SATHYA TECHNOLOGIES

KANNABABU

Chapter-1

Introduction to MVC

By.B.Kannababu

Q) **What is Design pattern?**

- Design pattern is a readymade solution for the problems that occur in software development
- Every design pattern has **some**specification or set of rules for solving the problems.
- By using the design patterns you can make your code more flexible, reusable and maintainable
- Design Patterns can be used in any Technology or Frameworks or Any Programming Environment

Q) List some of the Popular Design Patterns in SoftwareApplication Development?

- Singleton Design Pattern
- Factory Design pattern
- DAO Design Pattern
- MVC Design pattern

Q) What are the Advantages of Design patterns?

- They are reusable in multiple projects.
- They provide the solutions that help to define the system architecture.
- They capture the software engineering experiences.

- They provide transparency to the design of an application.
- They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
- Helps us to develop a Better Software Application

Q) What is MVC?

MVC is a Design pattern which is used to develop Web Applications

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.

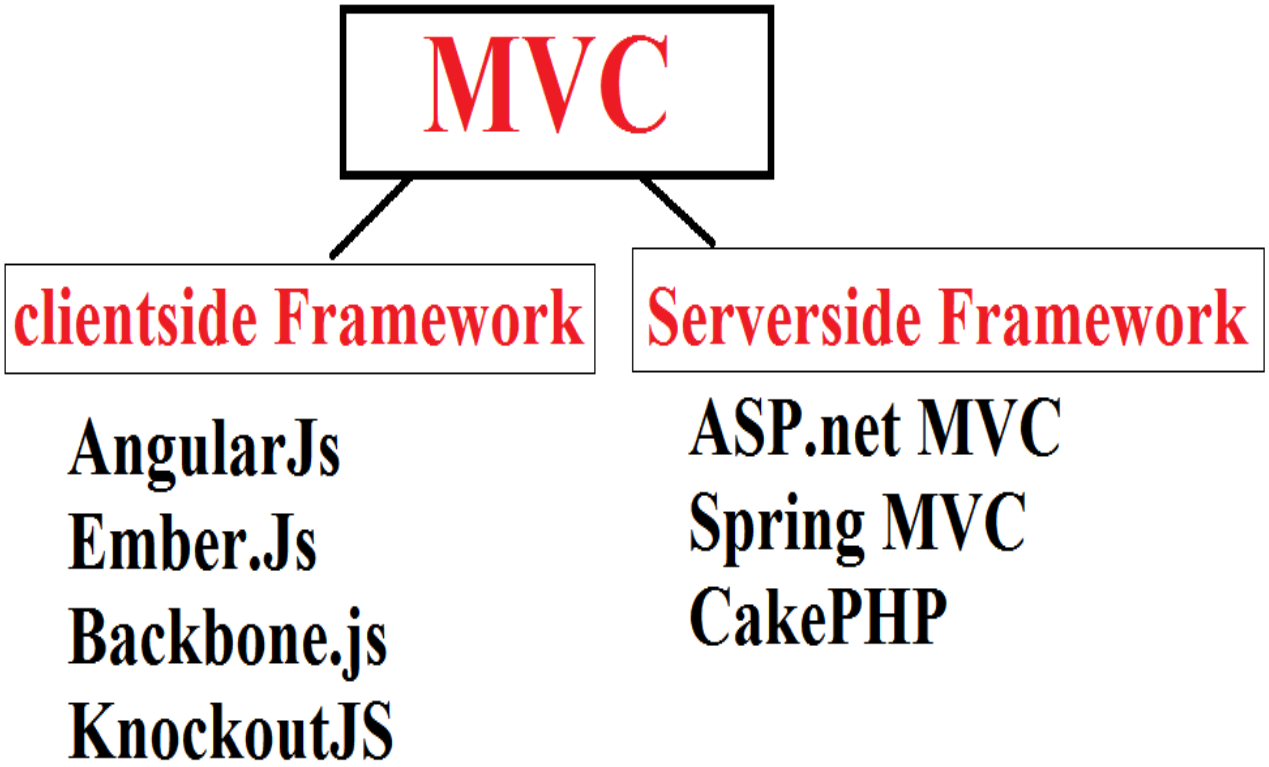
Q) What is Software Framework?

A **framework**, or software **framework**, is a platform for developing software applications

Q) What is the purpose of Software Framework?

The purpose of software framework is to simplify the development environment, allowing developers to dedicate their efforts to the project requirements.

Q) What are the Software Frameworks that are? Developed by using MVC Design Pattern?



Client side Frameworks: - Client side Frameworks will execute on Browser

Server Side Frameworks: - Server Side Frameworks will execute on web server

Q) What is ASP.net MVC?

ASP.net MVC is a software Framework which is used to develop Web applications by using MVC Design Pattern

Q) What is Web Browser?

A *web browser* is a software application for accessing information on the World Wide Web

Browser	Vendor
Internet Explorer	Microsoft
Google Chrome	Google
Mozilla Firefox	Mozilla
Opera	Opera Software
Safari	Apple
Sea Monkey	Mozilla Foundation

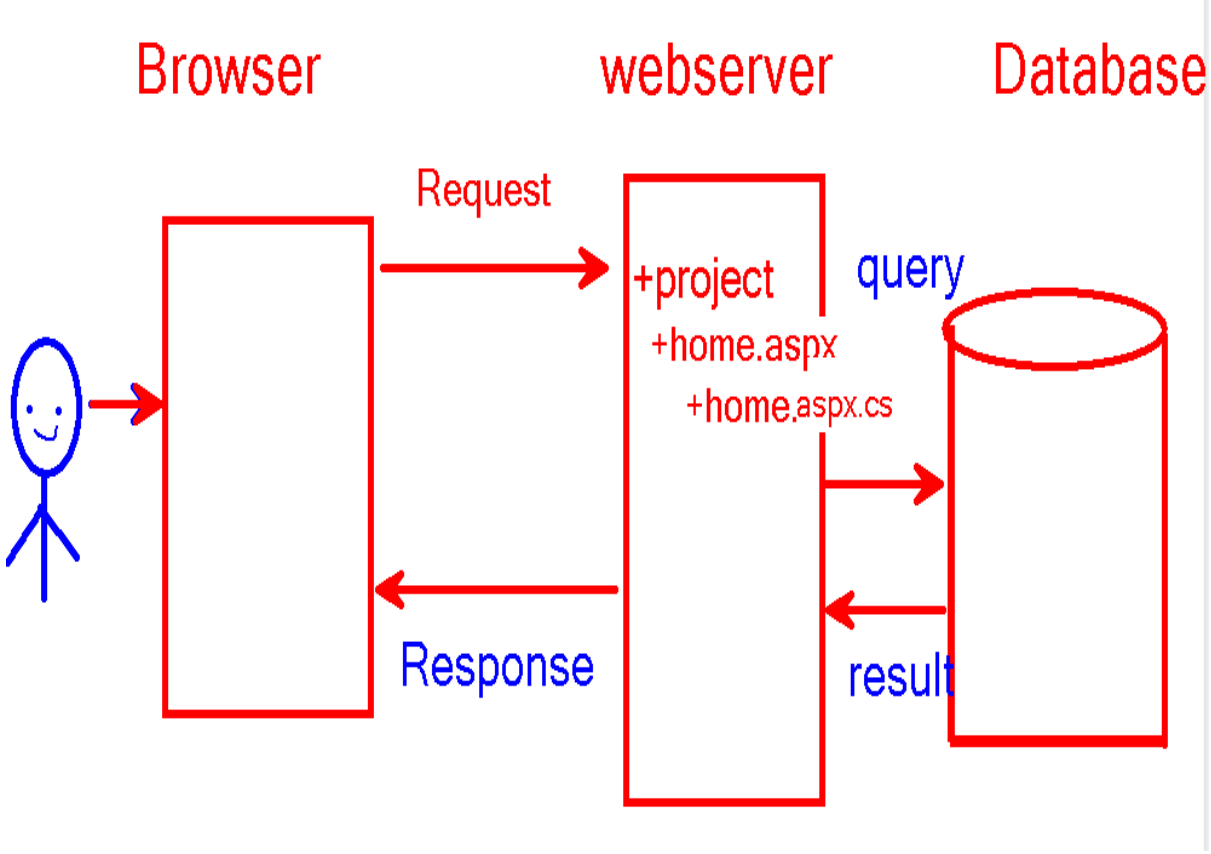
Q) What is Web Server?

A **Web server** is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.

Q) What is the role of Web server?

The basic **function** of a **web server** is to host websites and to deliver **web** content from its hosted websites over the **internet**. During the delivery of **web** pages, **web servers** follow a network protocol known as hyper text transfer protocol (**HTTP**).

Web Application Architecture: - The web application architecture describes the interactions between applications, databases, and middleware systems on the web. It ensures that multiple applications work simultaneously.



1. After developing web application we have to deploy the web application on web server
2. In order to deploy the web application on web server we need to purchase the space on Remote server and we need to purchase Domain name
3. The communication between client and server is due to http protocol
4. After Deploying the application on webserver,Enduser will access the application via Browser and internet
5. Enduser will open Browser and enter url

Q) What is url?

url is uniform resource locator

url is used to to access the resource from server via internet

Ex: - www.facebook.com

http://localhost:1086/home.aspx

6. When end-user will enter url then request will send to web server and web server will search home.aspx and render html o/p to Browser

7. Enduser will enter username and password and click on signin button then request will go to webserver
8. Then server side code will gets executed

   Within signin we will **write C# code+Sqlquery**to interact With Databases
9. Database will execute the query and the result of the query will send to webserver
10. Webserver will send the result to Browser

Chapter-2

Solid Principles

S: Single Responsibility Principle (SRP)

O: Open closed Principle (OSP)

L: Liskov substitution Principle (LSP)

I: Interface Segregation Principle (ISP)

D: Dependency Inversion Principle (DIP)

By.B.Kannababu

The reason behind most unsuccessful applications

Developers start building applications with good and tidy designs using their knowledge and experience. But over time, applications might develop bugs. The application design must be altered for every change request or new feature request. After some time we might need to put in a lot of effort, even for simple tasks and it might require a full working knowledge of the entire system. But we can't blame change or new feature requests. They are part of the software development. We can't stop them or refuse them either. So who is the culprit here? Obviously it is the design of the application.

The following are the design flaws that cause the damage in software, mostly.

1. Putting more stress on classes by assigning more responsibilities to them. (A lot of functionality not related to a class.)
2. Forcing the classes to depend on each other. If classes are dependent on each other (in other words tightly coupled), then a change in one will affect the other.

3. Spreading duplicate code in the system/application.

Solution

- Choosing the correct architecture (in other words MVC, 3-tier, Layered, MVP, MVVP and so on).
- Following Design Principles.
- Choosing correct Design Patterns to build the software based on its specifications.

Intro to SOLID principles

SOLID principles are the design principles that enable us to manage with most of the software design problems. Robert C. Martin compiled these principles in the 1990s. These principles provide us ways to move from tightly coupled code and little encapsulation to the desired results of loosely coupled and encapsulated real needs of a business properly. SOLID is an acronym of the following.

S: Single Responsibility Principle (SRP)

O: Open closed Principle (OSP)

L: Liskov substitution Principle (LSP)

I: Interface Segregation Principle (ISP)

D: Dependency Inversion Principle (DIP)

S: Single Responsibility Principle (SRP)

- **SRP says "Every software module should have only one reason to change".**

SRP says that a class should have only one responsibility and not multiple.

This means that every class, or similar structure, in your code should have only one job to do. Everything in that class should be related to a single purpose. Our class should not be like a Swiss knife wherein if one of them needs to be changed then the entire tool needs to be altered. It does not mean that your classes should only contain one method or property. There may be many members as long as they relate to the single responsibility.

The Single Responsibility Principle gives us a good way of identifying classes at the design phase of an application and it makes you think of all the ways a class can change.

A good separation of responsibilities is done only when we have the full picture of how the application should work.

What is likely to change?

- Software changes because users ask for changes. They ask for changes because something in their life has changed, and there is a gap between what they have and what they need.

O: Open/Closed Principle

The Open/closed Principle says "A software module/class is open for extension and closed for modification".

Here "Open for extension" means, we need to design our module/class in such a way that the new functionality can be added only when new requirements are generated. "Closed for modification" means we have already developed a class and it has gone through unit testing. We should then not alter it until we find bugs. As it says, a class should be open for extensions, we can use inheritance to do this. Okay, let's dive into an example.

```
public class Rectangle{
  public double Height {get;set;}
  public double Wight {get;set; }
}
```

Our app needs the ability to calculate the area of a of Rectangles. Since we already learned the Single Responsibility Principle (SRP), we don't need to put the area calculation code inside the rectangle. So here I created another class for area calculator.

```
public class AreaCalculator {
  public double  CalArea(Rectangle r)
  {
    double area;
     area =r.Height * r.Width;
    }
    return area;
  }
}
```

Hey, we did it. We made our app without violating SRP. No issues for now. But can we extend our app so that it could calculate the area of not only Rectangle but also the area of Circle as well? Now we have an issue with the area calculation issue, because the way to do circle area calculation is different. Hmm. Not a big deal. We can change the CalArea method a bit, so that it can accept any type of object

```
public class Rectangle{
  public double Height {get;set;}
  public double Wight {get;set; }
}
public class Circle {     public double Radius {get;set;}
}
public class AreaCalculator
{
  public double TotalArea(object o)
  {
    double area = 0;      Rectangle objRectangle;      Circle objCircle;
      if(o is objRectangle)
      {
        objRectangle = (Rectangle)obj;
        area = obj.Height * obj.Width;
      }
      else
      {
        objCircle = (Circle)obj;
        area = objCircle.Radius * objCircle.Radius * Math.PI;
      }
    }
    return area;
  }
}
```

Wow. We are done with the change. Here we successfully introduced Circle into our app. We can add a Triangle and calculate it's area by adding one more "if" block in the TotalArea method of AreaCalculator. But every time we introduce a new shape we need to alter the TotalArea method. So the AreaCalculator class is not closed for modification. How can we make our design to avoid this situation? Generally we can do this by referring to abstractions for dependencies, such as interfaces or abstract classes, rather than using concrete classes. Such interfaces can be fixed once developed so the classes that depend upon them can rely upon unchanging abstractions. Functionality can be added by creating new classes that implement the interfaces. So let's refract our code using an interface.

```
interface Shape
{
  double Area();
}
  public class Rectangle: Shape
  {
    public double Height {get;set;}
    public double Width {get;set;}
    public  double Area()
    {
      return Height * Width;
    }
  }
  public class Circle: Shape
  {
    public double Radius {get;set;}
    public  double Area()
    {
      return Radius * Radus * Math.PI;
    }
  }
```

```
public class AreaCalculator
{
  public double  Area(Shape s)
  {
    double area=0;
     area =s.Area();
    }
    return area;
  }
}
```

