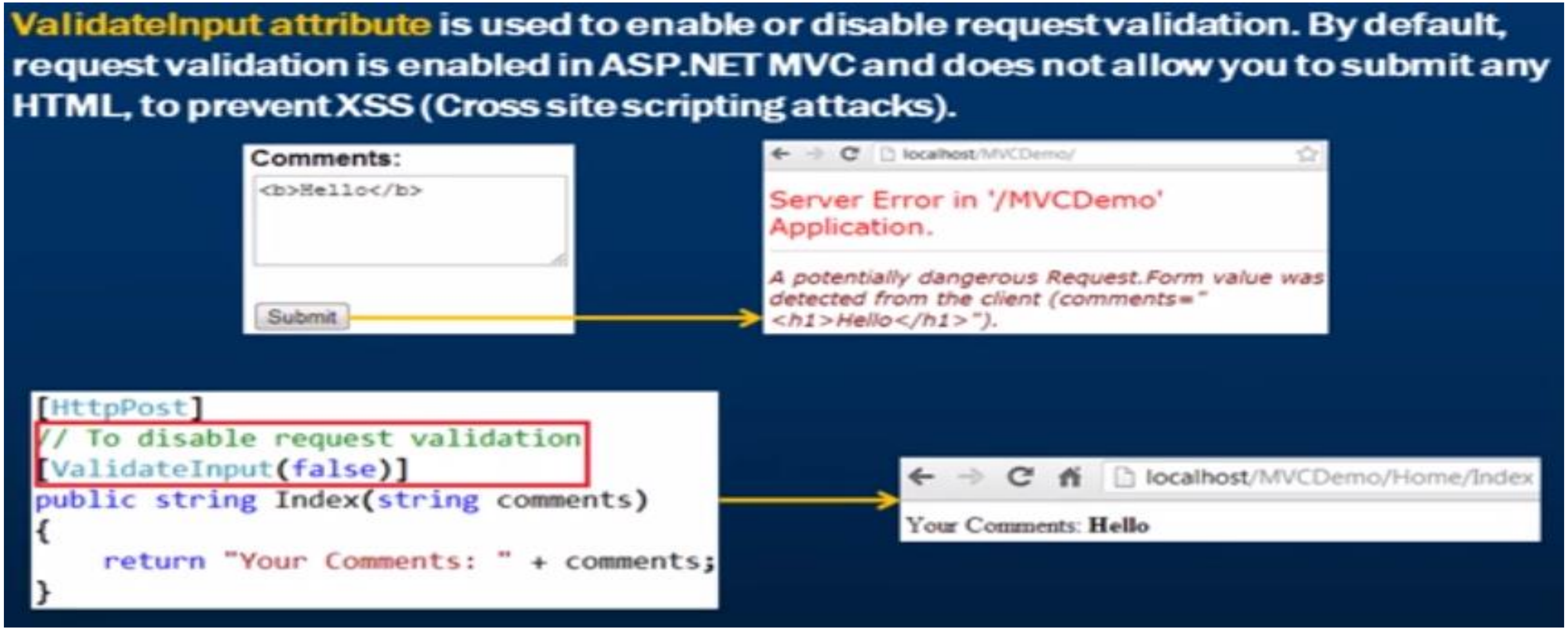
http://localhost/MVCDemo/Home/Login

~~https~~://localhost/MVCDemo/Home/Login

This method should be accessed only using HTTPS protocol

RequireHttps attribute can be applied on a controller as well. In this case, it is applicable for all action methods with in that controller. Sensitive data such as login credentials, credit card information etc, must always be transmitted using HTTPS. Information transmitted over https is encrypted.

Action Filters : Validate Input



ValidateInput attribute is used to enable or disable request validation. By default, request validation is enabled in ASP.NET MVC and does not allow you to submit any HTML, to prevent XSS (Cross site scripting attacks).

Comments:

```
<b>Hello</b>
```

Submit

localhost/MVCDemo/

Server Error in '/MVCDemo' Application.

A potentially dangerous Request.Form value was detected from the client (comments="<h1>Hello</h1>").

```
[HttpPost]
// To disable request validation
[ValidateInput(false)]
public string Index(string comments)
{
    return "Your Comments: " + comments;
}
```

localhost/MVCDemo/Home/Index

Your Comments: **Hello**

infogain

Day2

Views & Model

Views

- ASP.NET MVC does not include anything that directly corresponds to a page.
- In an ASP.NET MVC application, there is not a page on disk that corresponds to the path in the URL that you type into the address bar of your browser.
- The closest thing to a page in an ASP.NET MVC application is something called a *view*.
- In ASP.NET MVC application, incoming browser requests are mapped to controller actions. A controller action might return a view.
- A controller action might perform some other type of action such as redirecting you to another controller action.

Razor View

- Razor is an ASP.NET programming syntax used to create dynamic web pages with the C# or Visual Basic .NET programming languages.
- Razor was released for Microsoft Visual Studio 2010 in January 2011.
- Razor is a simple-syntax view engine and was released as part of ASP.NET MVC 3 and the Microsoft WebMatrix tool set.
- The idea behind Razor is to provide an optimized syntax for HTML generation using a code-focused templating approach, with minimal transition between HTML and code.
- The design reduces the number of characters and keystrokes, and enables a more fluid coding workflow by not requiring explicitly denoted server blocks within the HTML code

Razor View Vs Web Form View

Razor View Engine	Web Form View Engine
Razor Engine is an advanced view engine that was introduced with MVC3. This is not a new language but it is a new markup syntax.	Web Form Engine is the default view engine for the Asp.net MVC that is included with Asp.net MVC from the beginning.
The namespace for Razor Engine is System.Web.Razor.	The namespace for Webform Engine is System.Web.Mvc.WebFormViewEngine.
The file extensions used with Razor Engine are different from Web Form Engine. It has .cshtml (Razor with C#) or .vbhtml (Razor with VB).	The file extensions used with Web Form Engine are also like Asp.net Web Forms. It has .aspx extension for views, .ascx.
Razor has new and advance syntax that are compact, expressive and reduces typing.	Web Form Engine has the same syntax like Asp.net Web Forms uses for .aspx pages.

