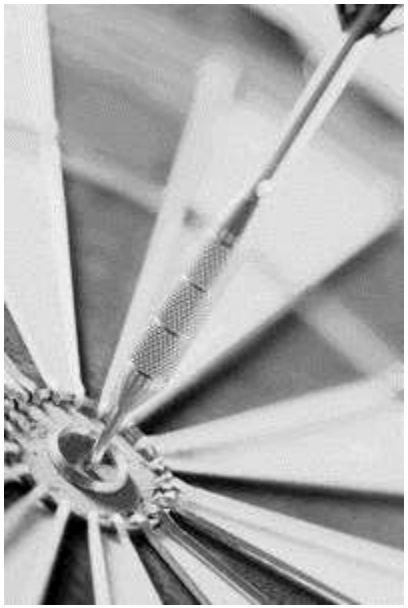
Introduction to MVC

Agenda

- What is MVC.
- MVC History.
- MVC-Component Interaction.
- View Engine.
- ViewData vs ViewBag vs TempData.
- Routing.
- Best Practices.

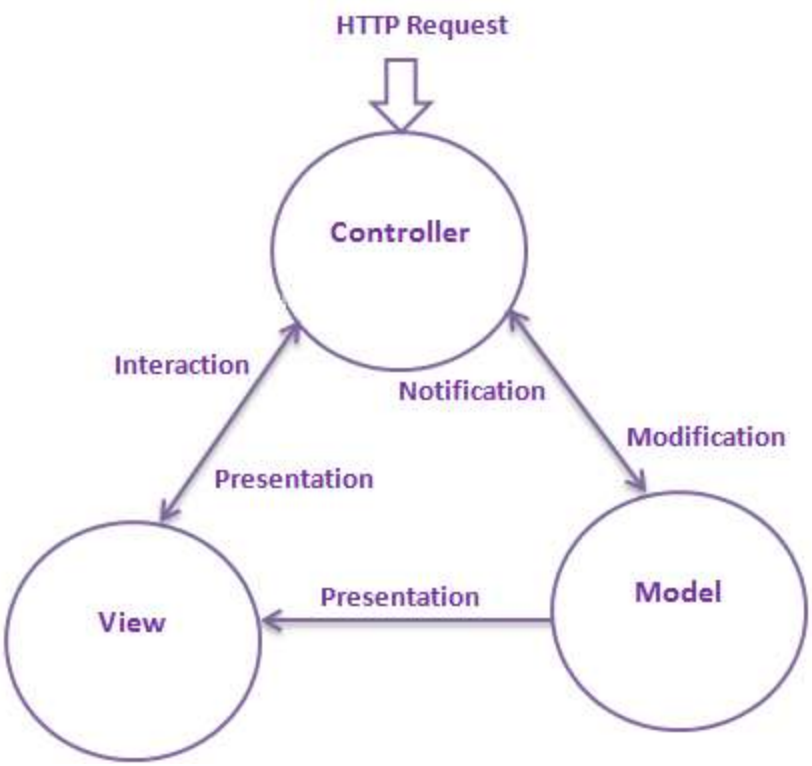
Objectives

- Upon completing this module, you will be able to:
  - Describe What is MVC
  - Describe MVC-Component Interaction
  - Describe View Engine
  - Describe ViewData vs ViewBag vs TempData
  - Describe Best Practices



The MVC

- Model–view–controller (MVC) is a software architecture pattern that splits an application into three main aspects : Model, View and Controller
- Originally formulated in the late 1970_s by Trygve Reenskaug as part of the Smalltalk
- MVC pattern forces a separation of concerns within an application **for example**, separating data access logic and business logic from the UI.



•4

The ASP.NET MVC History

- ASP.NET MVC 1.0
  - In February 2007, Scott Guthrie ("ScottGu") of Microsoft sketched out the core of ASP.NET MVC
  - Released on 13 March 2009
- ASP.NET MVC 2.0
  - Released just one year later, on 10 March 2010
- ASP.NET MVC 3.0
  - Released on 13 January 2011
- ASP.NET MVC 4.0
  - Released on 15 August 2012
- ASP.NET MVC 5.0
  - Released on 17 Oct 2014

•5

Benefits of MVC

- **Separation of Concerns**

  Separation of Concern is one of the core advantages of ASP.NET MVC . The MVC framework provides a clean separation of the UI , Business Logic , Model or Data.
- **Testability**
- ASP.NET MVC framework provides better testability of the Web Application and good support for the test driven development too.
- **Lightweight**
- ASP.NET MVC framework doesn’t use View State and thus reduces the bandwidth of the requests to an extent.
- **Full features of ASP.NET**
- One of the key advantages of using ASP.NET MVC is that it is built on top of ASP.NET framework and hence most of the features of the ASP.NET like membership providers , roles etc can still be used.