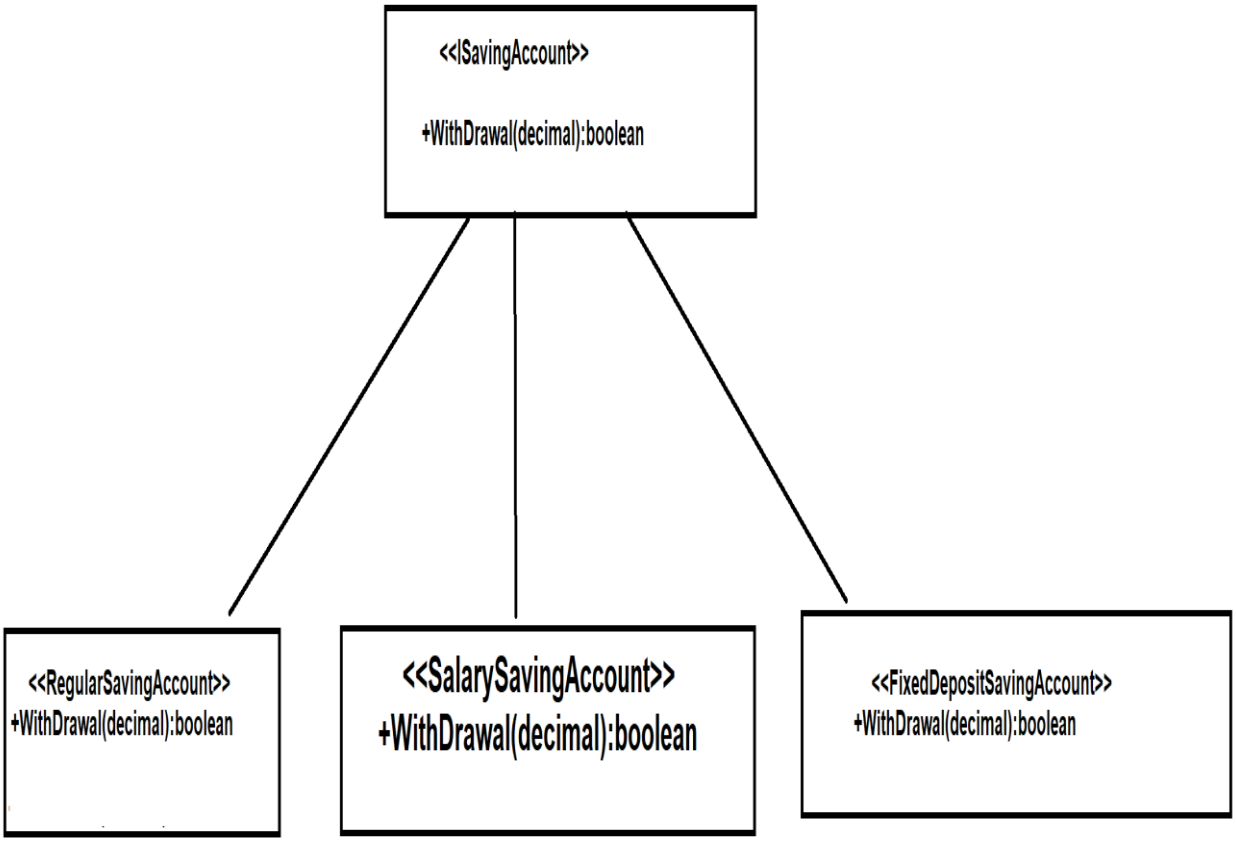
Now our code is following SRP and OCP both. Whenever you introduce a new shape by deriving from the "Shape" interface you need not change the "AreaCalculator" class. Awesome. Isn't it?

Liskov's Substitution Principle:-

Liskov Substitution Principle – is one of the SOLID principles defined by Barbara Liskov. Principle is based on the parent-child relationship in other words inheritance features of OOD (Object Oriented Design). Principle says "When class S is a subtype of class T then an object of type T can be replaced by an object of type S without affecting functionality/correctness of the implementation or program".

In simple words it says "Places in implementation

(Class/Function) that use a base class, in other words consume a service of a base class, must work correctly when the base class object is replaced by a child class (derived class) object."



Interface segregation principle (ISP)

This principle states that any client should not be forced to use an interface which is irrelevant to it.

```
public interface IEmployeeDatabase
{
    bool AddEmployeeDetails();
    bool ShowEmployeeDetails(int employeeId);
}
```

But now we are breaking something. We are forcing non-permanent employee class to show their details from database. So, the solution is to give this responsibility to another interface.

```
public interface IAddOperation
{
    bool AddEmployeeDetails();
}
public interface IGetOperation
{
    bool ShowEmployeeDetails(int employeeId);
}
```

And non-permanent employee will implement only IAddOperation and permanent employee will implement both the interface.

Dependency Inversion Principle:-

The definition of the principle, from Robert Martin's 1996 paper The Dependency Inversion Principle is this:

1. High-level modules should not depend upon low-level modules. Both should depend upon abstractions.
2. Abstractions should not depend upon details. Details should depend upon abstractions.

S A A T H Y A T E C H N O L O G I E S

Chapter-3

ASP.net vs ASP.net MVC

K A N N A B A B U

By.B.Kannababu

ASP.net Web Forms	ASP.net MVC
Page Life Cycle	No Page Life Cycle
ASP.net web forms has server controls	ASP.net MVC has HTML Helpers
Asp.Net Web Form supports view state for state management at client side	No 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)
No separation of concerns	Views and logic are kept separately.
Asp.Net supports Master pages	Asp.Net MVC supports Layouts
ASP.net supports user controls	Supports Partial Views
Tightly Coupled	Loosly Coupled
Partial classes	No partial classes
Parallel Development is not possible	No parallel development
Unit Testing is difficult	Unit Testing is easy
Performance is poor	Good
Less control over HTML	More control over HTML
Asp.Net Web Form follow a traditional event driven development mode	Asp.Net MVC is a lightweight and follow MVC (Model, View, Controller) pattern based development model

SATHYA TECHNOLOGIES

Chapter-4

Features of ASP.net MVC

KANNABABU

By.B.Kannababu

1.**Separation of Concerns**:-it is a process of dividing the application into Different modules.ASP.net does not support proper separation of concerns as UI code and Business Logic code both are available in single file
The separation the three components, allows the re-use of the business logic across applications
Maintainability and Unit testing is easy with this feature
2.**LooslyCoupling**:-means reduce the Degree of dependency.ASP.net is Tightly Coupled but MVC is Loosely Coupled Advantage of Loosely Coupling is Testing will become easy
3. **Parallel Development**:-Multiple programmers will involve in developing the application. Application Development is Easy
4. **Test Driven Development**: - MVC supports TDD approach in TDD Tester will provide guidelines to Developers Test Driven Development is the process where the developer creates the test case first and then fixes the actual implementation of the method
5. **Unit Testing**:-MVC is easy to perform UnitTesting.The Testing that was done by Developer is called as Unit Testing

6. **Clean Url**:- ASP.net supports File system, MVC does not support File system. MVC supports Clean URL because of clean url it is easy for SEO

7.**Performance is Good**:-In ASP.net whenever the page will gets executed then each and every time the entire page level events will fire .ASP.net supports Viewstate,Serversidecontrols,Events.Execution of the Page will take more time in MVC Performance is good

8. More Controllers on HTML

9. Added New Concepts

10. Easy to Learn and Develop

11. Support for Scaffolding Templates

12. Url Routing

SATHYA TECHNOLOGIES

Chapter-5

Versions of ASP.net MVC

KANNABABU

By.B.Kannababu

MVC Version	Visual Studio	Release date	Features
MVC 1.0	VS2008	13-Mar-2009	• MVC architecture withwebform engine • Routing • HTMLHelpers • AjaxHelpers • Auto binding
MVC 2.0	VS 2008,	10-Mar-2010	• Area • Asynchronouscontroller • Html helper methods with lambdaexpression • DataAnnotationsattributes • Client sidevalidation • Customtemplate • Scaffolding

MVC 3.0	VS 2010	13-Jan-2011	• Unobtrusive javascriptvalidation • Razor viewengine • Globalfilters • Remotevalidation • Dependency resolver forIoC • ViewBag
MVC 4.0	VS 2010 SP1, VS 2012	15-Aug-2012	• Mobile project template • Bundling andminification • Support for Windows AzureSDK
MVC 5.0	VS 2013	17-oct-2013	• Authenticationfilters • Bootstrapsupport • New scaffoldingitems • ASP.NetIdentity
MVC 5.2 - Current	VS 2013	28-Aug-2014	• Attribute basedrouting • bug fixes and minorfeatures upate

SATHYA TECHNOLOGIES

Chapter-6

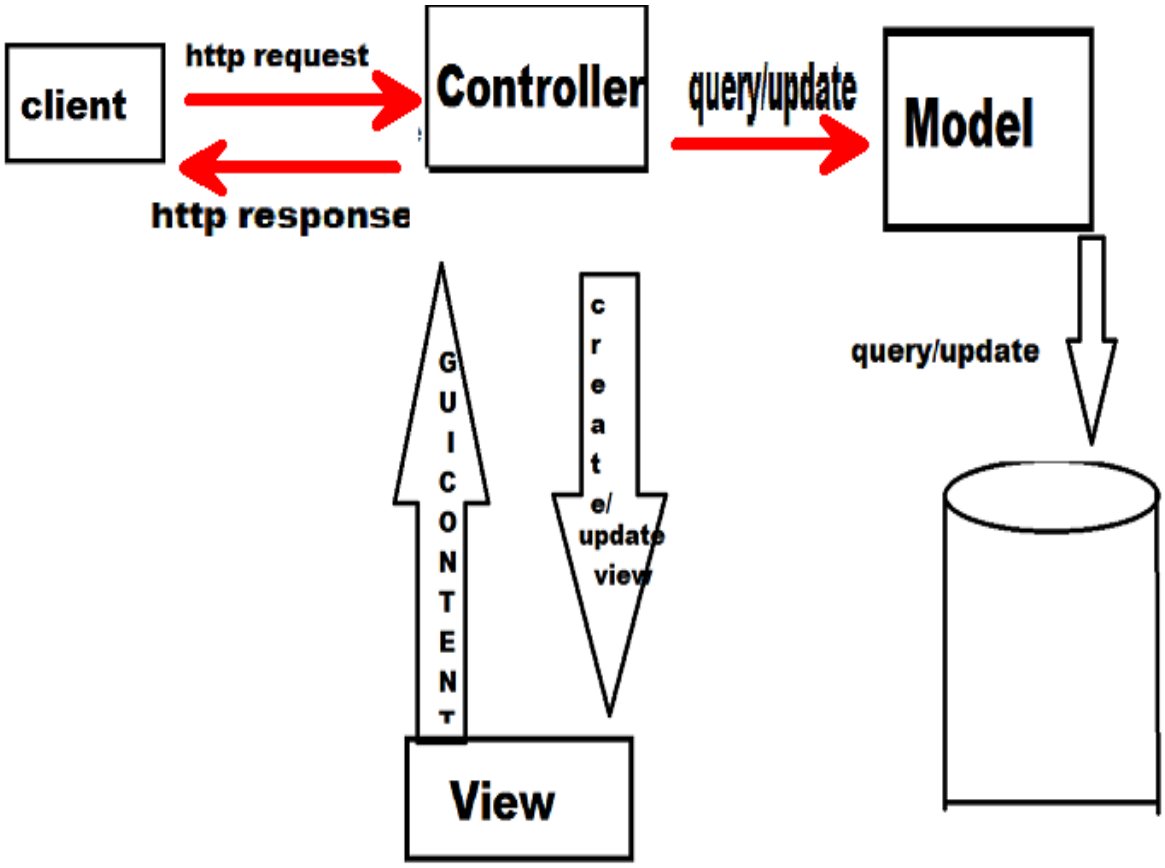
Architecture of ASP.net MVC

KANNABABU

By.B.Kannababu

By using MVC pattern, we can develop applications that are more flexible to changes without affecting the other components of our application.

- "**Model**", is for data.It contains all application validation/business logic that is not contained in view or controller.
- "**View**", contains HTML markup and view logic
- "**Controller**", contains control flow logic. It interacts with mvc model and views to control the flow of application execution.



What is a Model?

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

  A view can use model to display data in page.

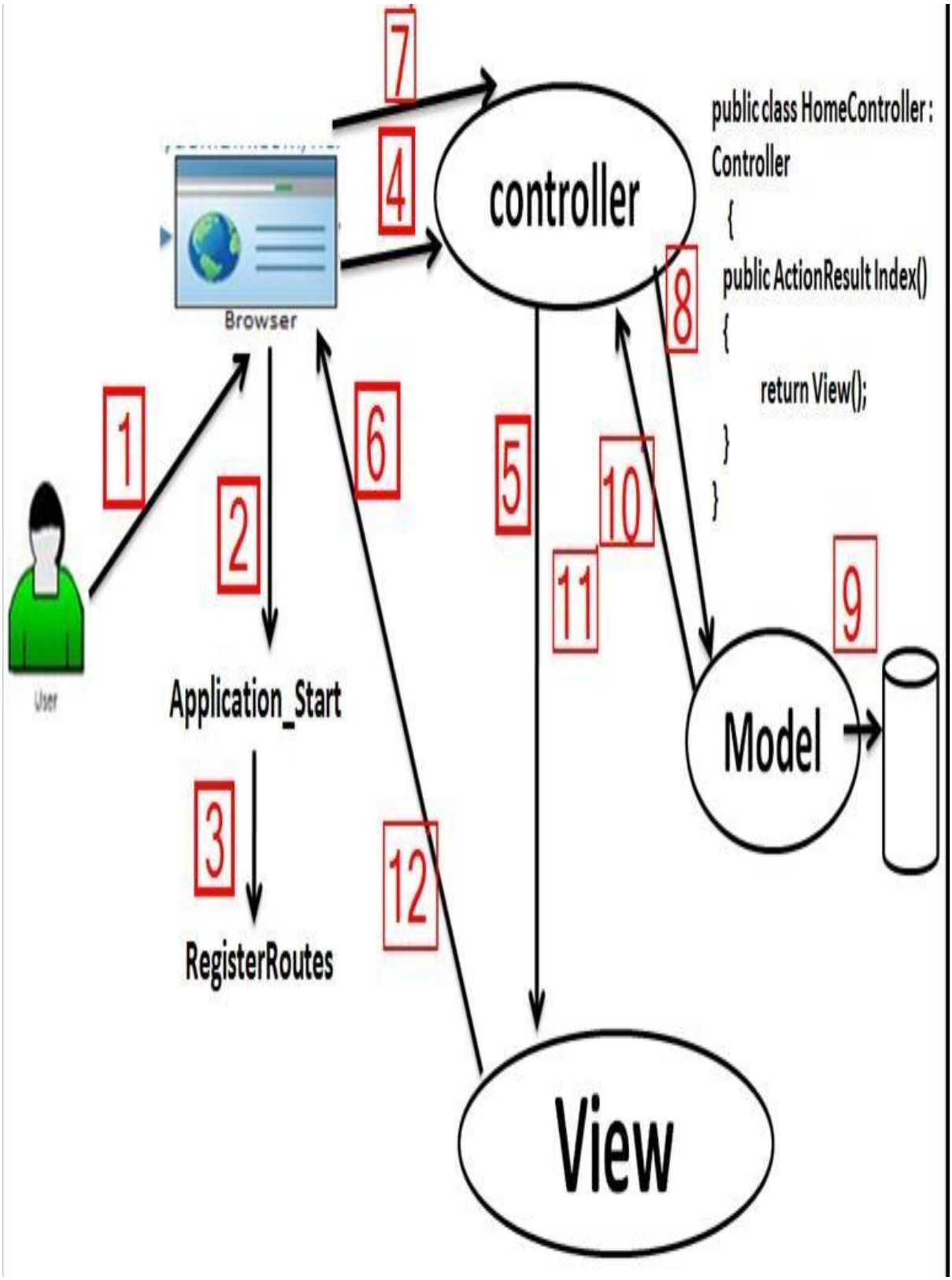
What is a View?

- View is an ASPX page without having a code behind file

- All page specific HTML generation and formatting can be done inside view
- One can use Inline code (server tags) to develop dynamic pages
- A request to view (ASPX page) can be made only from a controller's action method

What is a Controller?

