Contents

[1. Explain MVC (Model-View-Controller) in general? 2](#_Toc453574963)

[2. What is ASP.NET MVC? 2](#_Toc453574964)

[3. Difference between ASP.NET MVC and ASP.NET WebForms? 2](#_Toc453574965)

[4. What are the Core features of ASP.NET MVC? 2](#_Toc453574966)

[5. Can you please explain the request flow in ASP.NET MVC framework? 2](#_Toc453574967)

[6. What is the difference between each version of MVC 2, 3 , 4, 5 and 6? 3](#_Toc453574968)

[7. What is Routing in ASP.NET MVC? 4](#_Toc453574969)

[8. What are Action Methods in ASP.NET MVC? 4](#_Toc453574970)

[9. Explain the role of Model in ASP.NET MVC? 4](#_Toc453574971)

[10. What is ActionResult? 4](#_Toc453574972)

[7. What is the difference between ViewData, ViewBag and TempData? 6](#_Toc453574973)

[**Passing data from Controller to View in ASP.NET MVC** 6](#_Toc453574974)

[Understanding Routing in ASP.NET MVC 7](#_Toc453574975)

[How can we do validations in MVC? 7](#_Toc453574976)

[One of the easiest ways of doing validation in MVC is by using data annotations. Data annotations are nothing but attributes which can be applied on model properties. For example, in the below code snippet we have a simple Customer class with a property customercode. 7](#_Toc453574977)

[This CustomerCode property is tagged with a Required data annotation attribute. In other words if this model is not provided customer code, it will not accept it. 7](#_Toc453574978)

[Entity Framework 8](#_Toc453574979)

MVC (Latest Version MVC 6)

1. Explain MVC (Model-View-Controller) in general?

MVC (Model-View-Controller) is an architectural software pattern that basically decouples various components of a web application. By using MVC pattern, we can develop applications that are more flexible to changes without affecting the other components of our application.

* “Model”, is basically domain data.
* “View”, is user interface to render domain data.
* “Controller”, translates user actions into appropriate operations performed on model.

1. What is ASP.NET MVC?

ASP.NET MVC is a web development framework from Microsoft that is based on MVC (Model-View-Controller) architectural design pattern. Microsoft has streamlined the development of MVC based applications using ASP.NET MVC framework.

1. Difference between ASP.NET MVC and ASP.NET WebForms?

ASP.NET Web Forms uses Page controller pattern approach for rendering layout, whereas ASP.NET MVC uses Front controller approach. In case of Page controller approach, every page has its own controller i.e. code-behind file that processes the request. On the other hand, in ASP.NET MVC, a common controller for all pages processes the requests.

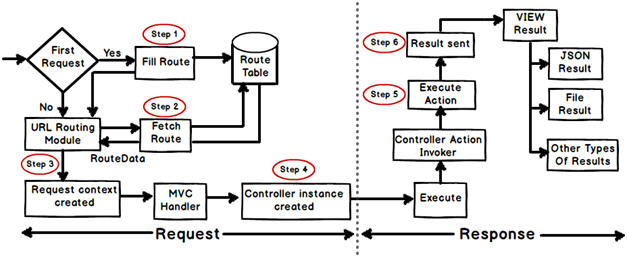
1. What are the Core features of ASP.NET MVC?

Core features of ASP.NET MVC framework are:

* Clear separation of application concerns (Presentation and Business Logic)
* An extensible and pluggable framework
* Extensive support for ASP.NET Routing
* Support for existing ASP.NET features

1. Can you please explain the request flow in ASP.NET MVC framework?

Request flow for ASP.NET MVC framework is as follows:  
Request hits the controller coming from client. Controller plays its role and decides which model to use in order to serve the request. Further passing that model to view which then transforms the model and generate an appropriate response that is rendered to client.



1. What is the difference between each version of MVC 2, 3 , 4, 5 and 6?

**MVC 6**

* ASP.NET MVC and Web API has 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 5**

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

**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 3**

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

**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

1. What is Routing in ASP.NET MVC?

In case of a typical ASP.NET application, incoming requests are mapped to physical files such as *.aspx* file. ASP.NET MVC framework uses friendly URLs that more easily describe user’s action but are not mapped to physical files.