Model-View-Controller

Model:

- Set of classes that describes the data we are working with as well as the business
- Rules for how the data can be changed and manipulated
- May contain data validation rules
- Often encapsulate data stored in a database as well as code used to manipulate the data

View:

- Defines how the application's user interface (UI) will be displayed
- The view is only responsible for displaying the data, that is received from the controller as the result.

Controller:

- The Controller is responsible for controlling the application logic and acts as the coordinator between the View and the Model.
- The Controller receive input from users via the View, then process the user's data with the help of Model and passing the results back to the View.

View

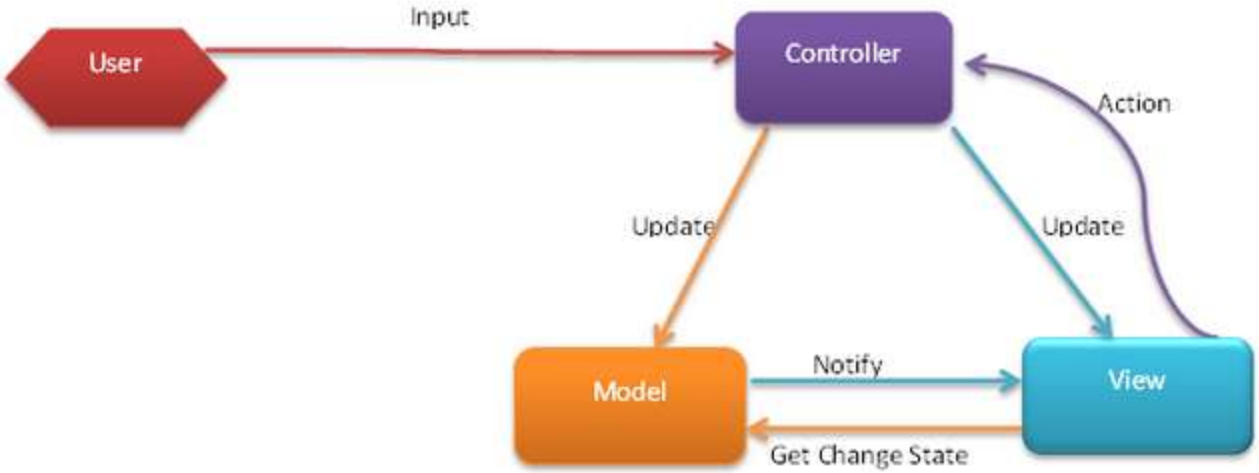
- Defines how the application's user interface (UI) will be displayed. The view represents the presentation of the application.

- The view object refers to the model. In ASP.NET the view is the set of web pages presented by a web application.

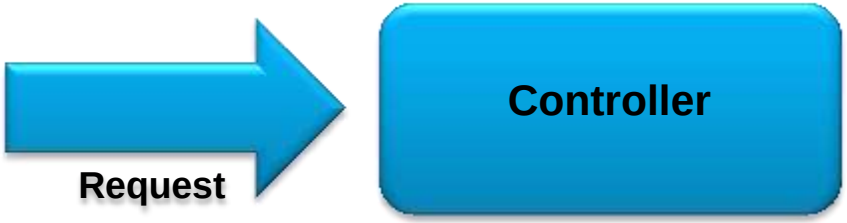
Controller

- The core MVC component
- A set of classes that handles
  - Communication from the user
  - Overall application flow
  - Application-specific logic
- Every controller has one or more "Actions"