- Controller is basically a C# or VB.NET class which inherits System.Mvc.Controller
- Controller is a heart of the entire MVC architecture
- Inside Controller's class action methods can be implemented which are responsible for responding to browser OR calling views.
- Controller can access and use model class to pass data to views
- Controller uses View Data to pass any data to view



1. Whenever client sends the request then IIS will accept the request
   http://servername:portno/controllername/Actionmethodname
2. http://localhost:1487/Home/Index
3. Whenever the request will go to the server for first time Application _Start will fire ,
   This event will fire only one time i.e. for first time when the first user is trying to access the application

3. Application_Start will invoke Register Routes. Register Routes will set the default Controller name and ActionmethodnameApplication_Start will invoke RegisterRoutes. RegisterRoutes will set the default Controllername and Actionmethodname

4. Request will go to Controller and Controller will invoke ActionMethod.Controller will check whether the request is Get or Post,as the Request is Get Request Get is invoked

```
Public class HomeController: Controller
{
   [HttpGet]
public ActionResult Index()
   {
return View();
    }}
```

5. Controller will invoke View

6. Index.cshtml page will execute and View will render Html output to Browser

7. Whenever user clicks on submit button in Browser, Controller will accept the request and it will check whether the request is Get or post and Controller will invoke Action Method of HttpPost

8. Controller will invoke Model

9. Model will interact with Database

10. Model will send result to Controller

11. Controller will invoke View

12. View will render Model object to Browser

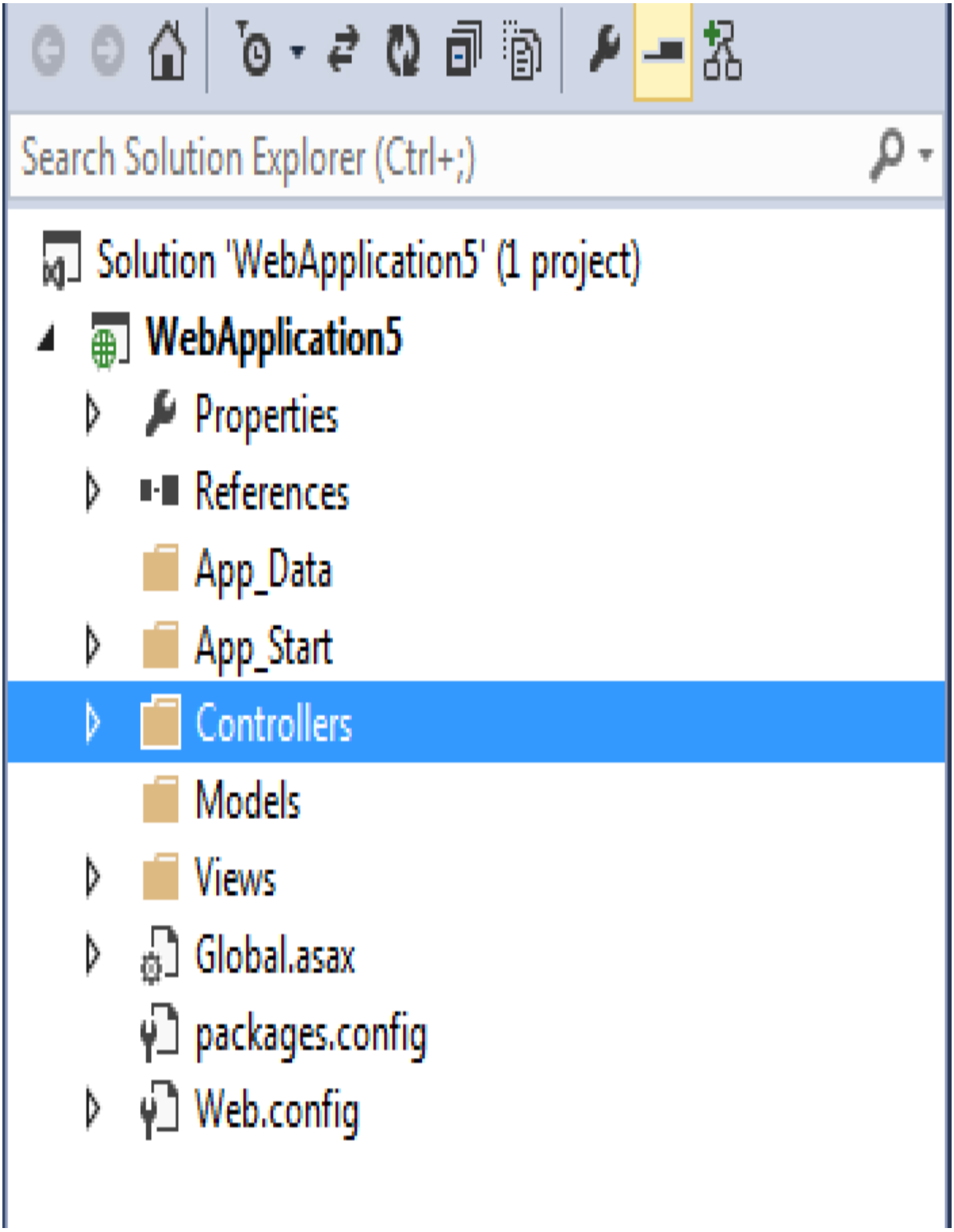
SATHYA TECHNOLOGIES

Chapter-7

FolderStructureof MVC

KANNABABU

By.B.Kannababu



App_Data:-App_Data stores application related storage files like DB files, XML files etc. We cannot access these files from any external browser for security reasons

2.**App_Start**:- App_Start folder can contain class files which will be executed when the application starts. Typically, these would be config files like

AuthConfig.csBundleConfig.csFilterConfig.csRouteConfig.cs etc.

MVC 5 includes BundleConfig.cs, FilterConfig.cs and RouteConfig.cs by default. We will see significance of these files later.

3. **Content**: - Content folder contains static files like css files, images and icons files. MVC 5 consits of Content folder which includes bootstrap.css, bootstrap.min.css and Site.css by default.

4. **Controllers**: - Controllers folder contains class files for the controllers.

Controllers handles users' request and returns a response.Controller name to end with "Controller". You will learn about the controller in the next section.

```
using System;
usingSystem.Web.Mvc;
namespace WebApplication5.Controllers
{
public class HomeController : Controller
{
[HttpGet]
public ActionResult Index()
{      return View();  } [HttpPost]
public ActionResult Index()
{      return View(); }
}
}
```

5. **Fonts**: Fonts folder contains custom font files for your application.

6. **Models**: Models folder contains model class files. Typically model class includes public properties, which will be used by application to hold and manipulate application data.

7. **Scripts**:-Scripts folder contains JavaScript or VBScript files for the application. MVC 5 includes javascript files for bootstrap, jquery 1.10 and modernizer by default.

8. **Views**: - Views folder contains html files for the application. Typically view file is a

.cshtml file where you write html and C# or VB.NET code.

Shared folder under View folder contains all the views which will be shared among different controllers e.g. layout files.

9.**Global.asax**:- Global.asax allows you to write code that runs in response to application level events, such as Application_BeginRequest, application_start, application_error, session_start, session_end etc.

10.**Packages.config**:-Packages.config file is managed by NuGet to keep track of what packages and versions you have installed in the application.

11.**Web.config:**Web.config file contains application level configurations.

Q) What is Bundling in ASp.net MVC?

in MVC Bundling means combining multiple files into a single file so that there will be having less http request to the server

This will definitely reduce the page loading time Mainly Bundling is done for css and Js files

Q) What is Minification?

Minification means removing the unwanted whitespaces, comments, line breaks from the code

Mainly Minification was done for CSS,JS files Minification will reduce the size of the file so that we can easily

accessthe file from server

Folder Structure of MVC

```
+App_Data
+App_Start
  +Routeconfig.cs
+Controllers
  +HomeController.cs
  +StudentController.cs
  +DeptController.cs
  +EmployeeController.cs
+Models
  +Student.cs
  +Dept.cs
  +Employee.cs
  +Login.c
+DAL
  +Student.cs
+Views
  +Home
    +index.cshtml
    +aboutus.cshtml
    +contactus.cshtml
    +Login.cshtml
  +Student
    +create.cshtml
    +delete.cshtml
    +update.cshtml
  +Dept

    +create.cshtml
    +delete.cshtml
    +update.cshtml
```

Controllers--->HomeController.cs

```
public class HomeController:Controller
{
 public ActionResult Index()
 {
  return View();
 }
 [HttpGet]
 public ActionResult Login()
 {
  return View();
 }
 [HttpPost]
 public ActionResult Login(Models.Login l)
 {
  return View();
 }
}
```

Models--->Login.cs

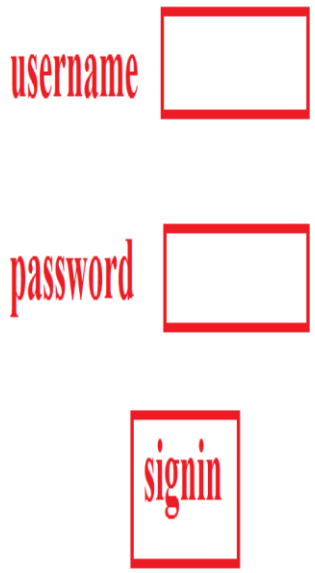
```
class Login
{
 public string Uname{set;get;}
 public string Password{set;get;}

}
```

login.cshtml

```
<html>
<head> </head>
<body>
EnterUserName
<input type="text">
<br>
EnterPassword
<input type="password">
<br>
<input type="button" value="login">
</body>
</html>
```

DAL--->Login.cs

```
public class Login
{
 public int CheckUser(Models.Login l)
 {
  goto database and check whether login user is valid or not
```



SATHYA TECHNOLOGIES

Chapter-8

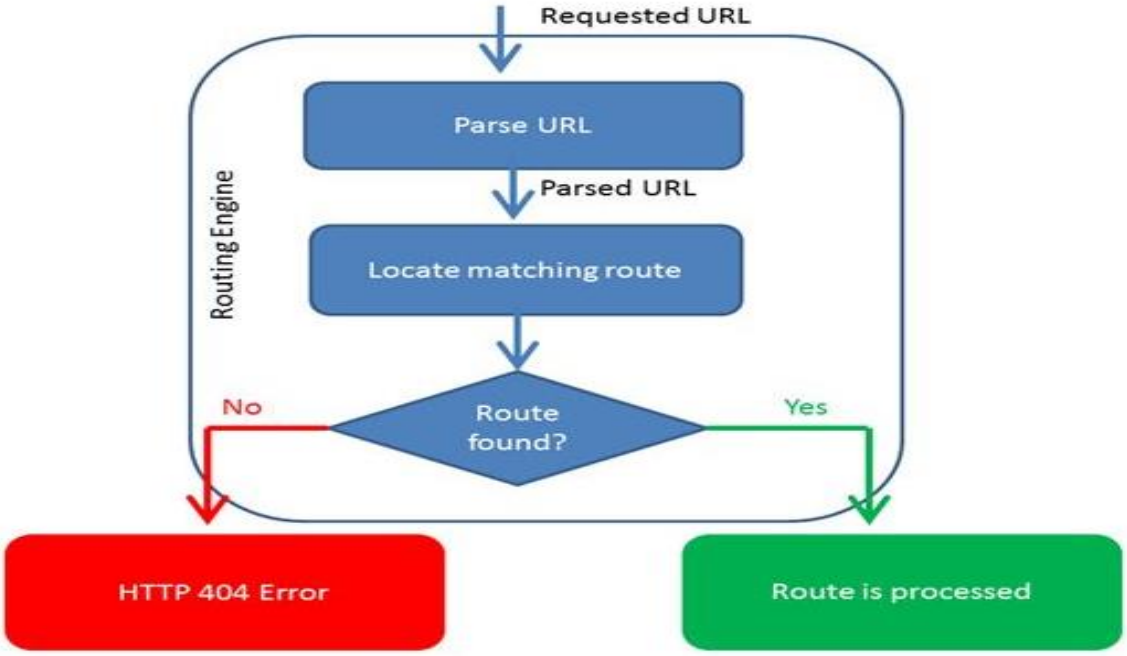
Routing in MVC

KANNABABU

By.B.Kannababu

Overview of ASP.NET MVC Routing

ASP.NET MVC routing is a pattern matching system that is responsible for mapping incoming browser requests to specified MVC controller actions. When the ASP.NET MVC application launches then the application registers one or more patterns with the framework's route table to tell the routing engine what to do with any requests that matches those patterns. When the routing engine receives a request at runtime, it matches that request's URL against the URL patterns registered with it and gives the response according to a pattern match. Let's see Figure



In Figure we can see how the routing engine processes a request and what response it sends. It gives a response according to URL match or not in the route table.

When the request's URL matches any of the registered route patterns in the route table then the routing engine forwards the request to the appropriate handler for that request. Thereafter the route is processed and gets a view on the UI.

When the request's URL does not match any of the registered route patterns then the routing engine indicates that it could not determine how to handle the request by returning a 404 HTTP status code.

Properties of Route:-

ASP.NET MVC routes are responsible for determining which controller method to execute for a given URL. A URL consists of the following properties:

Route Name: A route is a URL pattern that is mapped to a handler. A handler can be a controller in the MVC application that processes the request. A route name may be used as a specific reference to a given route.

URL Pattern: A URL pattern can contain literal values and variable placeholders (referred to as URL parameters). The literals and placeholders are located in segments of the URL that are delimited by the slash (/) character.

When a request is made, the URL is parsed into segments and placeholders, and the variable values are provided to the request handler. This process is similar to the way the data in query strings is parsed and passed to the request handler. In both cases variable information is included in the URL and passed to the handler in the form of key-value pairs. For query strings both the keys and the values are in the URL. For routes, the keys are the placeholder names defined in the URL pattern, and only the values are in the URL.

Defaults: When you define a route, you can assign a default value for a parameter. The defaults is an object that contains default route values.

Constraints: A set of constraints to apply against the URL pattern to more narrowly define the URL that it matches.

Understand the Default Route:-

The default ASP.NET MVC project templates add a generic route that uses the following URL convention to break the URL for a given request into three named segments.

url: "{controller}/{action}/{id}"

This route pattern is registered via call to the MapRoute() extension method of Route Collection.

```
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 }
    );
}
```

Ignore routes that end with axd extension

Route Name

URL Pattern

Default Values

When the MVC application launches the Application_Start() event handler of the global.asax execute that call the RegisterRoutes() method from RouteConfig class under App_Start directory (App_Start/RouteConfig.cs). The RegisterRoutes() route has a parameter that is a collection of routes called the RouteCollection that contains all the registered routes in the application. Figure 1.2 represents the default method that adds routes to the route table.\

ASP.NET MVC Routing with an Example:-

When the application starts up, ASP.NET MVC discovers all of the application's controllers by searching through the available assemblies for a class that implements the System.Web.Mvc. IController interface or derived from a class those implements this interface and whose class names end with the suffix Controller. When the routing framework uses this list to determine which controller it has access to, it chops off the Controller suffix from the entire controller class names.

URL	Controller	Action	Id
http://localhost:4736/	HomeController	Index	
http://localhost:4736/Book/	BookController	Index	

http://localhost:4736/Book/Create	BookController	Create	
http://localhost:4736/Book/Edit/2	BookController	Edit	2

Table 1.1 Request's URLs that match our default route pattern

The last request's URL (http://localhost:4736/Book/Edit/2) in table 1.1 is a perfect match to the registered default URL pattern because it satisfies every segment of the route pattern, but when we don't provide a complete request's URL then the routing engine automatically calls the controller and action method as per the default route pattern.