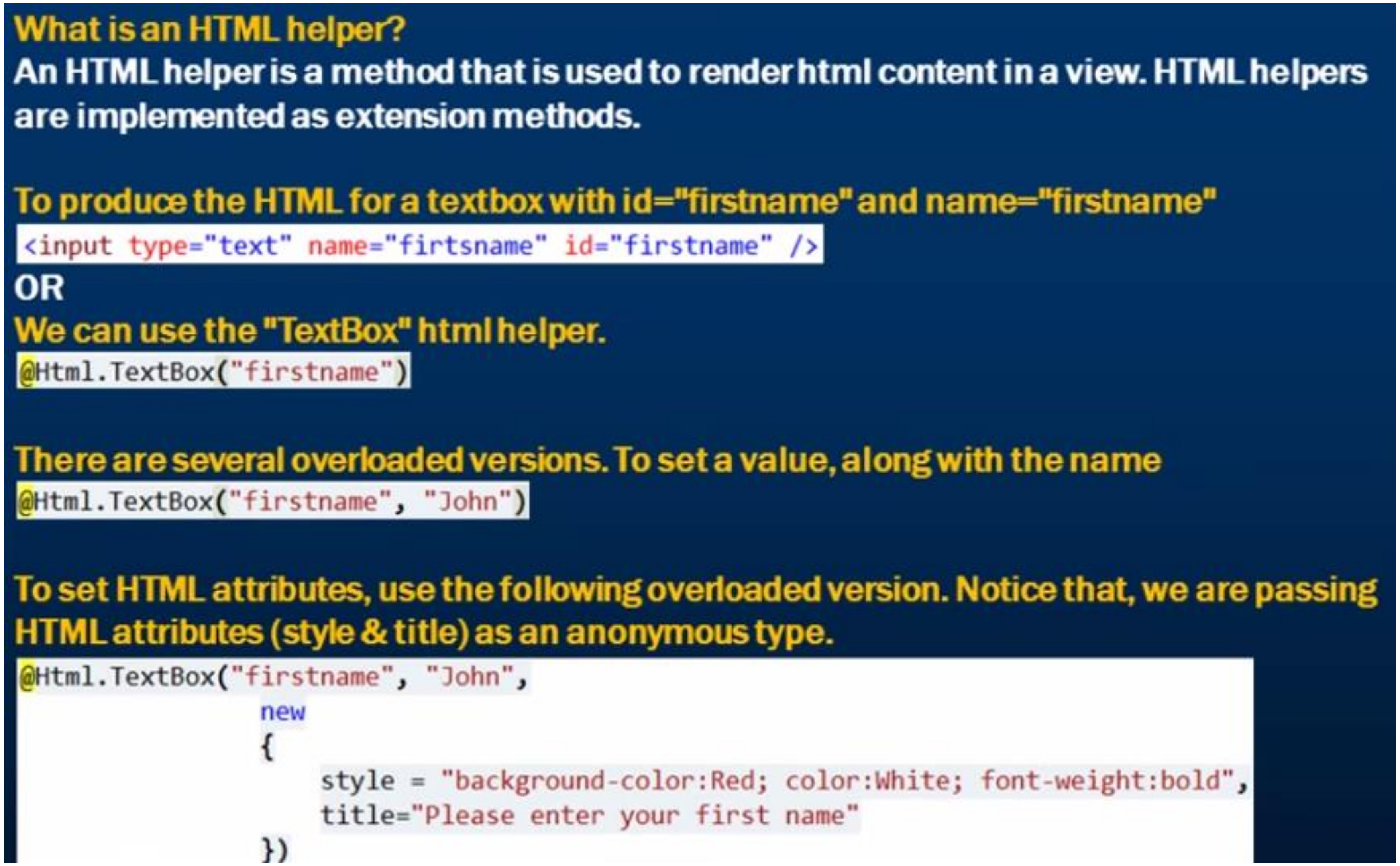
Razor View Vs Web Form View Contd.

Razor View Engine	Web Form View Engine
Razor syntax are easy to learn and much clean than Web Form syntax. Razor uses @ symbol to make the code like as: @Html.ActionLink("SignUp", "SignUp")	Web Form syntax are borrowed from Asp.net Web Forms syntax that are mixed with html and sometimes make a view messy. Webform uses <% and %> delimiters to make the code like as: <%: Html.ActionLink("SignUp", "SignUp") %>
Razor Engine is little bit slow as compared to Webform Engine.	Web Form Engine is faster than Razor Engine.
Razor Engine, doesn't support design mode in visual studio means you cannot see your page look and feel.	Web Form engine support design mode in visual studio means you can see your page look and feel without running the application.
Razor Engine support TDD (Test Driven Development) since it is not depend on System.Web.UI.Page class.	Web Form Engine doesn't support TDD (Test Driven Development) since it depend on System.Web.UI.Page class which makes the testing complex.

Razor View Scaffolding

- ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications.
- Visual Studio includes pre-installed code generators for MVC and Web API projects.
- We add scaffolding to your project when we want to quickly add code that interacts with data models.
- Using scaffolding can reduce the amount of time to develop standard data operations in your project.

HTML Helpers in MVC



What is an HTML helper?

An HTML helper is a method that is used to render html content in a view. HTML helpers are implemented as extension methods.

To produce the HTML for a textbox with id="firstname" and name="firstname"

```
<input type="text" name="firtsname" id="firstname" />
```

OR

We can use the "TextBox" html helper.

```
@Html.TextBox("firstname")
```

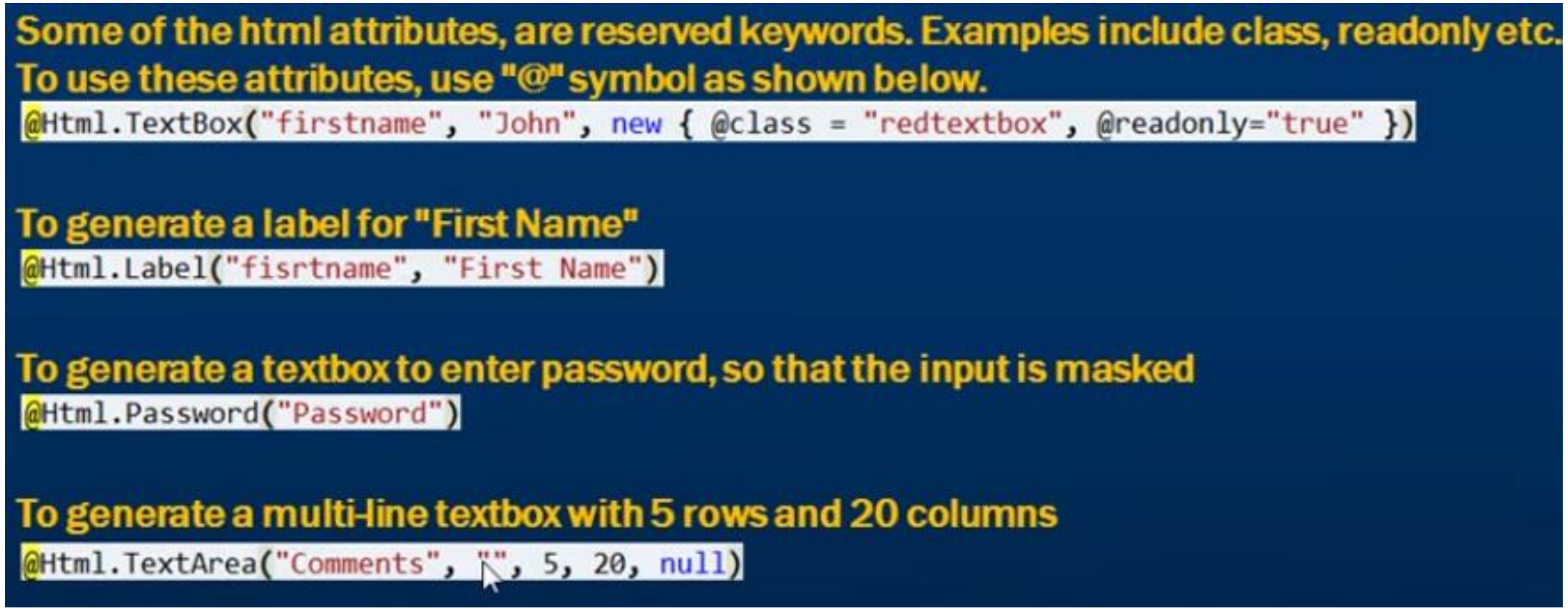
There are several overloaded versions. To set a value, along with the name

```
@Html.TextBox("firstname", "John")
```

To set HTML attributes, use the following overloaded version. Notice that, we are passing HTML attributes (style & title) as an anonymous type.

```
@Html.TextBox("firstname", "John",
                    new
                    {
                        style = "background-color:Red; color:White; font-weight:bold",
                        title="Please enter your first name"
                    })
```

HTML Helpers in MVC



Some of the html attributes, are reserved keywords. Examples include class, readonly etc.
To use these attributes, use "@" symbol as shown below.

```
@Html.TextBox("firstname", "John", new { @class = "redtextbox", @readonly="true" })
```

To generate a label for "First Name"