MVC-Component Interaction



MVC-Component Interaction



Step 1

Incoming request directed to **Controller**

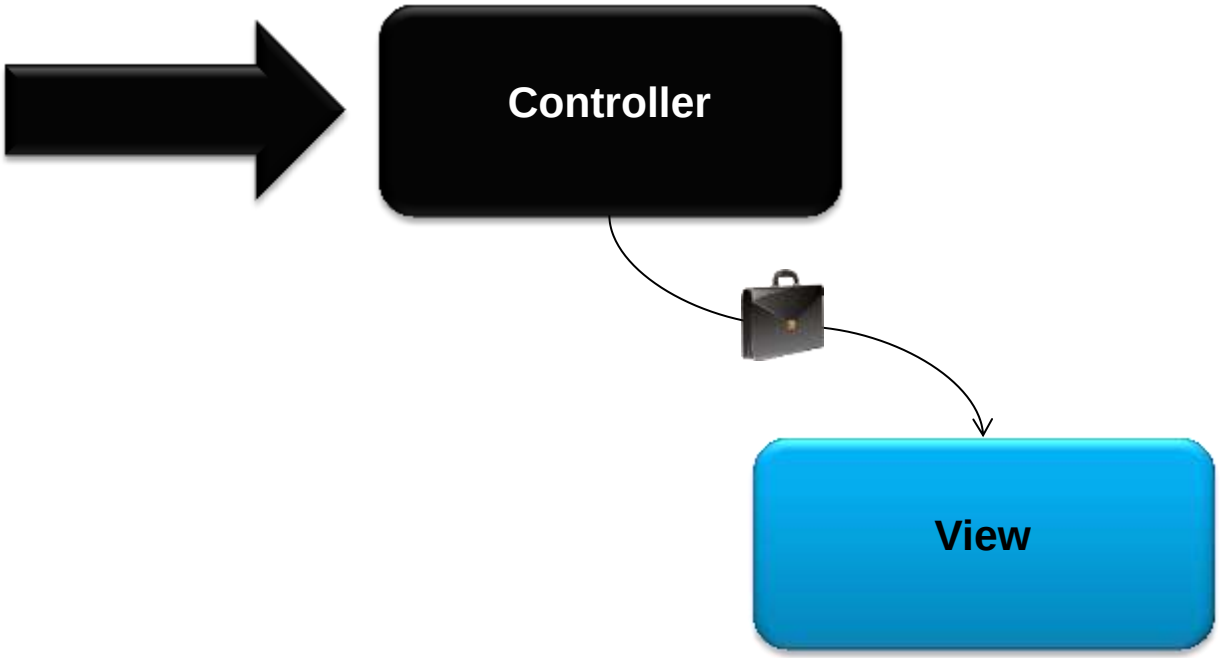
MVC-Component Interaction

•Model

Step 2

Controller processes request and forms a data **Model**

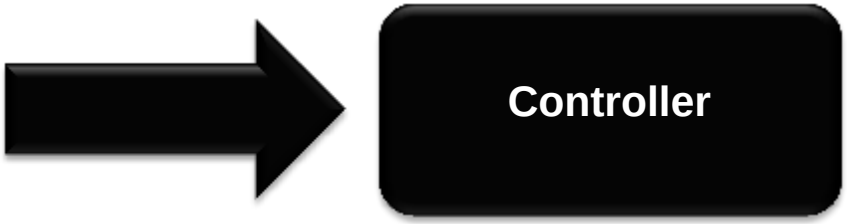
MVC-Component Interaction

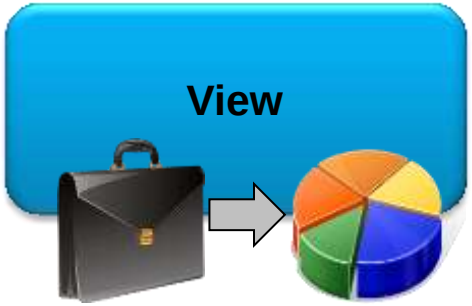


Step 3

Model is passed to **View**

MVC-Component Interaction

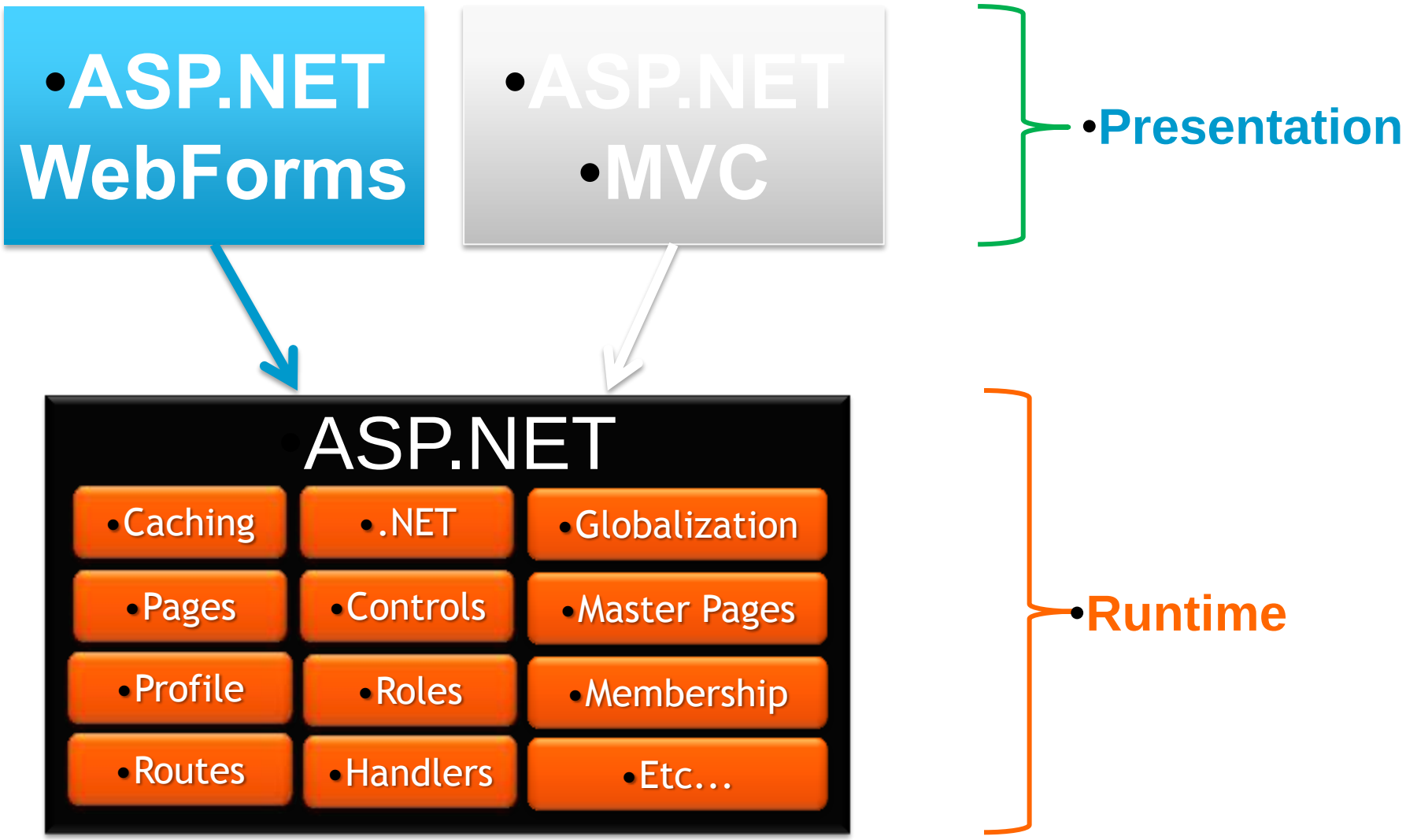




Step 4

View transforms **Model** into appropriate output format

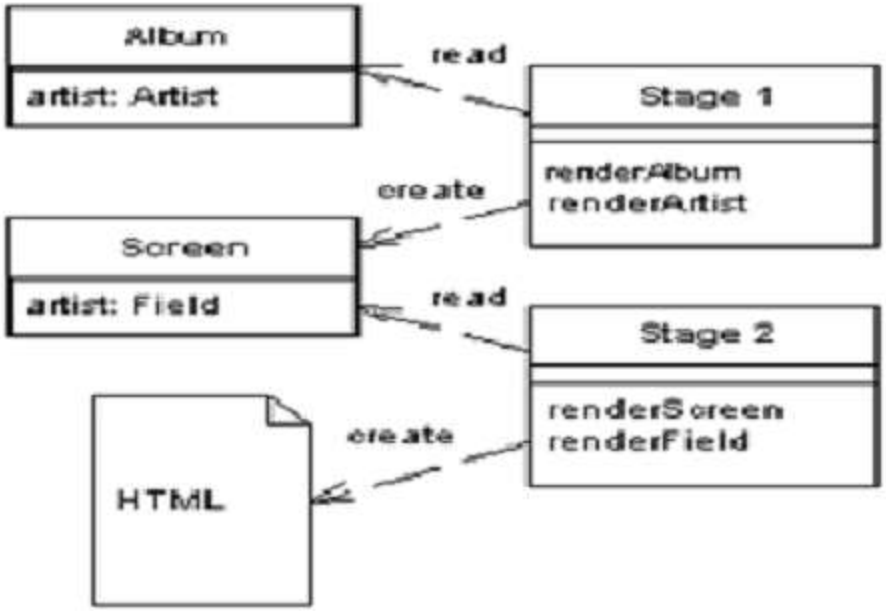
ASP.NET Core



•15

View Engine

View Engine is responsible for rendering the view into html form to the browser.



• Asp.net MVC support Web Form(ASPX) and Razor View Engine.

•There are many third party view engines (like Spark, Nhaml etc.) also available for Asp.net MVC. Now, Asp.net MVC is open source and can work with other third party view engines like Spark, Nhaml.

Razor View

Razor is an ASP.NET programming syntax used to create dynamic web pages with the C# or VB.NET programming languages. Razor was in development in June 2010 and was released for Microsoft Visual Studio 2010 in January 2011.

The Razor syntax is a template markup syntax, based on the C# programming language, that enables the programmer to use an HTML construction workflow. Instead of using the ASP.NET .ASPX markup syntax with<%= %> symbols to indicate code blocks, Razor syntax starts code blocks with a @ character and does not require explicit closing of the code-block.

ASPX vs Razor

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.