SATHYA TECHNOLOGIES

Chapter-9

Action Methods inMVC

KANNABABU

By.B.Kannababu

1. Observe the below code snippet

```
usingSystem;
classA
{
publicvoidShow()
   {   C.WL("i am show");   }
}
classB:A
{
publicvoidDisplay()
   {   C.WL("i am Display");   }
}
classC
{
staticvoidMain()
   {   }
}
```

Upcasting Required:-

Overriding Exists:-

Identify Object and Reference:-

Create Object:-

Invoke both the methods:-

2. Observe the below Code snippet

```
usingSystem;
classA
{
publicvirtualvoidShow()
   {C.WL ("i am A show");   }
}
classB:A
{
publicoverridevoidShow()
   {C.WL ("i am B Show");   }
publicvoidDisplay()
   {C.WL("i am Display");   }
}
classC
{
staticvoidMain()
   {
   }
}
```

Upcasting Required:-

Overriding Exists:-

Identify Object and Reference:-

Create Object:-

Invoke both the methods:-

3. Observe the below code snippet	4. Observe the below code snippet

usingSystem;
interfaceA
{voidShow();}
classB:A
{
publicvoidShow()
{ C.WL("i am B Show"); }
}
classC
{
staticvoidMain()
{
}
}
Upcasting Required:-
Overriding Exists:-
Identify Object and Reference:-
Create Object:-
Invoke Show method:-
	usingSystem;
	interfaceA
	{ voidShow();}
	classB:A
	{
	publicvoidShow()
	{C.WL ("i am B Show"); }
	}
	classC:A
	{
	publicvoidShow()
	{ C.WL ("i am C Show"); }
	}
	classD
	{
	staticvoidMain()
	{ }
	}
	Note:- Consider objects are creating dynamically
	Upcasting Required:-
	Overriding Exists:-
	Identify Object and Reference:-
	Create Object:-
	Invoke Show method:-

5. Observe code snippet

```
usingSystem;
classA
{
publicvirtualvoidShow()
   {  C.WLi am A show");   }
}
classB:A
{
publicoverridevoidShow()
   {  C.WL("i am B Show");  }
}
classC:A
{
publicoverridevoidShow()
   {   C.WL("i am C Show");  }
}
classD
{
staticvoidMain()
   {   }
}
```

Note:- Consider objects are creating dynamically

Upcasting Required:-

Overriding Exists:-

Identify Object and Reference:-

Create Object:-

Invoke Show method:-

6.

```
usingSystem;
abstractclassA
{ publicabstractvoidShow();}
classB:A
{
publicoverridevoidShow()
   {  C.WL("i am Show");   }
}
classC:A
{
publicoverridevoidShow()
   {  C.WL("i am C Show");  }
}
classD
{
staticvoidMain()
   {   }
}
```

Note:- Consider objects are creating dynamically

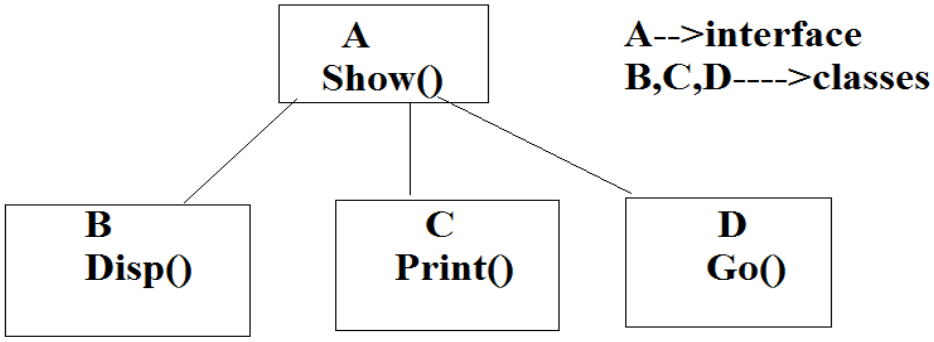
Upcasting Required:-

Overriding Exists:-

Identify Object and Reference:-

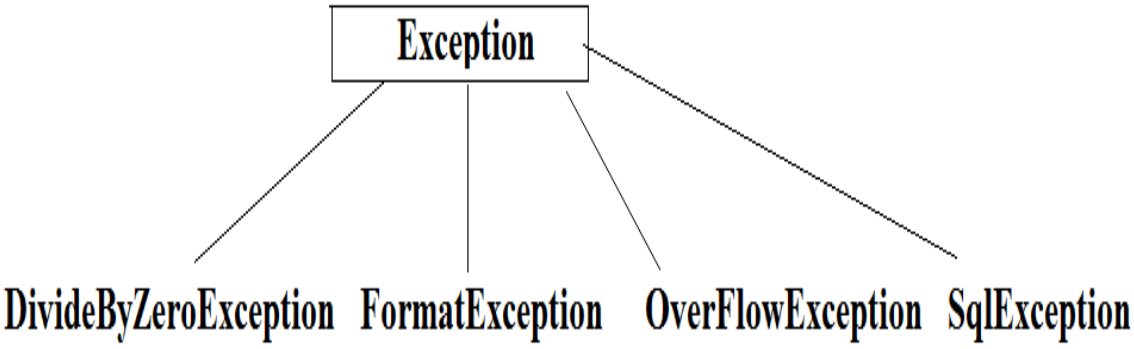
Create Object:-

Invoke Show method:-



hint:- invoke all the methods by applying upcasting

7.




```
try
{
}
catch(Exception e)
{
}
```

Consider Exception will occur at runtime within try block and control will jump to catch block

1. upcasting required

2. Overriding required

Explain the concept

8. class A
```
{
 public SqlConnectionCreateConnection()
{    return new SqlConnection();   }
  public A Show()
{
     return this;
}
   static void Main()
   {
    A a1=new A();
//invoke both the methods with a1 reference return the values and draw object diagram
  }
}
```

9. what is the output of below program?

```
usingSystem;
classA
{
staticvoidMain()
{C.WL(goFigure(60));
}
publicstaticintgoFigure(intx)
 {
if (x<100)
x=goFigure(x+10);
return (x-1);
 }
}
```

```
class ActionResult
{          }
class ViewResult
{
  public ViewResult View()
  {     return new ViewResult();  }
}
class PartialViewResult
{
  public PartialViewResult PartialView()
  {    return new PartialViewResult();  }
}
class ContentResult
{
  public ContentResult Content()
  {    return new PartialViewResult();  }
}
class FileResult
{
  public FileResult File()
  {    return new FileResult ();  }}
```

```
public class HomeController:Controller
{
  public ActionResult Index()
  {
      return View();
  }
  public ActionResult Aboutus()
  {
      return PartialView();
  }
  public ActionResult Contactus()
  {
      return Content();
  }
  public ActionResult GetData()
  {
      return File();
  }
}
```

Action Method: - in MVC whenever client sends the request Controller will accept the request and invoke Action Method.

Action Method consists of the Request Processing logic

- Action Methods can be accessed directly from Browser

syn to invoke Action Method from Browser: -

Controllername/Actionmethodname

Home/Index

Rules to declare Action method:-

- The method must be public.
- The method cannot be a static method.
- The method cannot be an extension method.
- The method cannot be a constructor, getter, or setter.
- The method cannot have open generic types.
- The method is not a method of the controller base class.
- The method cannot contain ref or out parameters

Note:-. ActionMethod will return the Derived class object of Action Result class

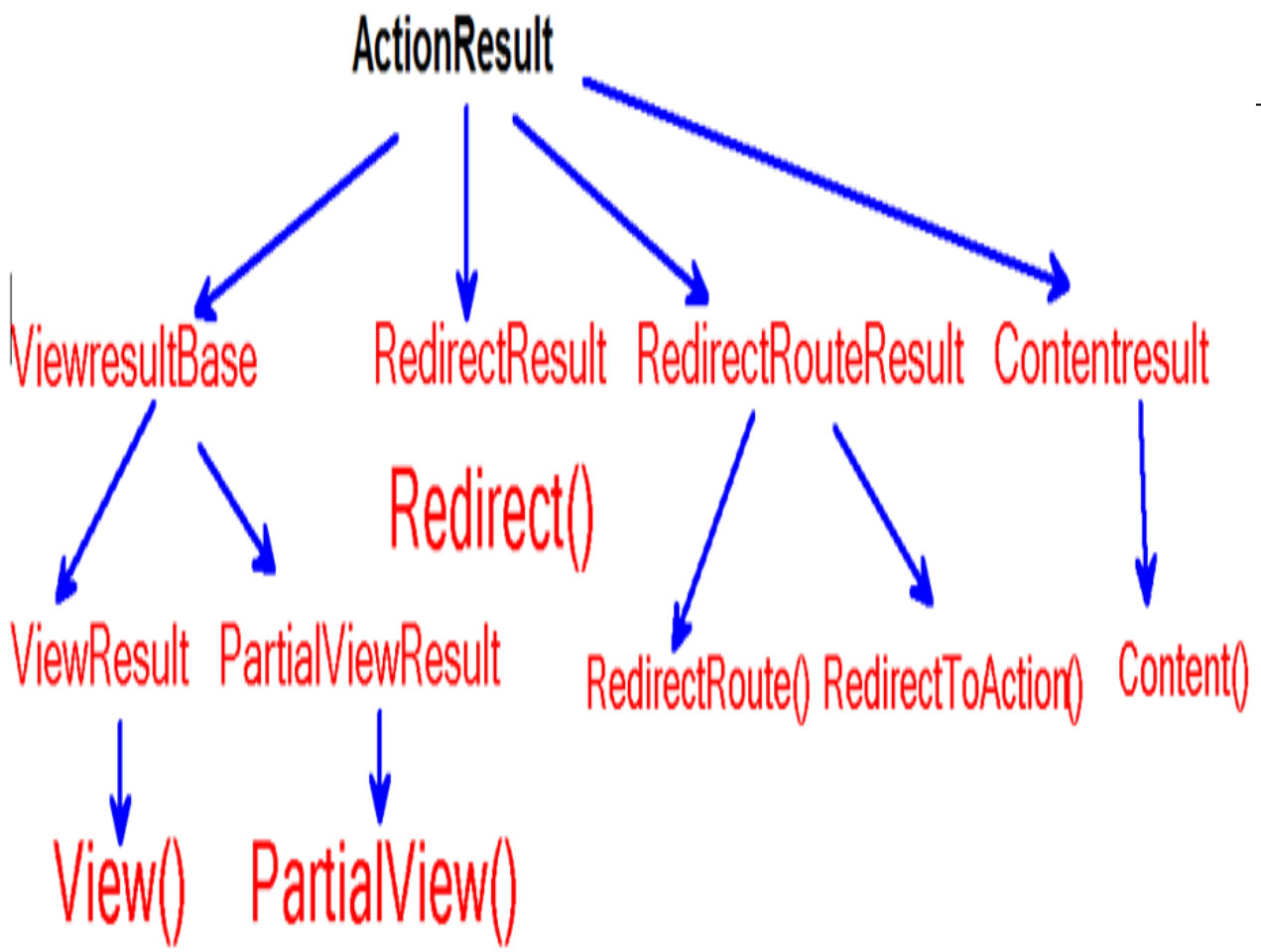
Action Methods are of 2 Types:-

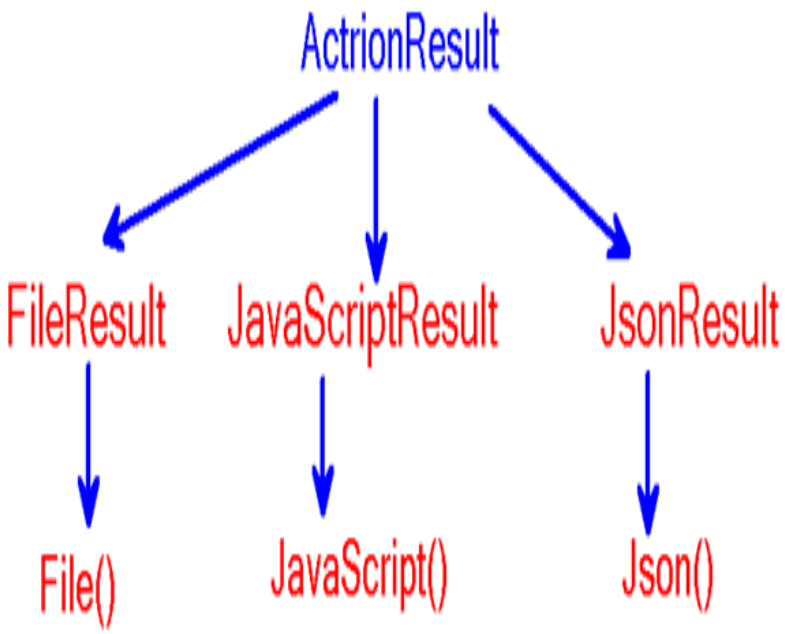
1. ActionMethod with View Result
2. ActionMethod without View Result
3. NonActionMethods

ActionMethod with View Result:-This method will return View Result Object most of the Action Methods will return View Result object and it will render the HTML output to the view This method will return View()

Ex:-

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





Result Class	Description	Base Controller method
ViewResult	Represents HTML and markup.	View()
EmptyResult	Represents No response.	
ContentResult	Represents string literal.	Content()
FileContentResult, FilePathResult, FileStreamResult	Represents the content of a file	File()
JavaScriptResult	Represent a JavaScript script.	JavaScript()
JsonResult	Represent JSON that can be used in AJAX	Json()
RedirectResult	Represents a redirection to a new URL	Redirect()
RedirectToRouteResult	Represent another action of same or other controller	RedirectToRoute()
PartialViewResult	Returns HTML	PartialView()
HttpUnauthorizedResult	Returns HTTP 403 status	

ActionResult: An ActionResult is a return type of a controller method, also called an action method ActionResult is the super class for all the Result classes

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

ViewResult:-Here Index is called Action method which returns the Index view. View result accepts the same view return types. It encapsulates the results of the Index method and that is used to render a specified view. View Result is a derived class of Action result class.

```
publicViewResult Index()
{
return View();
}
```

FileResult:-Here Index is called Action method which returns the Index view. File result accepts the binary File content return types. It encapsulates the binary file results of the Index method and that is used to send binary file content on view.

```
publicFileResult Index()
{
byte[] image =System.IO.File.ReadAllBytes(@"c:\pic.jpg");
return File(image, "applicaiton/jpeg");
}
```

JsonResult:-Here Index is called Action method which returns the Index view. JSON result accepts the JSON formatted content return types. It encapsulates the JSON results of the Index method and that is used to send JSON results on view.

```
publicJsonResult Index()
{
returnJson(true,JsonRequestBehavior.AllowGet);
}
```

RedirectResult:Here Index is called Action method which returns the Index view. Redirect result accepts the specified URL content return types. It encapsulates the URL content results of the Index method and that is used to send URL content results on view.

```
publicRedirectResult Index()
{     return Redirect("Login/Welcome");    }
```

JavaScriptResult:-Here Index is called Action method which returns the Index view. JavaScript result accepts the JavaScript content return types. It encapsulates the scripts content results of the Index method and that is used to send java script results on view.

```
publicJavaScriptResult Index(Employee employee)
{
return JavaScript("<script>alert('Hello!')</script>");
}
```

RedirectToRouteResult:-Here Index is called Action method which returns the Index view. RedirectToRoute accepts the route value dictionary. It encapsulates the route value dictionary results of the Index method and that is used to send route value results on view.

```
publicRedirectToRouteResult Index()
{
returnRedirectToRoute("Hello!“);
}
```

HttpStatusCodeResult:- Here Index is called Action method which returns the Index view. HttpStatusCode result accepts the specific HTTP response status. It encapsulates the HttpStatus code results of the Index

method and that is used to send Http Status results on view.

```
publicHttpStatusCodeResultIndex()
{
return new
HttpStatusCodeResult(HttpStatusCode.BadRequest, "Bad
Request");
}
```

ContentResult:-Here Index is called Action method which returns the Index view. Content result accepts the user defined content type. It encapsulates the content type results of the Index method and that is used to send content type results on view.

```
publicContentResult Index()
{ return Content("Hello world","html/plain",Encoding.UTF8);
}
```

Empty Result:-Here Index is called Action method which returns the Index view. Empty result accepts nothing in view. It encapsulates the nothing results in Index method and that is used to send empty results.

```
public ActionResult Index()
{   return new EmptyResult();   }
```

SATHYA TECHNOLOGIES

Chapter-10

Views in MVC

KANNABABU

By.B.Kannababu

View is a user interface. View displays data from the model to the user and also enables them to modify the data. ASP.NET MVC views are stored in **Views** folder. Different action methods of a single controller class can render different views, so the Views folder contains a separate folder for each controller with the same name as controller, in order to accommodate multiple views.

Shared folder contains views, layouts or partial views which will be shared among multiple views.