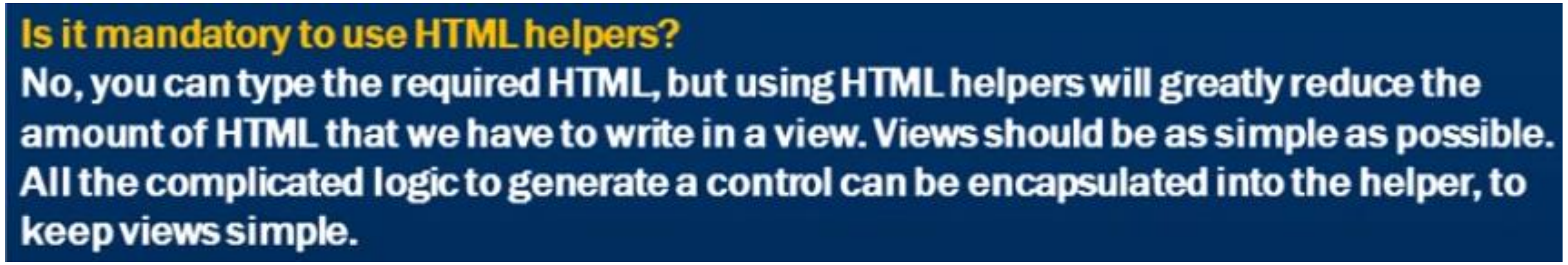
```
@Html.Label("fisrtname", "First Name")
```

To generate a textbox to enter password, so that the input is masked

```
@Html.Password("Password")
```

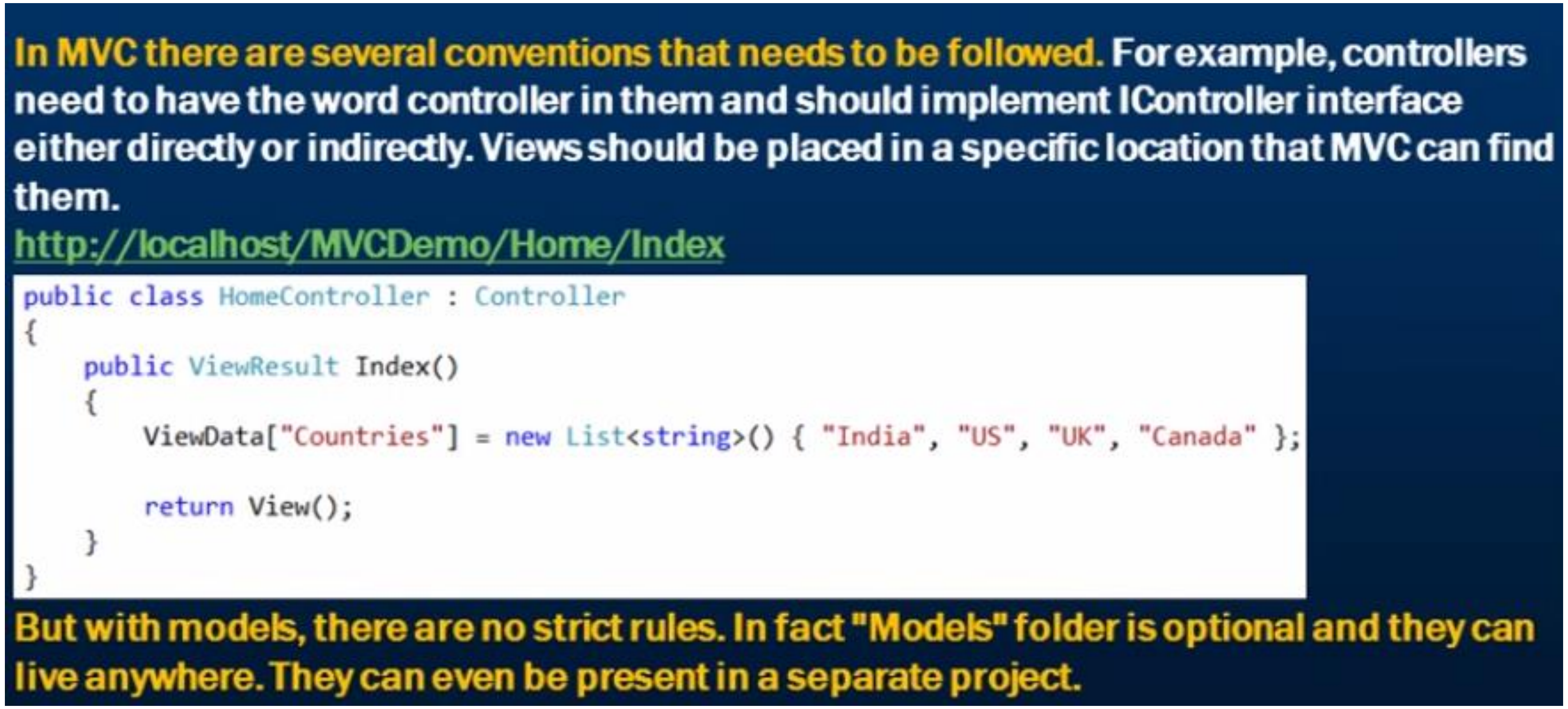
To generate a multi-line textbox with 5 rows and 20 columns

```
@Html.TextArea("Comments", "", 5, 20, null)
```



Is it mandatory to use HTML helpers?
No, you can type the required HTML, but using HTML helpers will greatly reduce the amount of HTML that we have to write in a view. Views should be as simple as possible. All the complicated logic to generate a control can be encapsulated into the helper, to keep views simple.

Using Business Object As Model



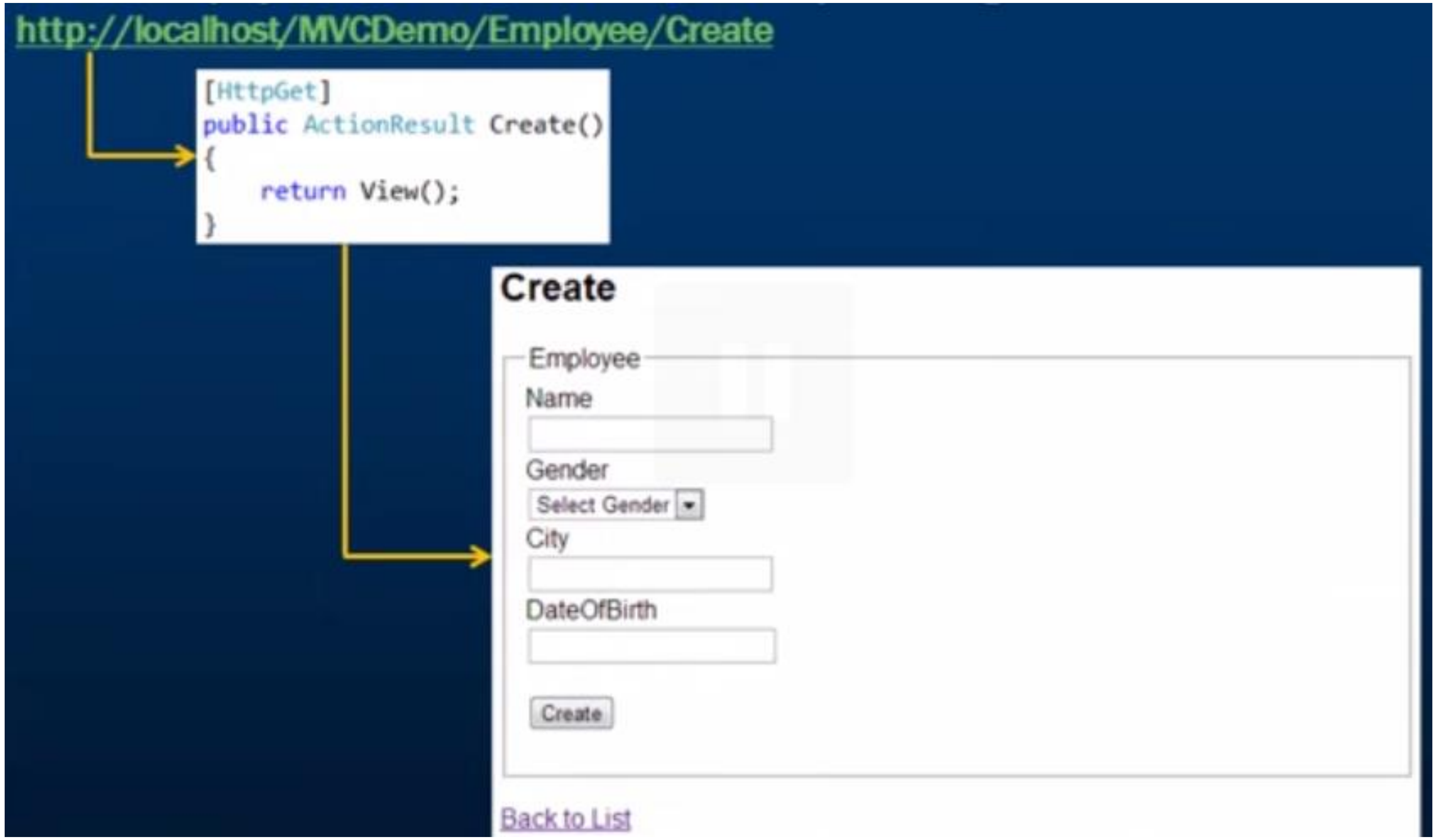
In MVC there are several conventions that needs to be followed. For example, controllers need to have the word controller in them and should implement IController interface either directly or indirectly. Views should be placed in a specific location that MVC can find them.