By default, Razor Engine prevents XSS attacks(Cross-Site Scripting Attacks) means it encodes the script or html tags like <,> before rendering to view.	Web Form Engine does not prevent XSS attacks means any script saved in the database will be fired while rendering the page
Razor Engine is little bit slow as compared to Webform Engine.	Web Form Engine is faster than Razor Engine.

Create New Project

- Create New Project
- Select Project Template : Asp.net MVC 4 Web Application
- Different Template available while creating MVC application .
  Those template related to the type of application to create
  Example
    Empty : Development from Scratch
    Internet : For Internet Application(Web)
    Intranet : For Intranet Application
- View Engine : It is mechanism to convert the data to HTML to send to client.
  ASPX : ASPX Pages (Use for Legacy Application & backward compabilty)
    Razor : .CSHTML Files
- Razor is preferred view engine for the MVC

File Structure & Naming Standards

MVC uses a standard directory structure and file naming standards, a very important part of MVC application development:

- Inside the ROOT directory of the application, there should be 3 directories each for model, view and Controller.
- Apart from 3 directories, there must be a *Global.asax* file in root folder, and a *web.config* like a traditional ASP.NET application.

- **Root** [directory]
  - **Controller** [directory]
    - Controller CS files
  - **Models** [directory]
    - Model CS files
  - **Views** [directory]
    - View aspx/ascx files
  - *Global.asax*
  - *Web.config*

Routing

•The ASP.NET Routing is responsible for mapping incoming browser requests to particular MVC controller actions.

•A new ASP.NET MVC application is already configured to use ASP.NET Routing. ASP.NET Routing is setup in two places.