Razor View Engine

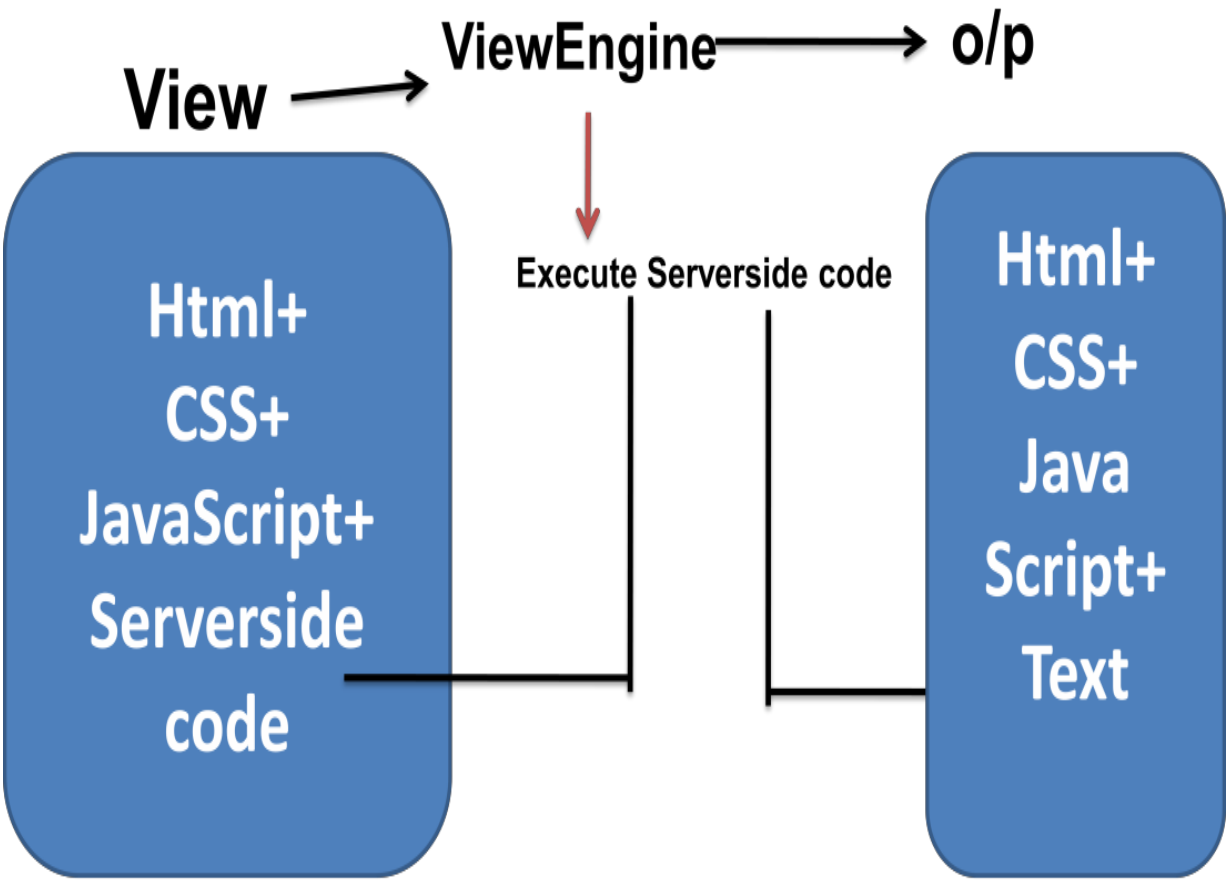
Microsoft introduced the Razor view engine and packaged with MVC 3. You can write a mix of html tags and server side code in razor view. Razor uses @ character for server side code instead of traditional <% %>. You can use C# or Visual Basic syntax to write server side code inside razor view. Razor view engine maximize the speed of writing code by minimizing the number of characters and keystrokes required when writing a view. Razor views files have .cshtml or vbhtml extension.

ASP.NET MVC supports following types of view files:

View file extension	Description
.cshtml	C# Razor view. Supports C# with html tags.
.vbhtml	Visual Basic Razor view. Supports Visual Basic with html tags.
.aspx	ASP.Net web form
.ascx	ASP.NET web control

Razor View EnginevsWebformViewEngine:-

Razor View Engine	WebformViewEngine
Available in MVC3.0	Available in MVC1.0
is advanced viewengine that was introduced	it is the default viewengine which following asp syntax
System.Web.Razor	System.Web.Mvc. WebformView Engine
Extension .cshtml	.aspx
razor engine doesnot have toolbox	we can use toolbox
@{ }	<% %>
Razor Engine Support TDD	Webform Engine doesnot support TDD
Razor support Easy Syntax	Webforms support complex syntax



ViewEngine is a component of ASP.net MVC which is responsible to convert Serverside code to Html code

Razor Syntax:-Single statement block and inline expression

```
<div>
    @{ var message = "Hello, Razor view engine"; }
    Message is : <b> @message </b>
</div>
```

Multi statement block:-

Conditional Statements:-

```
<div>
    @{
        var isValid = true;
        if(isValid)
        {
            <p>It is an if statment in code block</p>
        }
        else
        {
            <p>It is an else statment in code block</p>
        }
    }

   @if (true)
   {
       <p>It is a single if block</p>
   }
</div>
```

Looping:-

```
<div>
@{
for(int i=0;i<=10;i++)
{<h1>@i</h1>}
}
</div>
@{
string s=DateTime.Now.ToString();
}
<h1>@s</h1>
```

SATHYATECHNOLOGIES

KANNABABU

Chapter-11

Html Helpers inMVC

By.B.Kannababu

An HTML Helper is just a method that returns a HTML string. The string can represent any type of content that you want. For example, you can use HTML Helpers to render standard HTML tags like HTML <input>, <button> and <img> tags etc

1. HTML helpers in MVC are similarlike ASP.NET Web Form controls

2.HtmlHelpers are LeightWeight

3. No Viewstate is required

4. Can Easily convert HtmlHelper into Html code

5. HtmlHelpers can be considered as clientside controls to provide interaction with View

6. HtmlHelpers are Extension methods that generate HTML code

7. in MVC we will Design Views by using HtmlHelpers
Different Types of HtmlHelpers

- Inline HtmlHelper
- Build InHtmlHelper

– Standard HtmlHelper

– Strongly typed HtmlHelper

– Template HtmlHelper

- CustomHtmlHelper

Standard HtmlHelper:-

- HTML Form Elements
- There following HTML helpers can be used to render (modify and output) HTML form elements:

• BeginForm()
• EndForm()
• TextArea()
• TextBox()
• CheckBox()
• RadioButton()
• ListBox()
• DropDownList()
• Hidden()
• Password()
	@using (Html.BeginForm())
	{
	@Html.Label("username"
	) @Html.TextBox("uid")
	
	@Html.Label("pwd")
	@Html.TextBox("pwd")
	
	@Html.Label("ConfirmPasswor
	d") @Html.TextBox("cpwd")
	
	@Html.Label("Select Course")
	@Html.CheckBox("c1",false)
	@Html.CheckBox("c2",false)
	@Html.CheckBox("c3",false)
	
	}

Strongly Typed HTML Helpers:-

•The HTML elements are created based on model properties.

•The strongly typed HTML helpers work on lambda expression.

•The model object is passed as a value to lambda expression, and you can select the field or property from model object and use the model values as ids or values fro Html Elements.

syn:-

```
@Html.ElementnameFor(i/p=>modelvalue)

@Html.TextBoxFor(m=>m.Name)

@Html.TextArea(m=>m.Address , 5, 15, new{}))

@Html.PasswordFor(m=>m.Password)

@Html.CheckBoxFor(m=>m.IsApproved)
```

```
@Html.RadioButtonFor(m=>m.IsApproved, "val")
```

•Strongly typed Views will automatically Generate by Scafolding Templates based on Model class

Example:-

code for Homecontroller.cs:-

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

code for Index.cshtml:-

```
<h2>Index</h2>
@using (Html.BeginForm("MyIndex", "Home", FormMethod.Get))
{
<table>
<tr>
<td> @Html.Label("lbluname","EnterUsername") </td>
<td> @Html.TextBox("t1")  </td>
</tr>
<tr>
<td>@Html.Label("lbjpwd","Enter Password")</td>
<td>@Html.TextBox("t2") </td>
```

```
</tr>
<tr>
<td>Enter Address</td>
<td> @Html.TextArea("t3",null,5,10,null) </td>
</tr>
<tr>
<td></td>
<td><input type="submit" />        </td>
</tr></table>
}
```

DIFFERENCE BETWEEN HTML.LABEL() AND HTML.LABELFOR()

Html.Label()	Html.LabelFor()
It is loosely typed. It may be or not bounded with Model Properties.	It is strongly typed. Means, It will be always bounded with a model properties.
It requires property name as string.	It requires property name as lambda expression.
It doesn't give you compile time error if you have passed incorrect string as parameter that does not belong to model properties.	It checks controls at compile time and if any error found it raises error.
It throws run time error. Run time error gives bad impression to user and if the project is worthy, you may lose your client simply because of one error.	It throws compile time error which can be corrected before launching the project. It enhances user experience without throwing error.

Difference between TextBox and TextBoxFor:

- @Html.TextBox() is loosely typed method whereas @Html.TextBoxFor() is a strongly typed (generic) extension method.
- TextBox() requires property name as string parameter where asTextBoxFor() requires lambda expression as a parameter.
- TextBox doesn't give you compile time error if you have specified wrong property name.

It will throw run time exception.

- TextBoxFor is generic method so it will give you compile time error if you have specified wrong property name or property name changes. (Provided view is not compile at run time.)

Extension methods:-

it is a concept of adding new methods to existing class without

applying inheritance

while working with extesnionmethods no need to inherit the original

class and no need to modify the original class

Rules :-

1. Extesnion methods should be declared in static class

2. method parameter should be class name with this keyword

staticreturntypemethodname (this classnamearg)

{

}

note:-when we compile the program the compiler will add the

extesnion methods to the existing class

3. Extension method can be called by using objectname

generally we will use extsnion methods in linq queries

```
using System;
namespace ConsoleApplication8
{
class A
 {
public void Show()
 {
Console.WriteLine("i am show");
 }
public void Display()
 {
Console.WriteLine("i am display");
 }
public void Print()
 {
Console.WriteLine("i am print");
 }
 }
static class XXX
 {
public static void Newmethod(this A obj)
 {
Console.WriteLine ("i am newmethod");
 }
 }
```

==========

```
class Program
 {
static void Main()
 {
 A a1 = new A();
a1.Show();
a1.Display();
a1.Print();
a1.Newmethod();
Console.ReadLine();
 }
```

SATHYA TECHNOLOGIES

KANNABABU

Chapter-12

Form Collection in MVC

By.B.Kannababu

A **FormCollection** Object is used to retrieve form input values in action method.

FormCollection object makes programmers job easier and forms data can be easily accessed in action method.

FormCollection Example - Retrieving Form Data

Step 1: Create a New ASP.NET MVC Project or open Existing Project.
Step 2: Create a new Model **StudentModel.cs** and add the following code in it.

```
1. namespaceMvcForms.Models
2. {
3. publicclassStudentModel
4. {
5. publicintId{get;set;}
6. publicstringName{get;set;}
7. publicboolAddon{get;set;}
8. }
9. }
```

Step 3: Go to **Views Home Index.cshtml** and add the following code to create form.

```
<h3><b>Forms - Form Collection Objects</b></h3>
```

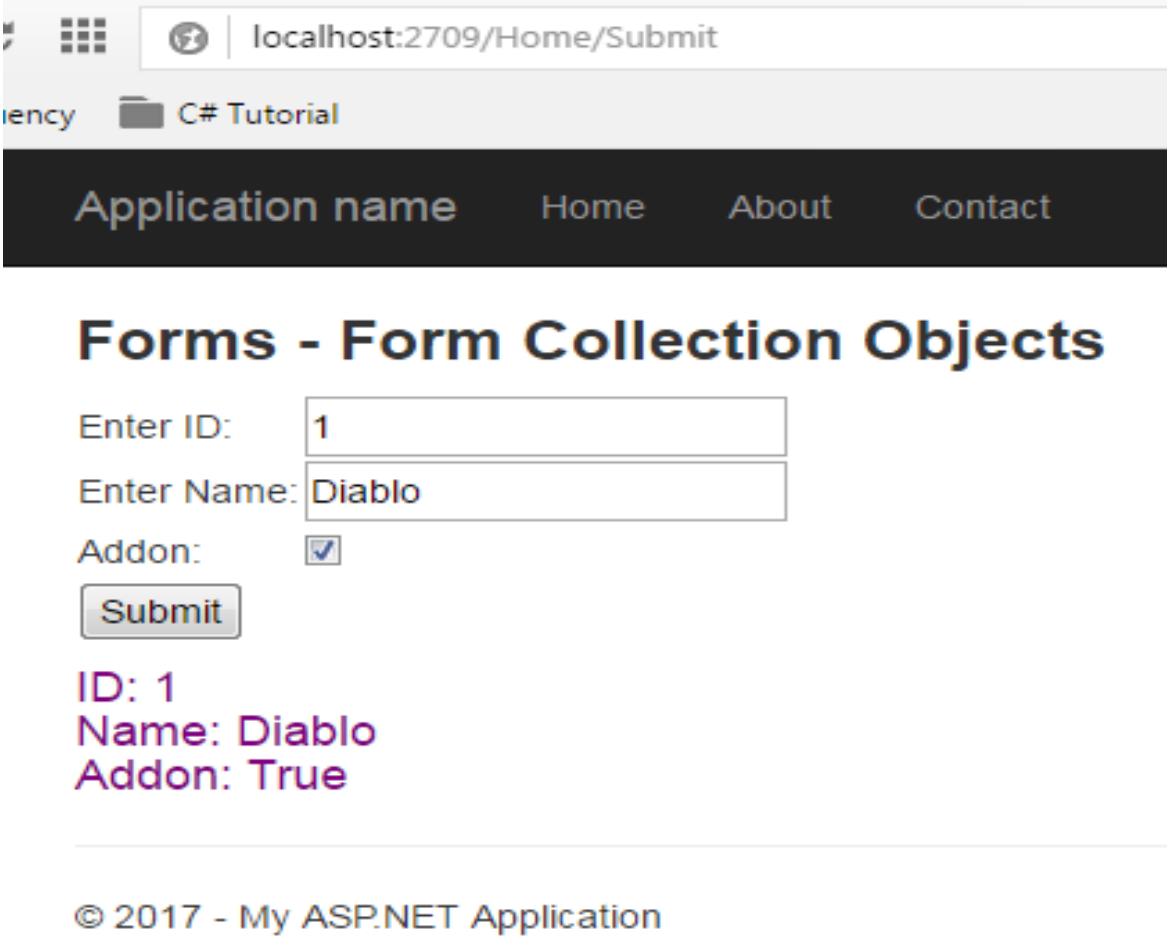
```
1. @using (Html.BeginForm("Submit", "Home", FormMethod.Post))
2. {
3. <table>
4. <tr>
5. <td>Enter ID: </td>
6. <td>@Html.TextBox("Id")</td>
7. </tr>
8. <tr>
9. <td>Enter Name: </td>
```

```
<td>@Html.TextBox("Name")</td>
</tr>
<tr>
<td>Addon: </td>
<td>@Html.CheckBox("Addon", false)</td>
</tr>
<tr>
<tdcolspan="2"><inputtype="submit"value="Submit"></td>
</tr>
</table>
}
<h4style="color:purple">
ID: @ViewBag.Id<br/>
Name: @ViewBag.Name<br/>
Addon: @ViewBag.Addon
</h4>
```

Step 4: Create following action method in **HomeController**

```
[HttpPost]
publicActionResultSubmit(FormCollection fc)
{
ViewBag.Id= fc["Id"];
ViewBag.Name= fc["Name"];
boolchk=Convert.ToBoolean(fc["Addon"].Split(',')[0]);
ViewBag.Addon=chk;

returnView("Index");
}
```

Step 5: Run your Project.



HTTPGET and HTTPPOST Method:-

HttpGet and HttpPost, both are the method of posting client data or form data to the server. HTTP is a HyperText Transfer Protocol that is designed to send and receive data between client and server using web pages. HTTP has two methods that are used for posting data from web pages to the server. These two Methods are HttpGet and HttpPost.