http://localhost/MVCDemo/Home/Index

```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        ViewData["Countries"] = new List<string>() { "India", "US", "UK", "Canada" };

        return View();
    }
}
```

But with models, there are no strict rules. In fact "Models" folder is optional and they can live anywhere. They can even be present in a separate project.

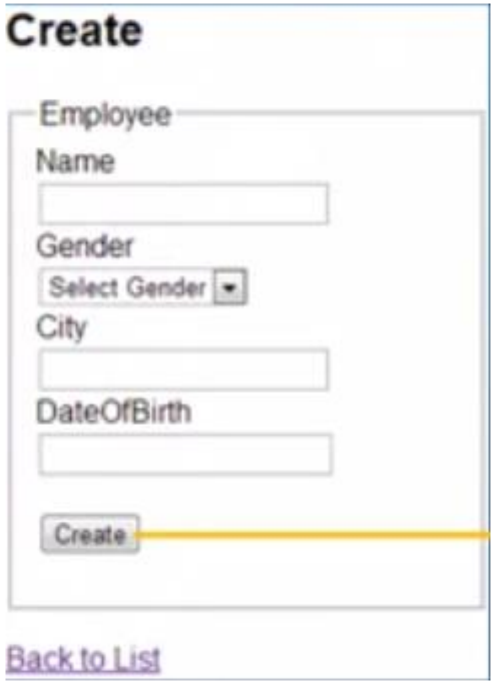
Inserting Data from View



http://localhost/MVCDemo/Employee/Create

```
[HttpGet]
public ActionResult Create()
{
    return View();
}
```

Inserting Data : Passing Data as a Form Collection

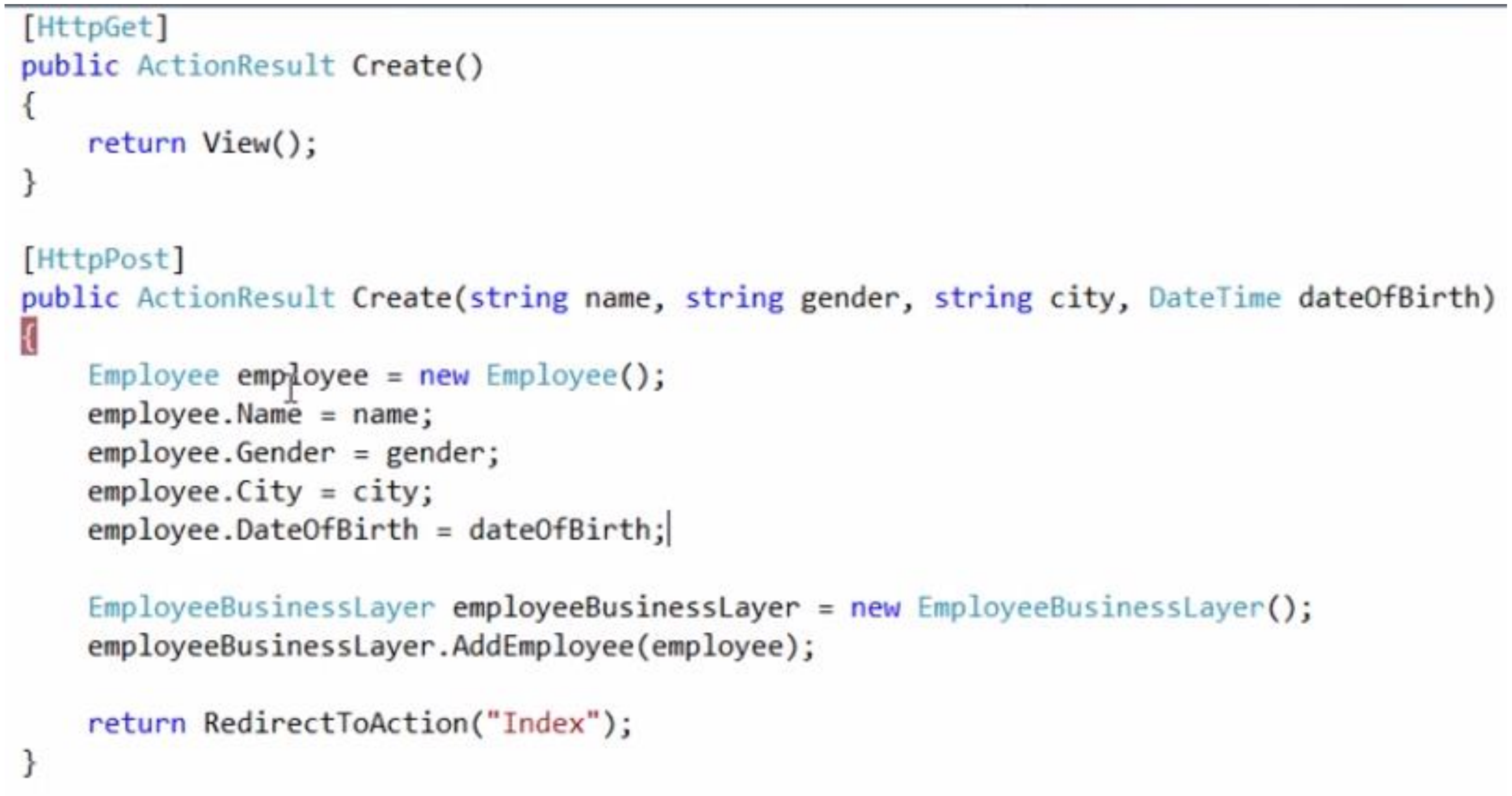


FormCollection class will automatically receive the posted form values in the controller action method, in key/value pairs. Keys & values can be accessed using key names or index. Do we really have to write all the dirty code of retrieving data from FormCollection