1. It is enabled in application's Web configuration file (Web.config file). There are four sections in the configuration file that are relevant to routing:
   I. system.web.httpModules section,
   II. system.web.httpHandlers section,
   III. system.webserver.modules section,
   IV. system.webserver.handlers section

2. The route table is created during the Application Start event(global.asax).

Routing

The Route Table:

```
routes.MapRoute(
    "Default",                                          // Route name
    "{controller}/{action}/{id}",                       // URL with parameter
    new { controller = "Home", action = "Index", id = "" }  // Parameter defaults
);
```

- The default route table contains a single route (named Default).
- Imagine that you enter the following URL into your web browser's address bar:
- /Home/Index/3
- The Default route maps this URL to the following parameters:
- controller = Home
- action = Index
- id = 3
- The following code is executed:
- HomeController.Index(3)

Controllers & Actions

- The core component of the MVC pattern
- All the controllers should be available in a folder by name Controllers
- Controller naming standard should be "nameController" (convention)
- Routers instantiate controllers in every request
  - All requests are mapped to a specific action
- Every controller should inherit Controller class
  - Access to Request (context) and HttpContext

Actions

- Actions are the ultimate request destination
  - Public controller methods
  - Non-static
  - No return value restrictions
- Actions typically return an ActionResult

```
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";
    return View();
}
```