HTTPGET

HttpGet method sends data using a query string. The data is attached to URL and it is visible to all the users. However, it is not secure but it is fast and quick. It is mostly used when you are not posting any sensitive data to the server like username, password, credit card info etc.

HTTPPost method hides information from URL and does not bind data to URL. It is more secure than HttpGet method but it is slower than HttpGet. It is only useful when you are passing sensitive information to the server.

SATHYATECHNOLOGIES

Chapter-13

StateManagement Techniques in ASP.net MVC

(ViewData, ViewBag, TempData)

KANNABABU

By.B.Kannababu

ASP.net MVC provides Different Techniques to pass data from controller to View or Action to Action or across actions

- ViewData
- ViewBag
- TempData
- Session
- Application

View Data is used to send data from Controller Action method to View

- ViewData will maintain the data in object format
- ViewData is a dictionary object that is derived from ViewDataDictionary class
- publicViewDataDictionary ViewData { get; set; }
- ViewData is a property of ControllerBase class.
- It's required typecasting for getting data

Syn to store the value in ViewData:-

ViewData["varname"]=value;

Syn to read the value in ViewData:-

string s=ViewData["varname"].ToString();

Ex:-1

```
public class HomeController : Controller
{
public ActionResult Index()
{
    ViewData ["x"] = "SathyaTechnologies"; return View();
}
}
```

Index.cshtml:-

```
<h2>Index</h2>
@{
string s = ViewData["x"].ToString();
}
<h1>@s</h1>
```

Ex:-3

```
public  ActionResult Index()
{
List<string> Student = new List<string>();
Student.Add("Jignesh");
Student.Add("Tejas");
Student.Add("Rakesh");
ViewData["Student"] = Student;
return View();
}
```

Index.cshtml:-

```
<ul>
@foreach (var student in ViewData["Student"] as
List<string>)
{<li>@ student</li>}
```

ViewBag:-

•ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.

- ViewBag is a property of ControllerBase class.
- public dynamic ViewBag { get; }
- It's life also lies only during the current request.
- If redirection occurs then it's value becomes null.
- It doesn't required typecasting for getting data. public abstract class ControllerBase

```
{
publicViewDataDictionary ViewData { get; set; }
public dynamic ViewBag { get; }
publicTempDataDictionary TempData { get; set; }
}
```

syn to store the value in ViewBag:-

ViewBag.Propertyname=value;

syn to retrieve the value from ViewBag:-

ViewBag.propertyname;

Ex:-

```
public class HomeController : Controller
{
public ActionResult Index()
{
ViewBag. Name="SathyaTechnologies";
return View();
}
}
```

Code for Index.cshtml:-

```
<h2>Index</h2>
@{string s = ViewBag.Name;}
<h1>@s</h1>
```

Note:-Once client request is completed ViewBag and ViewData will be destroyed

Ex:-

Ex to pass the data from Controller to View:-

```
public class HomeController : Controller
{
public ActionResult Index()
{
```

```
ViewData ["x"] = "SathyaTechnologies";
return View();}}
```

Index.cshtml:-

```
@{
string s =(string)ViewData["x"];
}
<h1>@s</h1>
```

Ex to pass Array using ViewData:-

```
public class HomeController : Controller
{
public ActionResult Index()
{
string[] s = new string[4] {"anil","sunil","ajay", "rahul" };
ViewData ["s"] = s; return View();
}}
```

Index.cshtml:-

```
@{
string[] s=(string[])ViewData["s"];
for (int i = 0; i<=s.Length-1; i++)
{
<h1>@s[i]</h1>
}}
```

Ex:-

```
public class HomeController: Controller
{
public ActionResult Index()
{ViewData.Add("Id",1);
ViewData.Add(new KeyValuePair<string,object>
        ("Name", "Bill"));
ViewData.Add("Age", 20);
return View();
}      }
```

Index.cshtml:-

```
@{
string s = ViewData["id"].ToString(); string s1 =
ViewData["Name"].ToString(); string s2 =
ViewData["Age"].ToString();
string s3 = s1 + " Rollno is " + s + " Age is" + s2;
}<h1>@s3</h1>
```

TempData:-

- It is used to send data from one Actionmethod to another Action method
- We can store any type of value in tempdata
- TempData will store data in object format

- It requires typecasting
- TempData is a dictionary object that is derived from TempDataDictionary class
- publicTempDataDictionary TempData { get; set; }
- TempData is a property of ControllerBase class.
- It's life is very short and lies only till the target view is fully loaded.
- It is used to store only one time messages like error messages, validation messages.

Persisting Data with TempData:-

•TempData is used to pass data from current request to subsequent request (means redirecting from one page to another). It's life is very short and lies only till the target view is fully loaded. But you can persist data in TempData by calling Keep() method.

•TempData is used to maintain the data for a single request if we want TempData to maintain data for next request also we have to use Keep and Peek method

•void Keep()

This method you can use when all items in TempData you want to retain and does not allow deletion for any TempData's items.

Example: @TempData["key"];

TempData.Keep();

•This method you can use when particular TempData's value you want to persist and does not allow deletion for that particular TempData's value. Rest of the TempData's values will be deleted.

•Example : @TempData["key "];

TempData.Keep("key ");

```
public class HomeController:Controller
{
publicActionResult Index()
{
TempData["x"] = "ST";
returnRedirectToAction("MyIndex");
}
publicActionResultMyIndex()
{
string s = TempData["x"].ToString();
TempData.Keep();
return View();}}
```

Peek and Read:

• If you set a TempData in your action method and if you read it in your view by calling "Peek" method then TempData will be persisted and will be available in next request.

```
string message=TempData.Peek("Message").ToString();
public class HomeController:Controller
{
public ActionResult Index()
{
TempData["x"] = "ST";
return RedirectToAction("MyIndex");
}
```

```
        public ActionResult MyIndex()
        {
        stringstr = TempData.Peek("x").ToString();
        return View();
}
}
```

Ex:-2 TempData with NormalRead:-

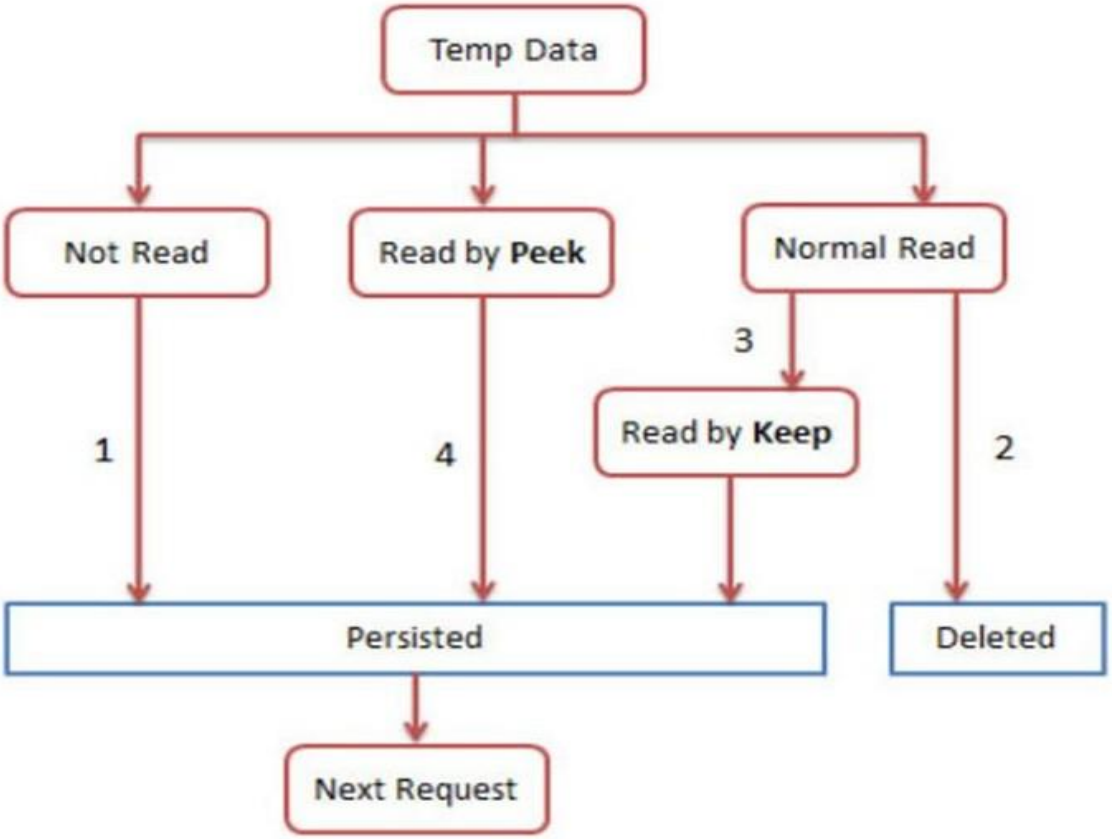
```
public ActionResult Index()
{
    TempData["x"] = "sathyatechnologies";
    return RedirectToAction("MyIndex","Home");
}
public ActionResult MyIndex()
{
    string s = TempData["x"].ToString();
    return View();
}
```

Not Persisted

Ex:-3 TempData with Keep():-

```
public ActionResult MyIndex()
{
    string s = TempData["x"].ToString();
    TempData.Keep();
    return View();
}
```

persisted



1. TempData with NotRead value is Persisted for Next Request
2. TempData with Normal Read value is Deleted i.evalue is not availablefor Next Request
3. TempData with Normal Read with Keep value is i.e Persisted for Next Request
4. TempData Read by Peek value is persisted for Next Request

View()	RedirectToAction()	Redirect()
1. it is used to navigate from one Actionmethod to another Action method	1. possible	1. possible
2. will hide the destination url address	2. will not hide	2. will not hide
3. syn:- View("Actioname");	3. syn:- RedirectToAcftion("actionname", "controllername");	3. syn:- Redirect(" complete url");
4. navigation within the same webserver	4. same webserver	4. different webserver and same webserver

SATHYA TECHNOLOGIES

Chapter-14

Model DataBinding

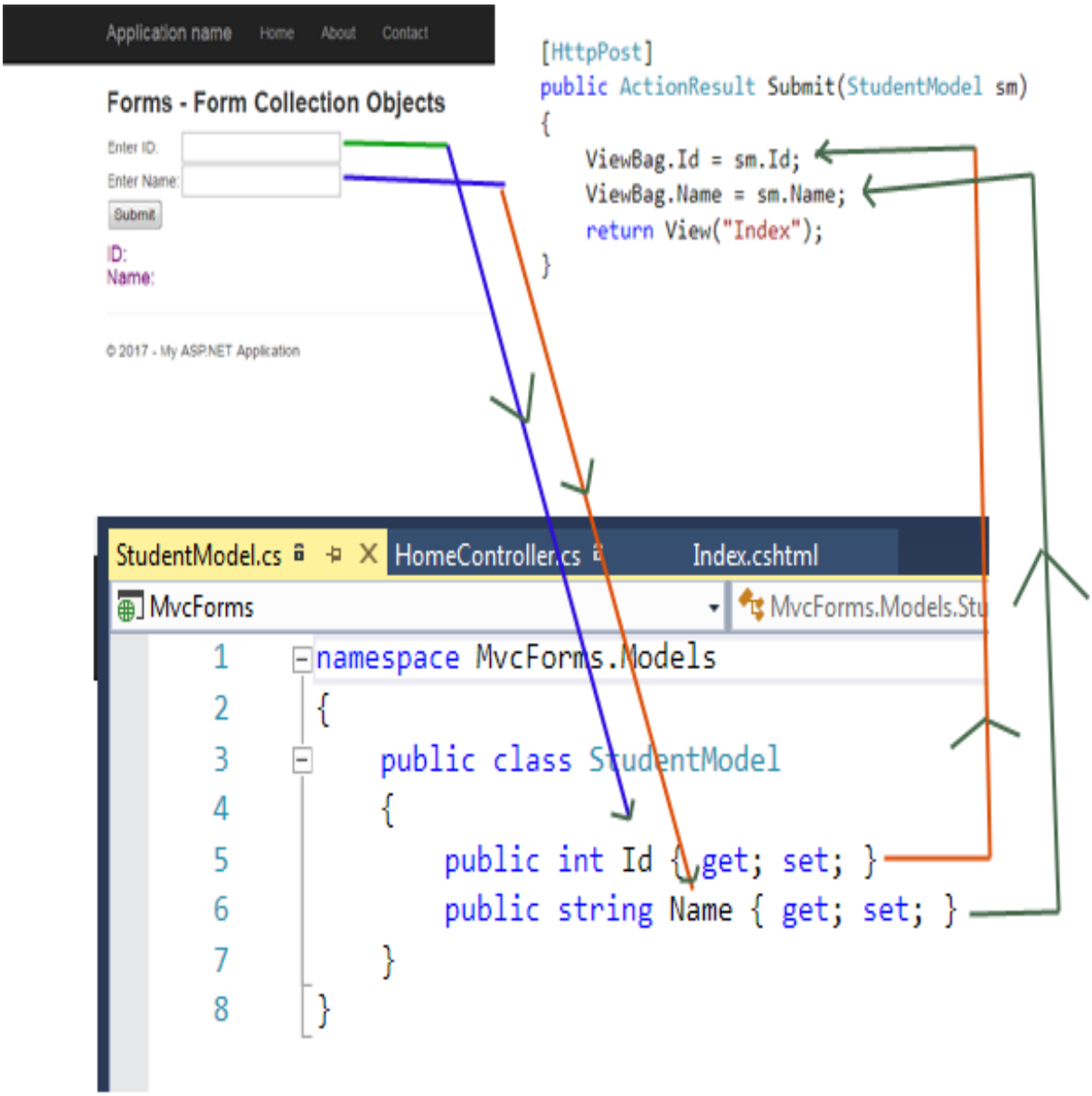
KANNABABU

By.B.Kannababu

QWHAT IS MODEL BINDING IN MVC?

Model binding means bind your input control to the respective model property

Do you ever wondered that how the form data gets automatically bound with model properties? Let's see the picture below:



You have noticed the following thing:

1. It is much cleaner code and there is no fancy coding.

2. No type casting needed.

3. No need to remember model properties because it appears in IntelliSense.

4. Adding and deleting properties from model is easy.

several model binding techniques:-

1. No Binding
2. Simple Binding
3. Class Binding
4. Complex Binding
5. FormCollection Binding
6. Bind Attribute

NO BINDING:-

No Binding means access form data directly without binding form to a specific model.