Inserting Data : Passing Data as Form Collection

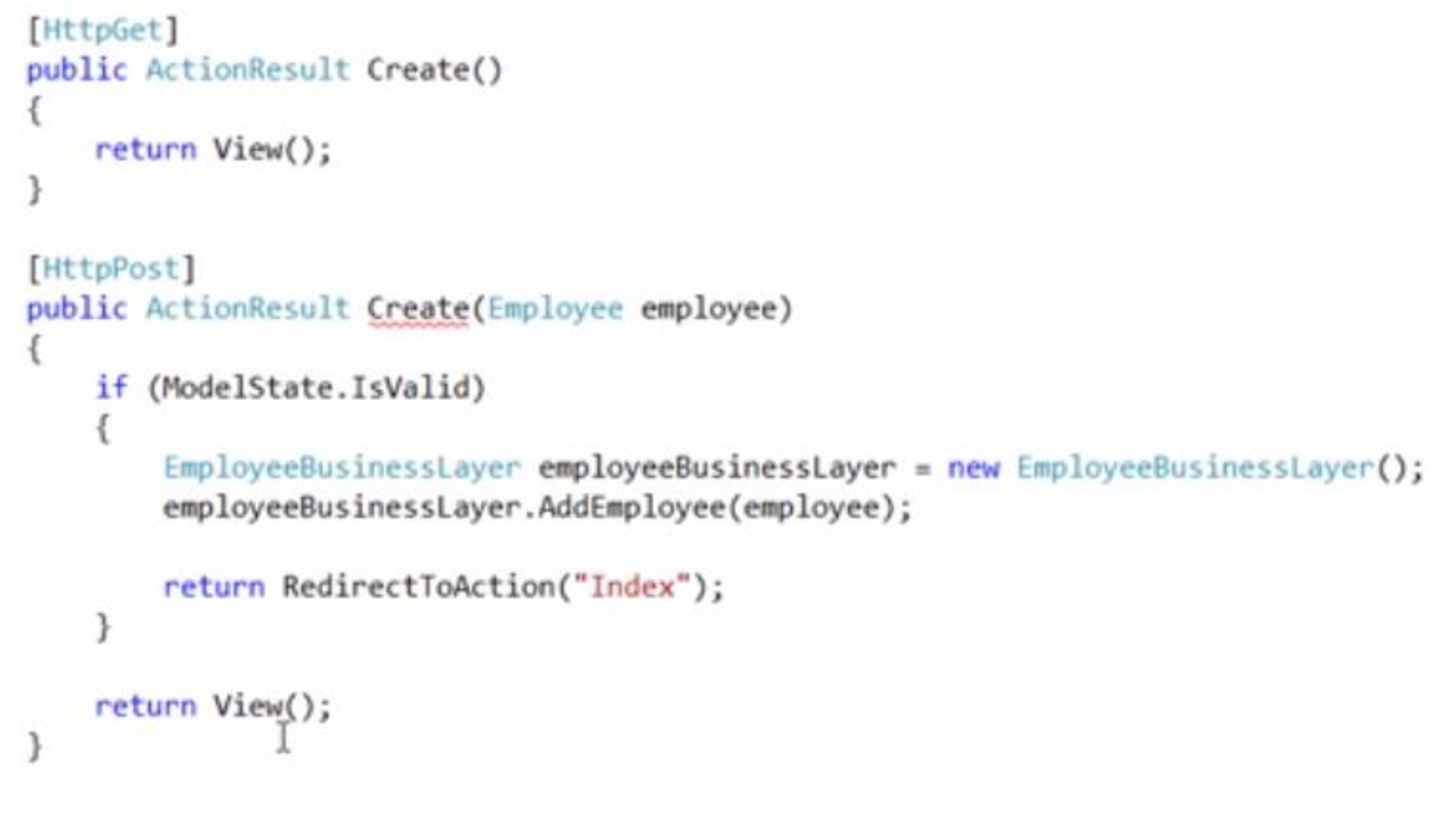
```
[HttpPost]
public ActionResult Create(FormCollection collection)
{
    Employee employee = new Employee();
    employee.Name = collection["Name"].ToString();
    employee.Gender = collection["Gender"].ToString();
    employee.City = collection["City"].ToString();
    employee.DateOfBirth = Convert.ToDateTime(collection["DateOfBirth"]);
    EmployeeBusinessLayer bl = new EmployeeBusinessLayer();
    bl.AddEmployees(employee);
    return RedirectToAction("Index");
}
```

Inserting Data : Passing Data as Action Parameters



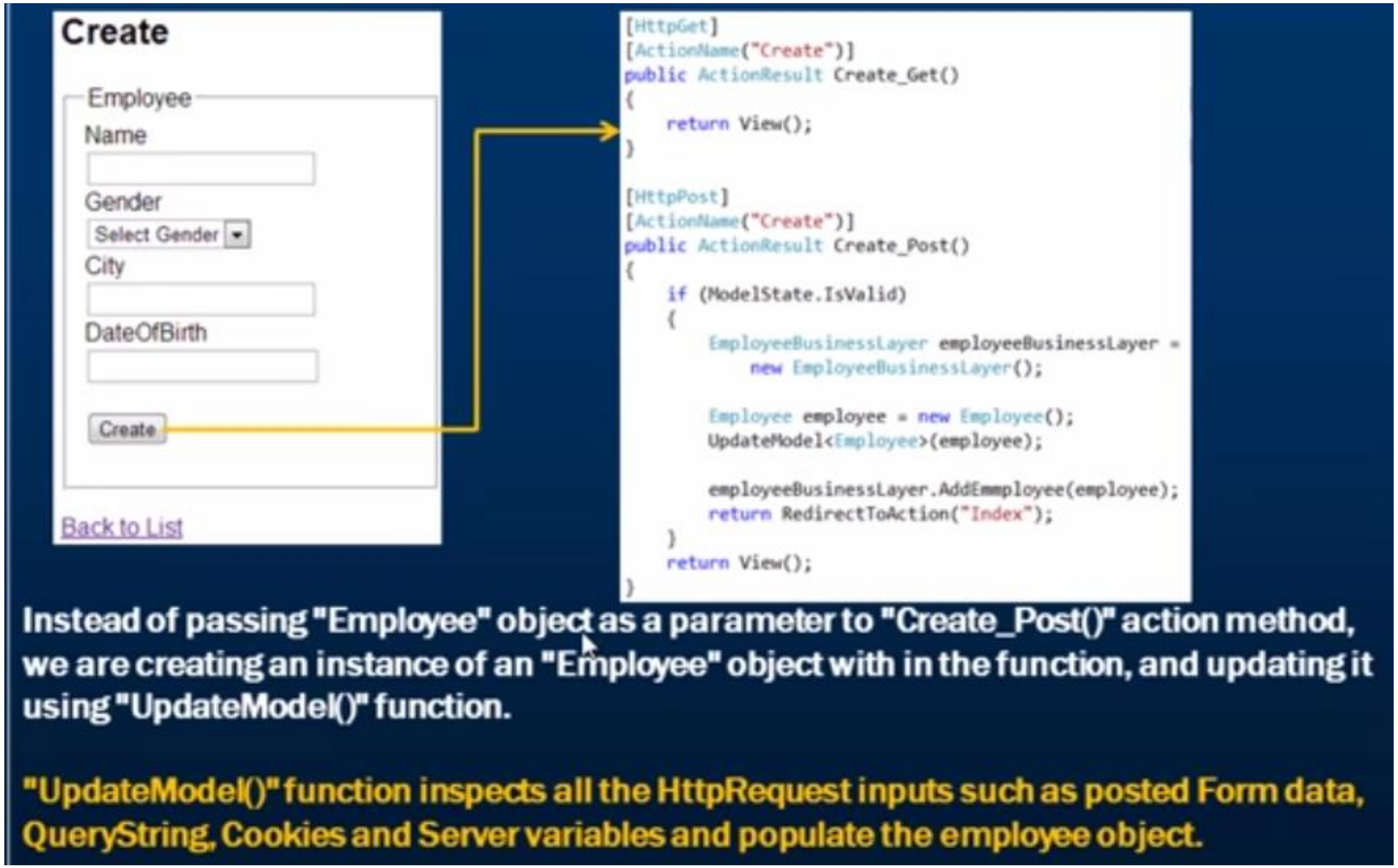
```
[HttpGet]
public ActionResult Create()
{
    return View();
}

[HttpPost]
public ActionResult Create(string name, string gender, string city, DateTime dateOfBirth)
{
    Employee employee = new Employee();
    employee.Name = name;
    employee.Gender = gender;
    employee.City = city;
    employee.DateOfBirth = dateOfBirth;

    EmployeeBusinessLayer employeeBusinessLayer = new EmployeeBusinessLayer();
    employeeBusinessLayer.AddEmployee(employee);

    return RedirectToAction("Index");
}
```

Inserting Data : Passing Data as Employee Object



```
[HttpGet]
public ActionResult Create()
{
    return View();
}

[HttpPost]
public ActionResult Create(Employee employee)
{
    if (ModelState.IsValid)
    {
        EmployeeBusinessLayer employeeBusinessLayer = new EmployeeBusinessLayer();
        employeeBusinessLayer.AddEmployee(employee);

        return RedirectToAction("Index");
    }

    return View();
}
```