ASP.NET MVC framework uses a routing engine, that maps URLs to controller classes. We can define routing rules for the engine, so that it can map incoming request URLs to appropriate controller.

Practically, when a user types a URL in a browser window for an ASP.NET MVC application and presses “go” button, routing engine uses routing rules that are defined in *Global.asax* file in order to parse the URL and find out the path of corresponding controller.

### 8. What are Action Methods in ASP.NET MVC?

I already explained about request flow in ASP.NET MVC framework that request coming from client hits controller first. Actually MVC application determines the corresponding controller by using routing rules defined in *Global.asax*. And controllers have specific methods for each user actions. Each request coming to controller is for a specific Action Method. The following code example, “ShowBooks” is an example of an Action method.

public ViewResult ShowBooks(int id)

{

var computerBook = db.Books.Where(p => P.BookID == id).First();

return View(computerBook);

}

### 9. Explain the role of Model in ASP.NET MVC?

One of the core features of ASP.NET MVC is that it separates the input and UI logic from business logic. Role of Model in ASP.NET MVC is to contain all application logic including validation, business and data access logic except view, i.e., input and controller, i.e., UI logic.

Model is normally responsible for accessing data from some persistent medium like database and manipulate it.

### 10. What is ActionResult?

By default, the Controller actions will return the ActionResult object. We can return various types of results asActionResult, which will decide how the output needs to render on the browser.

1. Content

When we need to return any text from a Controller action, we will use the Content type.

public ActionResult Index()

{

return Content("Hello from Home Controller");

}

1. RedirectToAction

Depending on the input values, we can redirect to another Action. For redirecting to another Action, we will use the RedirectToAction type.

public ActionResult Index()

{

*// Redirect to Verify action inside the Sample Controller*

return RedirectToAction("Verify", "Sample");

}

1. RedirectToRoute

As part of our sample application, we have a custom route defined with the name “sample”. This will route to the Index action inside the Sample Controller.

public ActionResult Index()

{

return RedirectToRoute("sample");

}

1. File is used to return the content of a file to the browser. For our sample, I am returning the *web.config* to the browser.

public ActionResult Index()

{

return File("Web.config", "text/html");

}

1. JSON - We can render the text to the result page or can send it as a file to the client using JSON notation.

public ActionResult Index()

{

return Json("hello from JSON","text/html", JsonRequestBehavior.AllowGet);

}

--- If there is no content type specified, it will download the content as a file.

public ActionResult Index()

{

return Json("hello from JSON", JsonRequestBehavior.AllowGet);

}

**10. What are Action Filters in ASP.NET MVC?**

If we need to apply some specific logic before or after action methods, we use action filters. We can apply these action filters to a controller or a specific controller action. Action filters are basically custom classes that provide a means for adding pre-action or post-action behavior to controller actions.

For example:

1. Authorize filter can be used to restrict access to a specific user or a role.
2. OutputCache filter can cache the output of a controller action for a specific duration.

11. What are Views?

1. View is an ASPX page without having a code behind file
2. All page specific HTML generation and formatting can be done inside view
3. One can use Inline code (server tags ) to develop dynamic pages
4. A request to **view** (ASPX page) can be made only from a controller’s action method

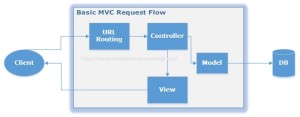
### 7. What is the difference between ViewData, ViewBag and TempData?

In order to pass data from controller to view and in next subsequent request, ASP.NET MVC framework provides different options i.e., ViewData, ViewBag and TempData.

Both ViewBag and ViewData are used to communicate between controller and corresponding view. But this communication is only for server call, it becomes null if redirect occurs. So, in short, it's a mechanism to maintain state between controller and corresponding view.

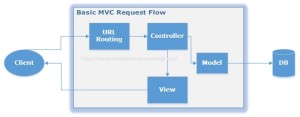
ViewData is a dictionary object while ViewBag is a dynamic property (a new C# 4.0 feature). ViewData being a dictionary object is accessible using strings as keys and also requires typecasting for complex types. On the other hand, ViewBag doesn't have typecasting and null checks,compile time error checking.