Example:

Model: StudentModel.cs

```
namespaceMvcForms.Models
{
public class StudentModel
    {
publicint Id { get; set; }
public string Name { get; set; }
    }
```

```
}
```

View: Index.cshtml:-

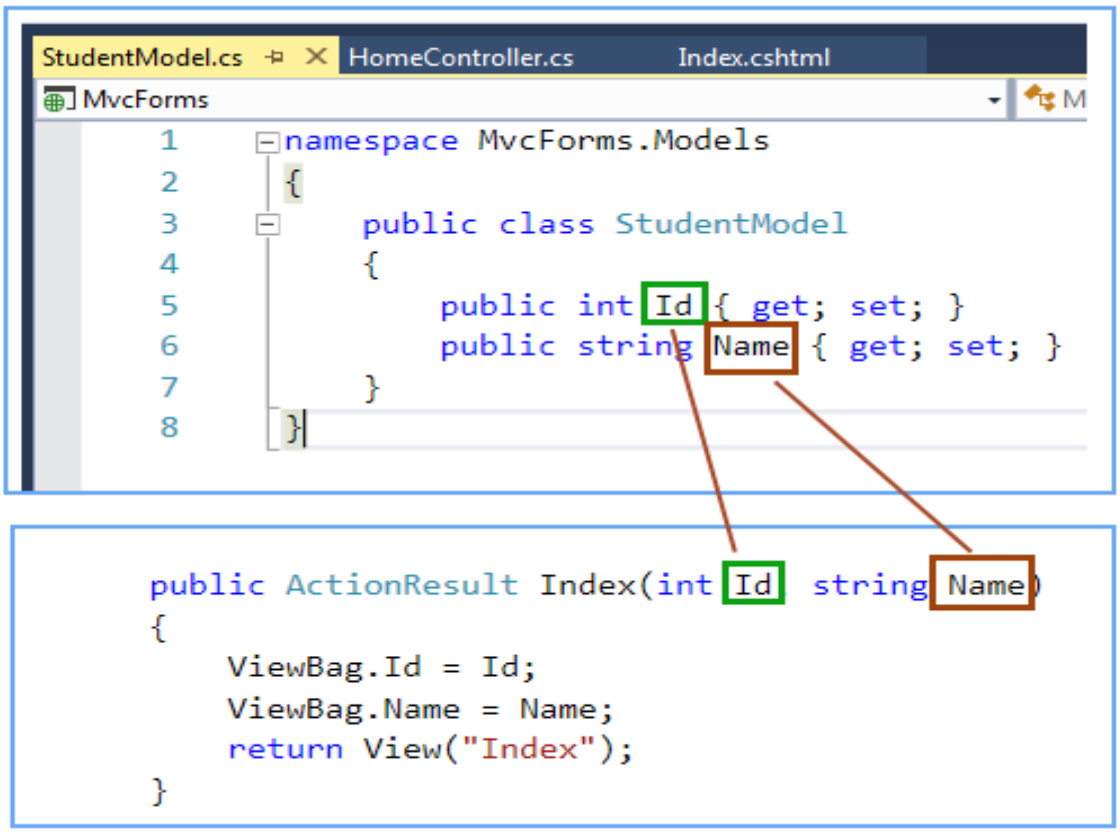
```
@using (Html.BeginForm())
{
<table>
<tr>
<td>Enter ID: </td>
<td>@Html.TextBox("Id")</td>
</tr>
<tr>
<td>Enter Name: </td>
        <td>@Html.TextBox("Name")</td>
        </tr>
        <tr>
            <tdcolspan="2">
            <inputtype="submit"value="Submit"></td>
        </tr>
        </table>
        }
        <h4style="color:purple">
            ID: @ViewBag.Id<br/>
            Name: @ViewBag.Name<br/>
        </h4>
```

Controller: HomeController.cs:-

```
    publicActionResultIndex()
    {
      if(Request.Form.Count>0)
      {
        ViewBag.Id=Request.Form["Id"];
        ViewBag.Name=Request.Form["Name"];
        returnView("Index");
       }
        returnView();

    }
```

SIMPLE BINDING:-

In simple binding, pass the parameter in action method with the same name as model properties and MVC Default Binder will automatically map correct action method for the incoming request.



Controller: HomeController.cs

```
public ActionResult Index(int Id, string Name)
    {
        ViewBag.Id = Id;
        ViewBag.Name = Name;
return View("Index");
```

```
}
```

CLASS BINDING:-In Class Binding, pass model as a parameter in action method and then access its entire member variable.

```
public ActionResult Index(StudentModel sm)
{
    ViewBag.Id = sm.Id;
    ViewBag.Name = sm.
    return View("Index
}

public ActionResult Ab
{
    ViewBag.Message =
```

Equals
GetHashCode
GetType
Id
Name — string StudentModel.
ToString

Controller: HomeController.cs

```
public ActionResult Index(StudentModelsm)
 {
        ViewBag.Id = sm.Id;
        ViewBag.Name = sm.Name;
return View("Index");
}
```

COMPLEX BINDING:-Complex Binding means apply multiple related models in a single view. Here, in this example, I will show you how to bind multiple models in a single view.

Step 1: Create two models class like that.

Model: StudentModel.cs

```
namespaceMvcForms.Models
{
public class StudentModel
   {
publicint Id { get; set; }
public string Name { get; set; }

publicCourseModelcourseModel { get; set; }
   }}
```

Model: CourseModel.cs:-

```
namespaceMvcForms.Models
{
public class CourseModel
    {
publicint Id { get; set; }
public string Course { get; set; }
public string Duration { get; set; }
    }
}
```

I have added CourseModel as a property in StudentModel, so the StudentModel has right to access CourseModel Property.

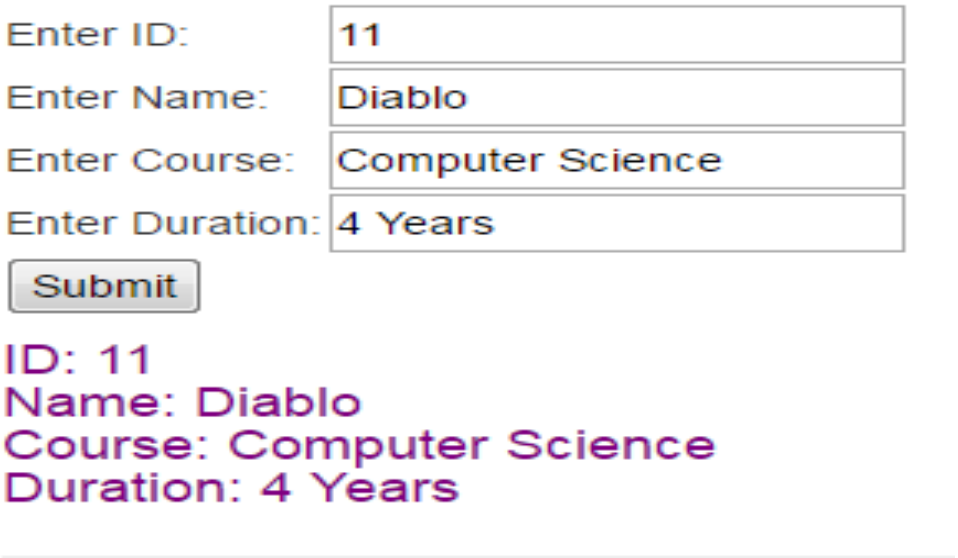
View:

Index.cshtml:-

```
@model
MvcForms.Models.StudentModel
@using (Html.BeginForm())
{
<table>
<tr>
<td>Enter ID: </td>
<td>@Html.TextBoxFor(m
=>m.Id)</td>
</tr>
<tr>
<td>Enter Name: </td>
<td>@Html.TextBoxFor(m
=>m.Name)</td>
</tr>
<tr>
<td>Enter Course: </td>
<td>@Html.TextBoxFor(m
=>m.courseModel.Course)</td>
</tr>
<tr><td>Enter Duration: </td>
<td>@Html.TextBoxFor(m
=>m.courseModel.Duration)</td
>
</tr>
<tr>
<td colspan="2"><input
type="submit"
value="Submit"></td>
</tr>
</table>
}
<h4 style="color:purple">
   ID: @ViewBag.Id<br />
   Name: @ViewBag.Name<br
/>
   Course:
@ViewBag.Course<br />
Duration:
@ViewBag.Duration<br />
</h4>
```

Step 3: Finally, the Action Method

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

        [HttpPost]
public ActionResult Index(StudentModelsm)
        {
            ViewBag.Id = sm.Id;
            ViewBag.Name = sm.Name;
ViewBag.Course = sm.courseModel.Course;
ViewBag.Duration = sm.courseModel.Duration;
return View("Index");
        }
```



SATHYATECHNOLOGIES

Chapter-15

MVC Basic Programs

KANNABABU

By.B.Kannababu

MVC Basic Examples:-

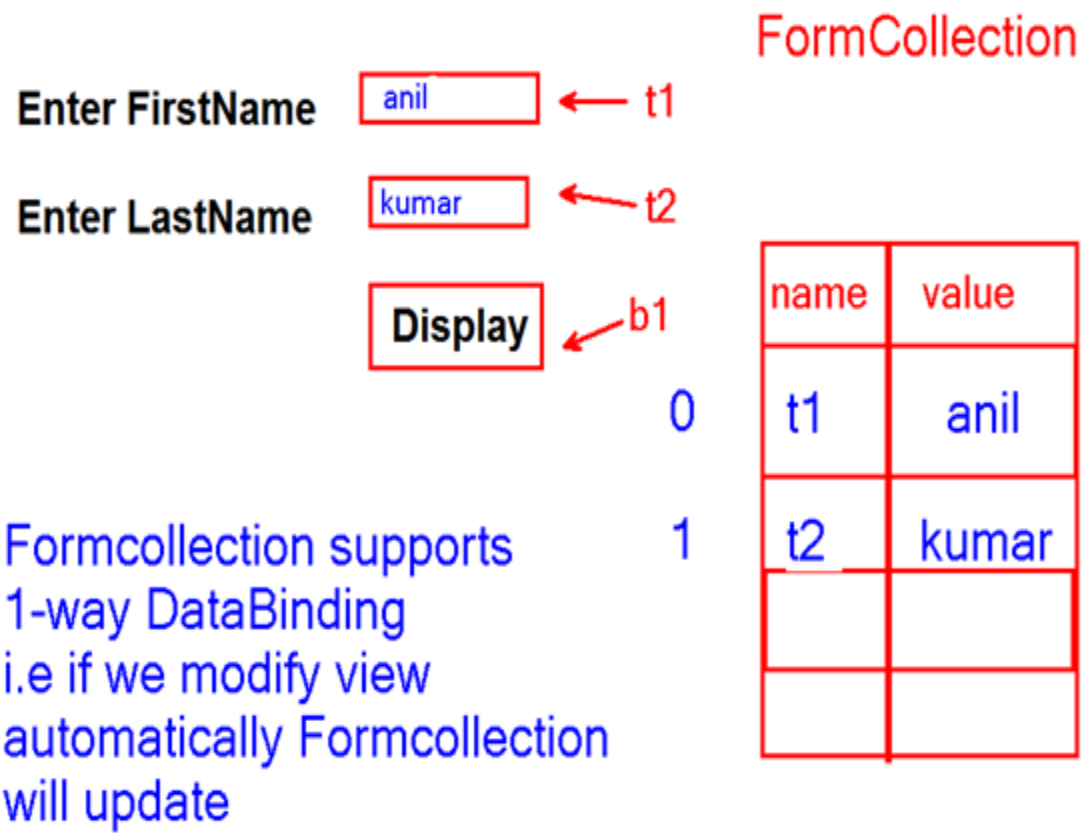
index.cshtml

```
<html>
<head></head>
<body>
   <form name="f1" method="post" action="/Home/Index">
      Enter FirstName
      <input type="text" name="t1" />
      <br />
      Enter LastName
      <input type="text" name="t2" />
      <br />
      <input type="submit" name="b1" value="Display" />
      <br />
   </form>
</body>
</html>
```

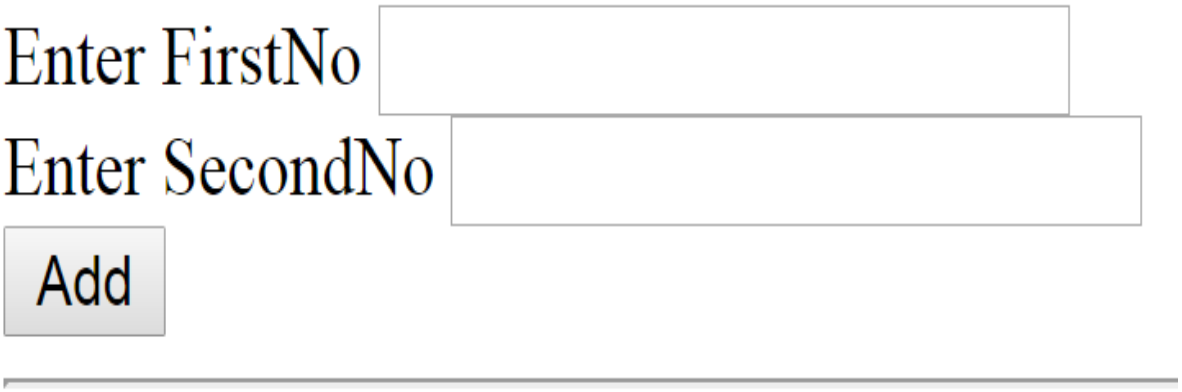
2. goto--->File-->new project--->select ASp.net Webapplication---->
web Empty---> MVC---> ok
goto--> solutionexplorer--> rc on Controllers-->Add Controller-->
name=HomeController---->

```
public class HomeController : Controller
{
    [HttpGet]
  public ActionResult Index()
  {
    return View();
  }
  [HttpPost]
  public string Index(FormCollection f)
  {
    string fullname = f[0] + f[1];
    return fullname;
  }
}
```

Formcollection:- it is used to catch the form values within the controller



Ex:-2



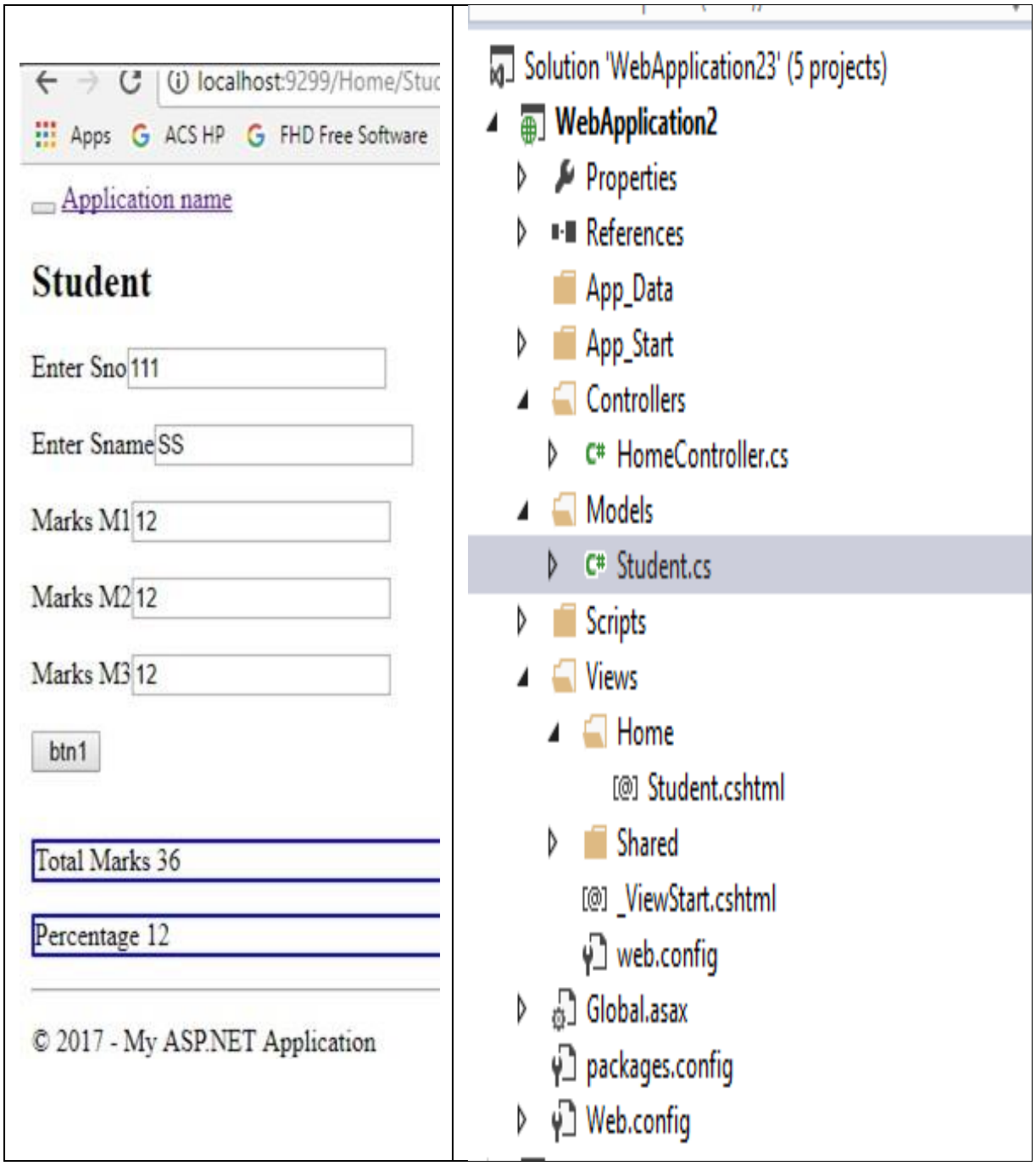
HomeController.cs:-

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.Mvc;
namespaceWebApplication14.Controllers
{
publicclassHomeController : Controller
   {
publicActionResultIndex()
        {
returnView();
        }
        [HttpPost]
publicstringIndex(FormCollectionf)
        {
intx=int.Parse(f[0]);
inty=int.Parse(f[1]);
intz=x+y;
return"sum is "+z;
        }
        }}
```