Inserting Data : Update Model

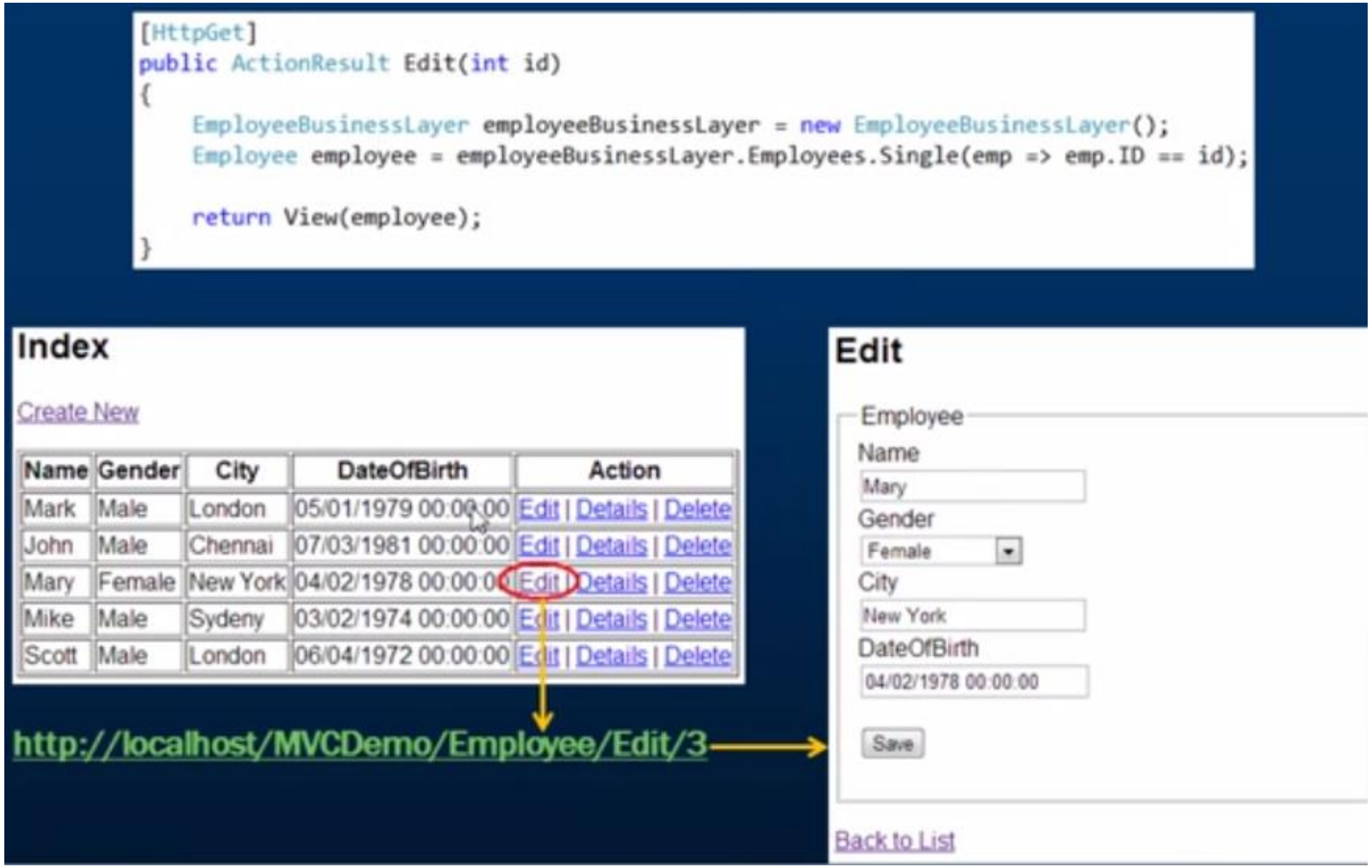


```
[HttpGet]
[ActionName("Create")]
public ActionResult Create_Get()
{
    return View();
}

[HttpPost]
[ActionName("Create")]
public ActionResult Create_Post()
{
    if (ModelState.IsValid)
    {
        EmployeeBusinessLayer employeeBusinessLayer =
            new EmployeeBusinessLayer();

        Employee employee = new Employee();
        UpdateModel<Employee>(employee);

        employeeBusinessLayer.AddEmmployee(employee);
        return RedirectToAction("Index");
    }
    return View();
}
```

Instead of passing "Employee" object as a parameter to "Create_Post()" action method, we are creating an instance of an "Employee" object with in the function, and updating it using "UpdateModel()" function.

"UpdateModel()" function inspects all the HttpRequest inputs such as posted Form data, QueryString, Cookies and Server variables and populate the employee object.

Updating A Record



```
[HttpGet]
public ActionResult Edit(int id)
{
    EmployeeBusinessLayer employeeBusinessLayer = new EmployeeBusinessLayer();
    Employee employee = employeeBusinessLayer.Employees.Single(emp => emp.ID == id);

    return View(employee);
}
```

Index

Create New

Name	Gender	City	DateOfBirth	Action
Mark	Male	London	05/01/1979 00:00:00	Edit | Details | Delete
John	Male	Chennai	07/03/1981 00:00:00	Edit | Details | Delete
Mary	Female	New York	04/02/1978 00:00:00	Edit | Details | Delete
Mike	Male	Sydeny	03/02/1974 00:00:00	Edit | Details | Delete
Scott	Male	London	06/04/1972 00:00:00	Edit | Details | Delete