Action Results

- Controller action response to a browser request
- Inherits from the base ActionResult class
- Different results types:

Name	Framework Behavior	Producing Method
JsonResult	Returns data in JSON format	Json
RedirectResult	Redirects the client to a new URL	Redirect / RedirectPermanent
ViewResult PartialViewResult	Response is the responsibility of a view engine	View / PartialView

Action Parameters

- ASP.NET MVC maps the data from the HTTP request to action parameters in few ways:
  - Routing engine can pass parameters to actions
    - http://localhost/Users/NikolayIT
    - Routing pattern: Users/{username}
  - URL query string can contains parameters
    - /Users/ByUsername?username=NikolayIT

```
public ActionResult ByUsername(string username)
{
    return Content(username);
}
```

Action Selectors

The [AcceptVerbs] attribute can be applied to action methods in a controller so that the appropriate overloaded method is invoked for a given request.

```
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Edit(int id) {
   // code snipped
   // this is invoked when viewing the edit page
}

[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(int id, FormCollection formValues) {
   // code snipped
   // this is invoked when POSTing data to the edit page
}
```

Action Filters

- Sometimes it is required 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.
- The ASP.NET MVC framework includes several action filters:
  - OutputCache – This action filter caches the output of a controller action for a specified amount of time.
  - HandleError – This action filter handles errors raised when a controller action executes.
  - Authorize – This action filter enables you to restrict access to a particular user or role.

Limitation of HandleError

- It has no support to log the exceptions since it suppressed the error once it is handled.
- It only catch 500 Http error and doesn't catch other HTTP errors like 404,401 etc.
- It doesn't catch the errors that occur outside the controllers like errors occurs in model.
- It returns error view even if error occurred in AJAX calls that is not useful in the client-side. It would be great to return a piece of JSON in AJAX exceptions .

Partial Views

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

- When a partial view is instantiated, it gets its own copy of the ViewDataDictionary object that is available to the parent view. The partial view therefore has access to the data of the parent view. However, if the partial view updates the data, those updates affect only the partial view's ViewData object. The parent view's data is not changed.

- By using partial view we can render a view inside a parental view and to create reusable content in the project.

ViewData vs ViewBag

ViewData and ViewBag are used for the same purpose to transfer data from controller to view. Both life lies only in current request.

ViewData Example

```
//Controller Code
public ActionResult Index()
{
    List<string> Student = new List<string>();
    Student.Add("Jignesh");
    Student.Add("Tejas");
    Student.Add("Rakesh");

    ViewData["Student"] = Student;
    return View();
}
//page code
<ul>
   <% foreach (var student in ViewData["Student"] as List<string>)
      { %>
   <li><%: student%></li>
   <% } %>
</ul>
```

ViewBag Example

```
//Controller Code
public ActionResult Index()
{
    List<string> Student = new List<string>();
    Student.Add("Jignesh");
    Student.Add("Tejas");
    Student.Add("Rakesh");

    ViewBag.Student = Student;
    return View();
}
//page code
<ul>
   <% foreach (var student in ViewBag.Student)
      { %>
   <li><%: student%></li>
   <% } %>
</ul>
```

TempData

- TempData is a dictionary which is derived from TempDataDictionary class.
- it hold data between two consecutive http requests. TempData help us to transfer data between controllers or between actions.

TempData Example

```
//Controller Code
public ActionResult Index()
{
    List<string> Student = new List<string>();
    Student.Add("Jignesh");
    Student.Add("Tejas");
    Student.Add("Rakesh");

    TempData["Student"] = Student;
    return View();
}
//page code
<ul>
    <% foreach (var student in TempData["Student"] as List<string>)
        { %>
    <li><%: student%></li>
    <% } %>
</ul>
```

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 is also work with .net framework 3.5 and above	ViewBag is only work with .net framework 4.0 and above	TempData is also work 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
It value become null if redirection is 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 is only work during the current and subsequent request

Best Practices

- Chose RAZOR over ASPX view engine. This is not a best practice, but a **strong recommendation.**
- Razor is cleaner, more intuitive, more fluid, more expressive.