TempData is also a dictionary object that stays for the time of an HTTP Request. So, Tempdata can be used to maintain data between redirects, i.e., from one controller to the other controller.

[](http://www.webdevelopmenthelp.net/wp-content/uploads/2014/06/MVC-Request-Flow2.jpg)

**Passing data from Controller to View in ASP.NET MVC**

**ASP.NET MVC** is a framework that facilitates building web applications based on MVC (Model-View-Controller) design pattern. Request coming from client reaches the *Controller* through URL Rewriting Module. *Controller* decides which model to use in order to fulfill the request. Further passing the *Model* data to *View* which then transforms the *Model* data and renders response to client as shown in following basic level request flow diagram.

[](http://www.webdevelopmenthelp.net/wp-content/uploads/2014/06/MVC-Request-Flow2.jpg)

In this ASP.NET MVC Tutorial, we will discuss and implement different options to pass data from ASP.NET MVC *Controller* to *View*. Following are the available options to pass data from a *Controller* to *View* in ASP.NET MVC:

* ViewBag
* ViewData
* TempData

If we want to maintain state between a *Controller* and corresponding *View*- ***ViewData*** and ***ViewBag*** are the available options but both of these options are limited to a single server call (meaning it’s value will be null if a redirect occurs). But if we need to maintain state from one *Controller* to another (redirect case), then ***TempData*** is the other available option.

It’s common that initially it might be a bit difficult for a ASP.NET WebForms developer to digest above flow and need for options to pass data from *Controller* to *View.* Because in WebForms approach, Controller and View are tightly coupled to each other.

**ViewBag Example**

As we discussed earlier that *ViewBag* and *ViewData* serves the same purpose but *ViewBag* is basically a dynamic property (a new C# 4.0 feature) having advantage that it doesn’t have typecasting and null checks.

# Understanding Routing in ASP.NET MVC

Routing plays an important role in an ASP.NET MVC Application execution flow. Basically, it maps request URL to a specific controller action using a Routing Table.In order to describe user’s actions, MVC framework uses friendly URLs against actions instead of mapping it to physical files as in case of an asp.net Web Form application. In a typical ASP.NET Web Form application, request is mapped to a physical file as follows:

*//Displaying all employees*  
*http://locahost:XXXX/Employee.aspx*  
 *//Displaying employee by Id*  
*http://locahost:XXXX/Employee.aspx?Id=10*

As opposite to above approach, MVC framework maps request URL to controller as follows:

*//Displaying all employees*  
*http://locahost:XXXX/****Employee****/*  
 *//Displaying employee by Id*  
*http://locahost:XXXX/****Employee****/****10****/*

In above example URL, “Employee” is a Controller. The point to understand that how this mapping is performed in ASP.NET MVC framework?

What are HTML helpers 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("LastName") %>

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") %>

How can we do validations in MVC?

One of the easiest ways of doing validation in MVC is by using data annotations. Data annotations are nothing but attributes which can be applied on model properties. For example, in the below code snippet we have a simple Customer class with a property customercode.

This CustomerCode property is tagged with a Required data annotation attribute. In other words if this model is not provided customer code, it will not accept it.

public class Customer

{

[Required(ErrorMessage="Customer code is required")]

public string CustomerCode { set; get; }

}

In order to display the validation error message we need to use the ValidateMessageFor method which belongs to the Html helper class.

<% using (Html.BeginForm("PostCustomer", "Home", FormMethod.Post))

{ %>

<%=Html.TextBoxFor(m => m.CustomerCode)%>

<%=Html.ValidationMessageFor(m => m.CustomerCode)%>

<input type="submit" value="Submit customer data" />

<%}%>

Later in the controller we can check if the model is proper or not by using the ModelState.IsValid property and accordingly we can take actions.

public ActionResult PostCustomer(Customer obj)

{

if (ModelState.IsValid)

{

obj.Save();

return View("Thanks");

}

else

{

return View("Customer");

}

}

**Basic Data Annotations**

1. [StringLength(12)]
2. [RegularExpression(@”[A-Za-z0-9.\_%+-]+@[)
3. [Range(0,20)]
4. [Compare(“Password”)]

## Entity Framework