http://localhost/MVCDemo/Employee/Edit/3

Edit

Employee

Name: Mary

Gender: Female

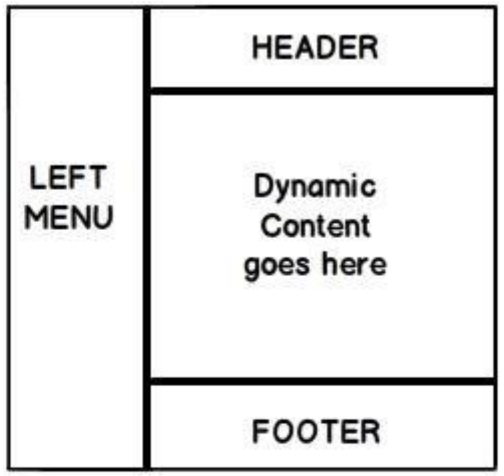
City: New York

DateOfBirth: 04/02/1978 00:00:00

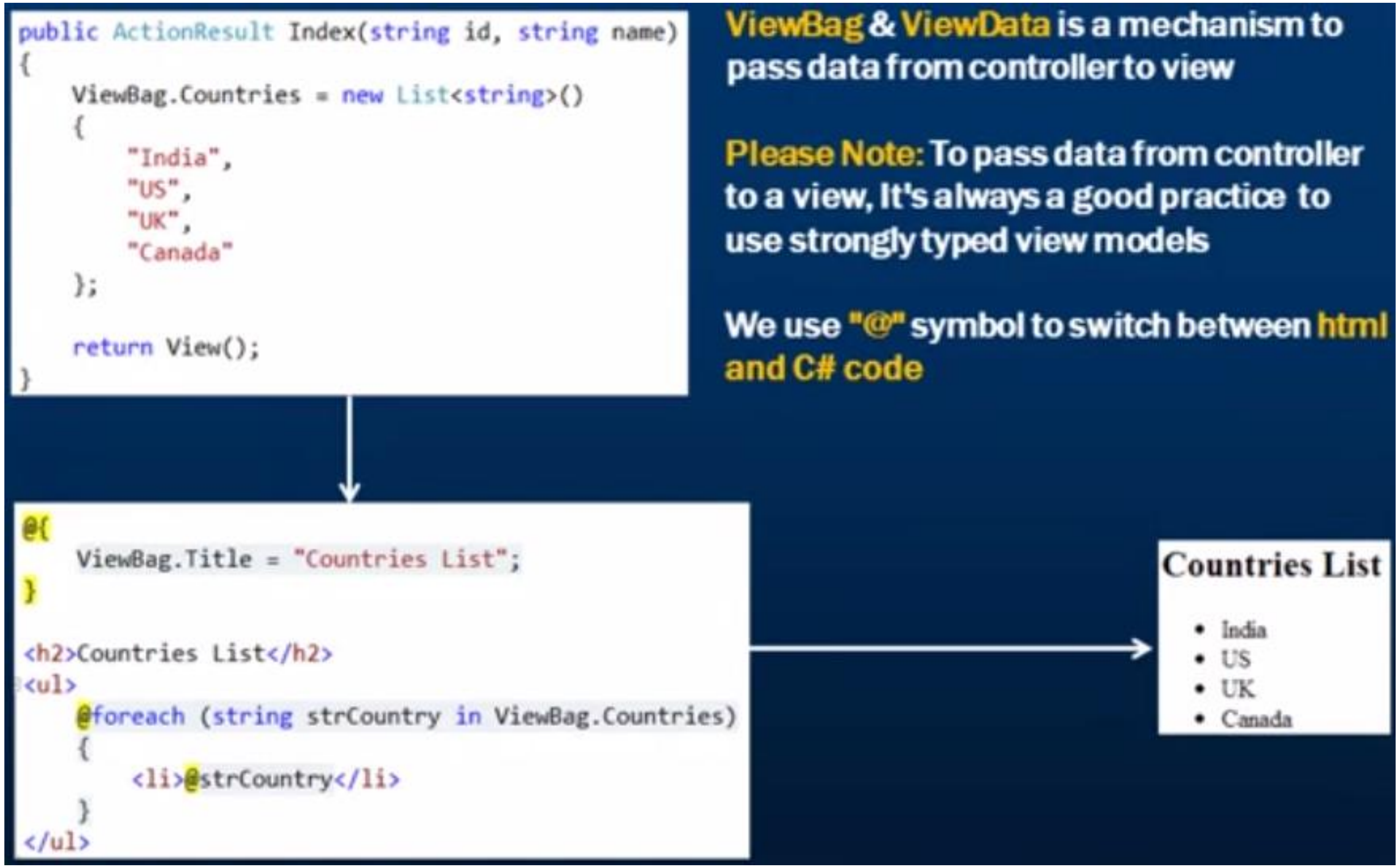
Back to List

Partial Views

- A partial view enables you to define a view that will be rendered inside a parent view.
- Partial views are implemented as ASP.NET user controls (.ascx).



Views In MVC



```
public ActionResult Index(string id, string name)
{
    ViewBag.Countries = new List<string>()
    {
        "India",
        "US",
        "UK",
        "Canada"
    };

    return View();
}
```

```
@{
    ViewBag.Title = "Countries List";
}

<h2>Countries List</h2>
<ul>
    @foreach (string strCountry in ViewBag.Countries)
    {
        <li>@strCountry</li>
    }
</ul>
```

ViewBag & ViewData is a mechanism to pass data from controller to view

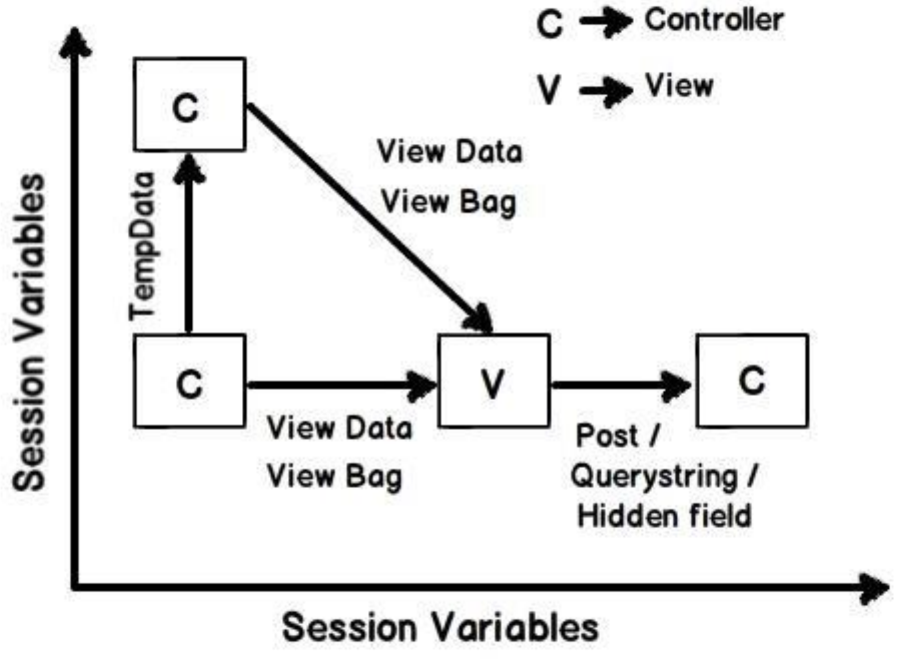
Please Note: To pass data from controller to a view, It's always a good practice to use strongly typed view models

We use "@" symbol to switch between html and C# code

Countries List

- India
- US
- UK
- Canada

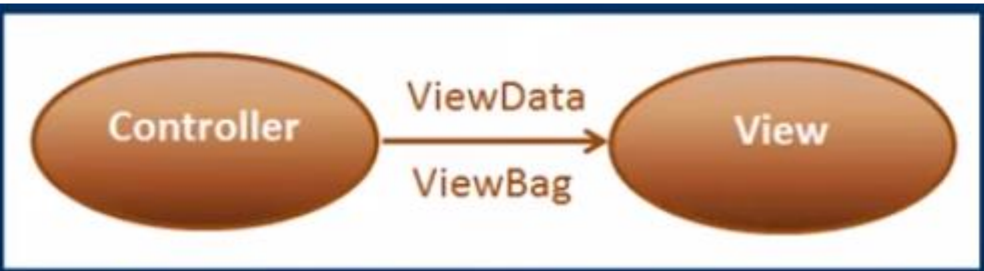
Data Sharing : Temp Data, View Data, View Bag



Temp data - 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.

View data - Helps to maintain data when you move from controller to view.
View Bag - It's a dynamic wrapper around view data. When you use Viewbag type, casting is not required. It uses the dynamic keyword internally.

View Data & View Bag



Both ViewData and ViewBag are used to pass data from a controller to a view. ViewData is a dictionary of objects that are stored and retrieved using strings as keys.

```
ViewData["YourKey"] = "SomeData";
```

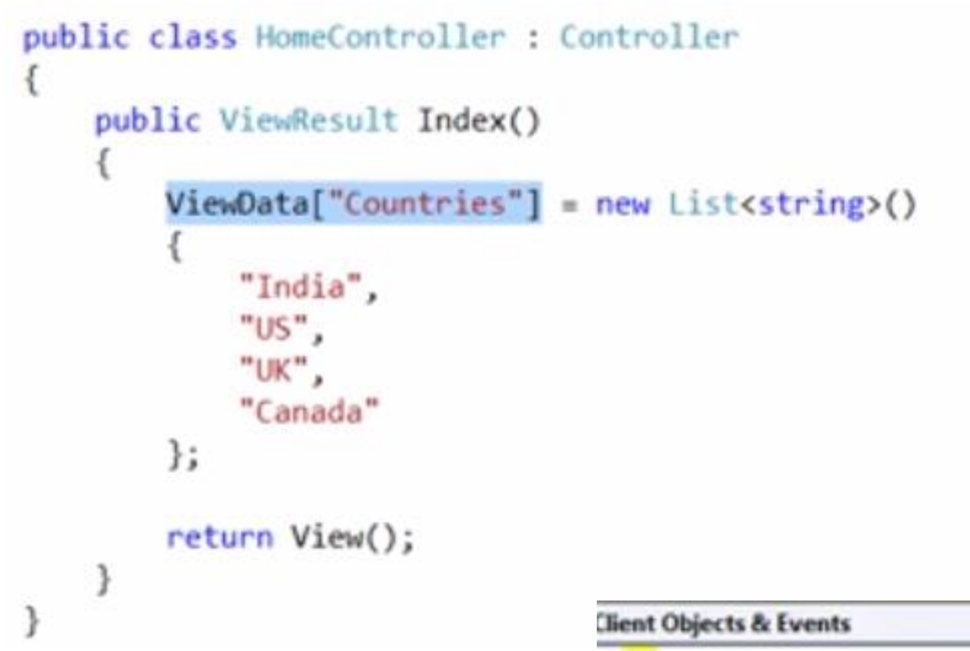
```
ViewBag.YourProperty = "SomeData";
```

ViewBag uses the dynamic feature that was introduced in to C# 4. It allows an object to have properties dynamically added to it.