WebForms View Engine

```
<div>
    <ul>
    <% foreach (var item in Model) { %>
        <li>
            <%= item.Name %>
            <%= item.Surname %>
        </li>
    <% } %>
    </ul>
</div>
```

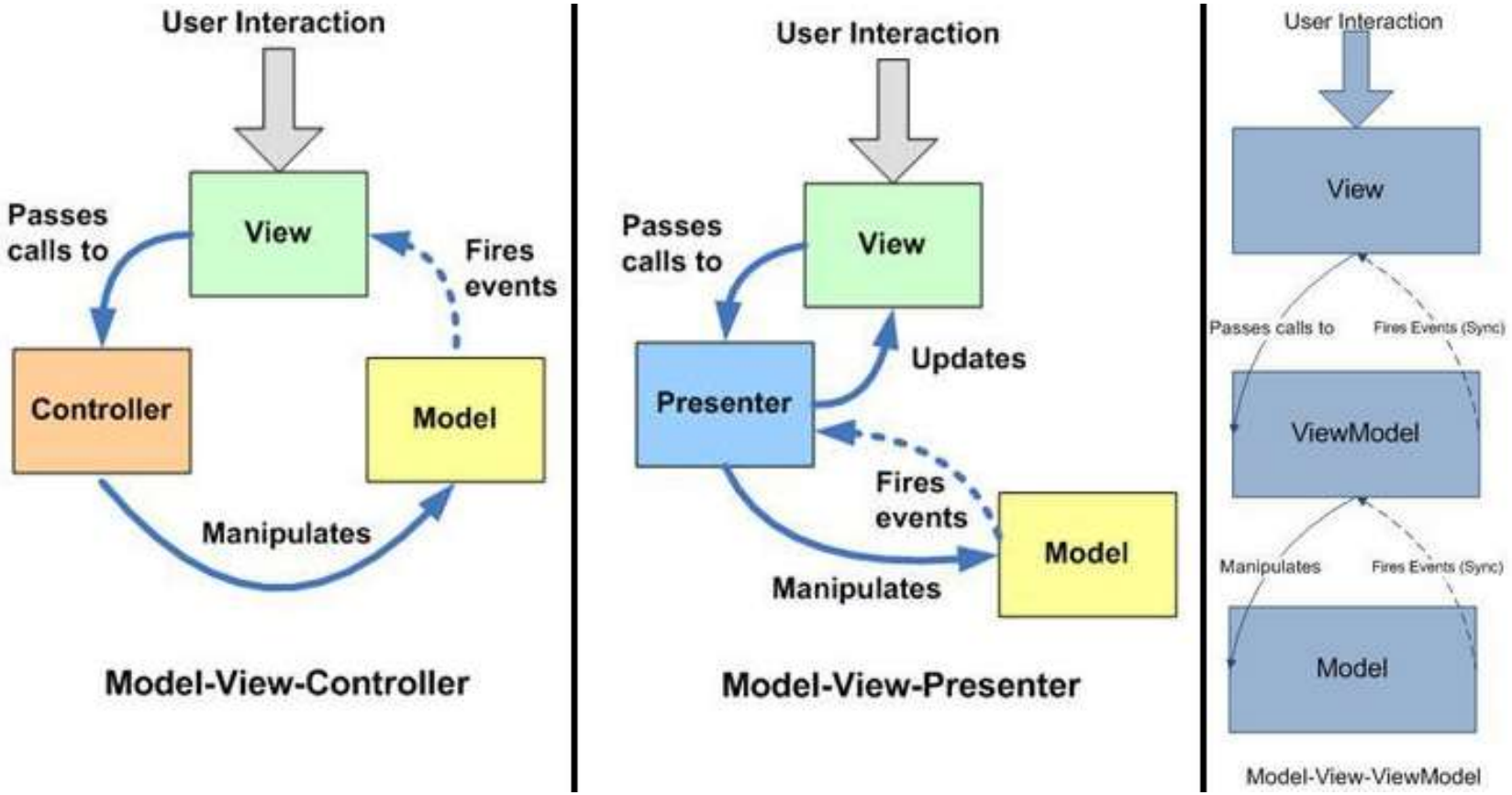
Razor View Engine

```
<ul>
    @foreach (var item in Model) {
        <li>
            @item.Name
            @item.Surname
        </li>
    }
</ul>
```

Best Practices

- Minimize code in Views
- Keep your logic out of your views!
- View must be simple and dumb!
- If you have at least one control flow (if/else), write Html Helper…

MVC vs MVP vs MVVM



Questions and Comments

- What questions or comments do you have?