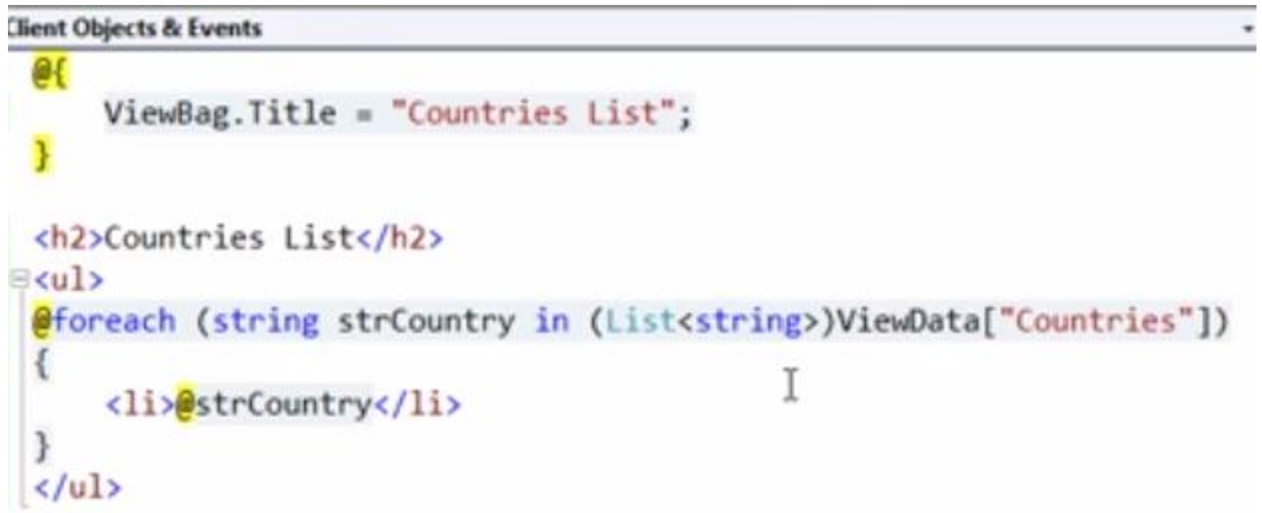
Both ViewData & ViewBag does not provide compile time error checking. For example, if you mis-spell keys or property names, you wouldn't get any compile time error. You get to know about the error only at runtime. Internally ViewBag properties are stored as name/value pairs in the ViewData dictionary.

To pass data from controller to a view, It's always a good practice to use strongly typed view models over ViewBag & ViewData. Strongly typed view models provide compile time error checking.

View Data



```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        ViewData["Countries"] = new List<string>()
        {
            "India",
            "US",
            "UK",
            "Canada"
        };

        return View();
    }
}
```



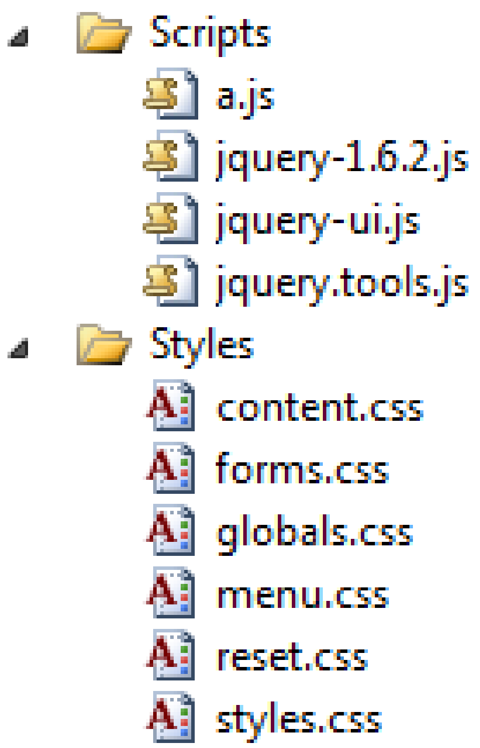
```
@{
    ViewBag.Title = "Countries List";
}

<h2>Countries List</h2>
<ul>
@foreach (string strCountry in (List<string>)ViewData["Countries"])
{
    <li>@strCountry</li>
}
</ul>
```

Bundling and Minification

Bundling and Minification

- Improve loading performance of JavaScript and CSS
  - Reduce # and size of HTTP requests
- Works by convention (no configuration required)
- Fully customizable and extensible

Bundling and Minification



- Scripts
  - a.js
  - jquery-1.6.2.js
  - jquery-ui.js
  - jquery.tools.js
- Styles
  - content.css
  - forms.css
  - globals.css
  - menu.css
  - reset.css
  - styles.css

```
<link href="styles/reset.css" rel="Stylesheet" />
<link href="styles/styles.css" rel="Stylesheet" />
<link href="styles/content.css" rel="Stylesheet" />
<link href="styles/globals.css" rel="Stylesheet" />
<link href="styles/forms.css" rel="Stylesheet" />
<link href="styles/menu.css" rel="Stylesheet" />
```

```
<script src="scripts/js"></script>
<link href="styles/css" rel="stylesheet" />
```

URL Resolution Enhancements

- Razor now resolves ~/ within all standard HTML attributes
- Today you write:

```
<script src=”@Url.Content(“~/Scripts/Site.js”)”></script>
```

- Razor now allows you to just write:

```
<script src=”~/Scripts/Site.js”)”></script>
```

Conditional Attribute Enhancements

- Today you write:

```
@{
    string myClass = null;
    if (someCondition) {
        myClass = ”shinyFancy”;
    }
}
<div @{if (myClass != null) { <text>class=”@myClass”</text> } }>Content</div>
```

- Now you can write:

```
@{
    string myClass = null;
    if (someCondition) {
        myClass = ”shinyFancy”;
    }
}
<div class=”@myClass”>Content</div>
```

- Will automatically omit attribute name if value is null

MVC 4 vs. MVC 5

MVC 4	MVC 5
Released on Aug 15, 2012	Released on 17 October 2013
Runs on .Net 4.0, 4.5 and with Visual Studio 2010SP1 & Visual Studio 2012	Runs on .Net 4.5, 4.5.1 and with Visual Studio 2013
Authentication Filter not available.	Authentication Filter is available.
Filter overrides feature not available.	Filter overrides feature is available.
Attribute routing feature is not available.	Attribute routing feature is available.
Bootstrap is not available.	Bootstrap is available.
ASP.NET Web API	ASP.NET Web API2
Enhancements to default project templates	Asp.Net Identity
jQuery Mobile is used for Mobile project template	It also provided HTML functionality also.
Task support for Asynchronous Controllers	Task support for Synchronous and Asynchronous Controllers
Asp.Net Identity is not present	Asp.Net Identity is present
Authentication filters – doesn’t run prior to authorization filters in the ASP.NET MVC pipeline	Authentication filters - run prior to authorization filters in the ASP.NET MVC pipeline

Thanks a lot for having me

infogain

Thank You