Code for index.cshtml:-

```
<html>
<head></head>
<body>
<formname="f1"method="post"action="/Home/Index">
        Enter FirstNo
<inputtype="text"name="t1"/>
<br/>
        Enter SecondNo
```

```
<inputtype="text"name="t2"/>
<br/>
<inputtype="submit"value="Add"/>
<br/>
</form>
</body>
</html>
```



HomeController.cs:-

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.Mvc;
using WebApplication2.Models;
namespace WebApplication2.Controllers
{
public class HomeController : Controller
  {
public ActionResult Index()
     {
return View();
   }
public ActionResult Student(Student obj)
{
int total = obj.M1 + obj.M2 + obj.M3;
int percentage = total / 3;
ViewBag.total = total;
ViewBag.percentage = percentage;
return View();
} }  }
```

Models→Student.cs:-

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
```

```
usingSystem.Web;
namespace WebApplication2.Models
{
public class Student
{
publicintSno { get; set; }
public string Sname { get; set; }
publicint M1 { get; set; }
publicint M2 { get; set; }
publicint M3 { get; set; }
publicint Total { get; set; }
publicint Percentage { get; set; }
}
}
```

Views-→Student.cshtml:-

```
@{
ViewBag.Title = "Student";
}
<h2>Student</h2>@{
using (Html.BeginForm())
{
@Html.Label("Enter Sno")
```

```
@Html.TextBox("Sno")
<br /><br />
@Html.Label("Enter Sname")
@Html.TextBox("Sname")
<br /><br />
@Html.Label("Marks M1")
@Html.TextBox("M1")
<br /><br />
@Html.Label("Marks M2")
@Html.TextBox("M2")
<br /><br />
@Html.Label("Marks M3")
@Html.TextBox("M3")
<br /><br />
<input type="submit" value=" btn1" name="CALCULATE" />
<br /><br />
}
}
@{
if(IsPost==true)
{
int total = ViewBag.total;
```

```
int percentage = ViewBag.percentage;
<p style="border :2px solid blue;font:2px">Total Marks
@total</p>
<p style="border :2px solid blue;font:2px">Percentage
@percentage</p>
}
}
```

Enter Ino 1
Enter ItemName Vegpizaa
Enter Qty 5
Price 120

Display

Total bill :- 600

Enter Eno 101
Enter Ename anil
Enter BasicSalary 20000

Display

Tsal is :- 32000

da=0.2*bsal-->4000
hra=0.4*bsal -->8000
Tsal=bsal+da+hra
20000+4000+8000

3.

Enter Sno
Enter Sname
Enter M1
Enter M2
Enter M3

if per>=75 and <=100 print A grade

if per>=60 and <75 print B Grade

if per>=40 and <60 print C Grade

Display

else print Fail

Total is

Percentage is

Grade is

5. Enter Itemno 101 Total Bill is 1120

Enter Product cost 1000

Enter CGST% 5

Enter SGST% 7

Display

CGSTAmt 50

SGCTAmt 70

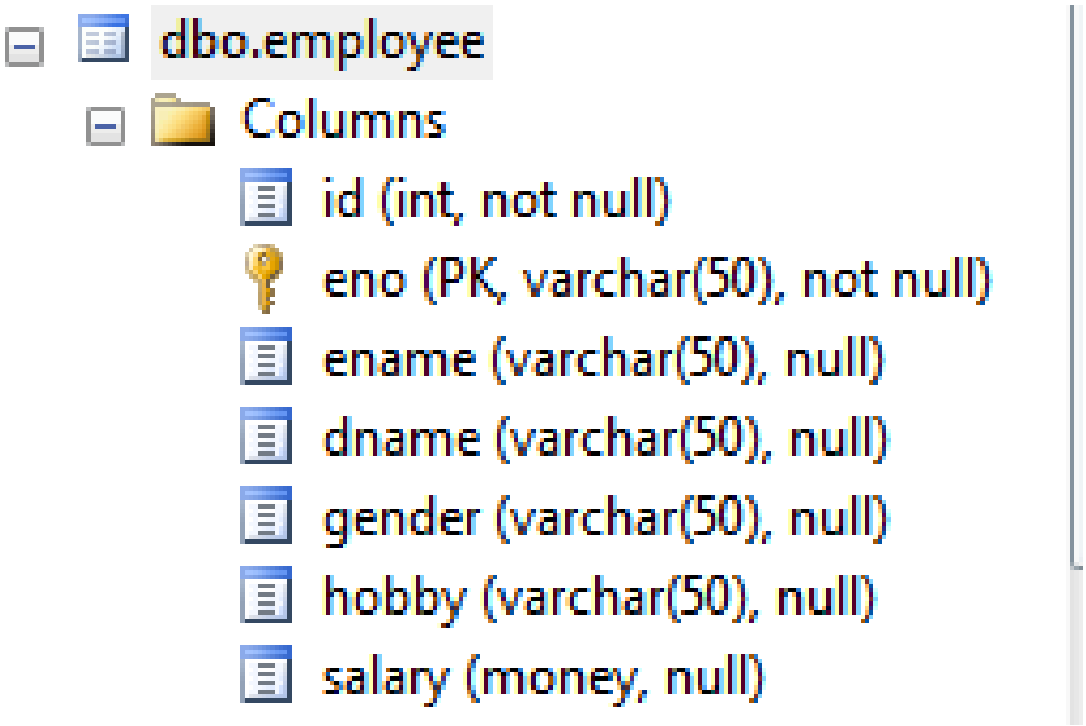
SATHYA TECHNOLOGIES

Chapter-16

ADO.net with ASP.net MVC

KANNABABU

By.B.Kannababu




```
createprocedure proc_addemp
(@eno varchar(50),
@ename varchar(50),
@dname varchar(50),
@gender varchar(50),
@hobby varchar(50),
@salary money)
asbegin
insertinto employee values(
@eno,@ename,@dname,@gender,
@hobby,@salary)
End
```

Models-→Employee.cs

```
usingSystem;
usingSystem.Collections.Generic;
namespaceSwiggy.Models
{
publicclassEmployee
   {
publicstringEno { get; set; }
publicstringEname { get; set; }
publicdecimalSalary { get; set; }
publicList<string>Hobbies
      {
get
         {
List<string>hobby=newList<string>();
hobby.Add("Cricket");
hobby.Add("Football");
hobby.Add("Hockey");
```

```
returnhobby;
        }     }
publicstringGender { get; set; }
publicList<string>Dnames
      {
get
        {
List<string>Dname=newList<string>();
Dname.Add("Accounts");
Dname.Add("Quality");
Dname.Add("Admin");
returnDname;
        }
      }
publicstringDname { get; set; }
publicstringHobby { get; set; }
   }
}
```

Code for DAL(Employee.cs):-

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Data;
usingSystem.Data.SqlClient;
usingSystem.Configuration;
namespaceSwiggy.DAL
{
publicclassEmployee
   {
publicintAddEmployee(Models.Employeee1)
      {
SqlConnectioncon=newSqlConnection(ConfigurationManager.Conn
ectionStrings["constr"].ToString());
con.Open();
```

```
SqlCommandcmd=newSqlCommand("proc_addemp",con);
cmd.CommandType=CommandType.StoredProcedure;
cmd.Parameters.AddWithValue("@eno", e1.Eno);
cmd.Parameters.AddWithValue("@ename", e1.Ename);
cmd.Parameters.AddWithValue("@dname", e1.Dname);
cmd.Parameters.AddWithValue("@gender", e1.Gender);
cmd.Parameters.AddWithValue("@hobby", e1.Hobby);
cmd.Parameters.AddWithValue("@salary", e1.Salary);
inti=cmd.ExecuteNonQuery();
con.Close();
returni;
        }
    }
}
```

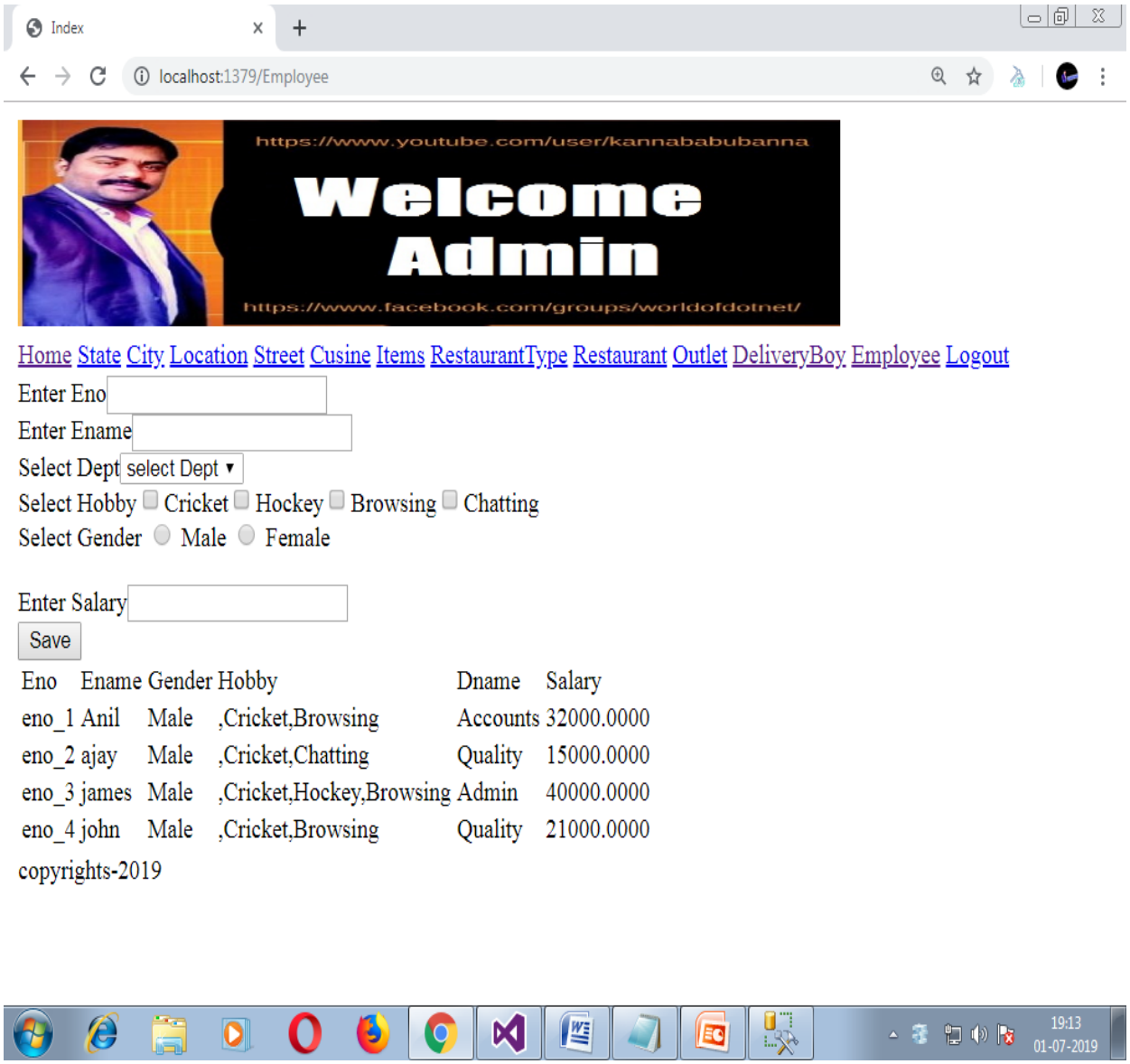
Code for EmployeeController.cs:-

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.Mvc;
usingSwiggy.Models;
usingSwiggy.DAL;
namespaceSwiggy.Controllers
{
publicclassEmployeeController : Controller
    {
DAL.Employeeobjdalemp=newDAL.Employee();
publicActionResultIndex()
        {
Models.Employeeobjempmodel=newModels.Employee();
List<SelectListItem>li=newList<SelectListItem>();
li.Add(newSelectListItem() { Text="Select Dept", Value="0" });
foreach (varitemin objempmodel.Dnames.ToList())
            {
li.Add(newSelectListItem() { Text=item });
            }
```

```
TempData["lihb"] =objempmodel.Hobbies;
TempData.Keep();
TempData["lidept"] =li;
TempData.Keep();
returnView();
        }
        [HttpPost]
publicActionResultIndex(FormCollectionf,stringGender,Models.Emp
loyeee1)
        {
Models.Employeem=newModels.Employee();
m.Eno=f[0];
m.Ename=f[1];
m.Dname=f[2];
m.Gender=Gender;
stringval="";
foreach (varitemine1.Hobbies)
            {
if (f[item].ToString().Contains("true"))
                {
val=val+","+item;
                }
            }
m.Hobby=val;
m.Salary=decimal.Parse(f["salary"]);
inti=objdalemp.AddEmployee(m);

returnView();
        }
        }
}
```

Code for index.cshtml:-

```
@model Swiggy.Models.Employee
@{
ViewBag.Title="Index";
Layout="~/Views/_AdminLayout.cshtml";
}
@using(Html.BeginForm("Index","Employee",FormMethod.Post))
{
@Html.Label("Enter Eno")
@Html.TextBoxFor(model=>model.Eno)
<br/>
@Html.Label("Enter Ename")
@Html.TextBoxFor(model=>model.Ename)
<br/>
@Html.Label("Select Dept")
@Html.DropDownList("dept", TempData["lidept"]
asList<SelectListItem>)
<br/>
@Html.Label("Select Hobby")
foreach (stringitemin TempData["lihb"] asList<string>)
   {
@Html.CheckBox(item)@item
   }
<br/>
<div>
@Html.Label("Select Gender")
<inputid="Gender"name="Gender"value="Male"type="radio">
      Male
<inputid="Gender"name="Gender"value="FeMale"type="radio">
      Female
</div>
@Html.Label("Enter Salary")
@Html.TextBoxFor(model=>model.Salary)
<br/>
<inputtype="submit"value="Register"/>
}
```



Code for DAL:-

```
publicIEnumerable<Models.Employee>GetEmps()
        {
List<Models.Employee>emps=newList<Models.Employee>();
SqlConnectioncon=newSqlConnection(ConfigurationManager.Conn
ectionStrings["constr"].ToString());
con.Open();
SqlCommandcmd=newSqlCommand("proc_getemps",con);
SqlDataReaderdr=cmd.ExecuteReader();
```

```
if(dr.HasRows)
        {
while (dr.Read())
            {
Models.Employeeew=newModels.Employee();
ew.Eno=dr[0].ToString();
ew.Ename=dr[1].ToString();
ew.Gender=dr[2].ToString();
ew.Hobby=dr[3].ToString();
ew.Dname=dr[4].ToString();
ew.Salary=Convert.ToDecimal(dr[5]);
emps.Add(ew);
            }
        }
returnemps;
    }
```

Code for EmployeeController.cs:-

```
publicclassEmployeeController : Controller
  {
DAL.Employeeobjdal=newDAL.Employee();
publicActionResultIndex()
    {
Models.Employeeobjmodelemp=newModels.Employee();
List<SelectListItem>lidept=newList<SelectListItem>();
lidept.Add(newSelectListItem() { Text="select Dept", Value="" });
foreach (varitemin objmodelemp.Depts.ToList())
        {
lidept.Add(newSelectListItem() { Text=item });
        }
TempData["lidept"] =lidept;
TempData.Keep();
TempData["hobby"] =objmodelemp.Hobbies;
TempData.Keep();
IEnumerable<Models.Employee>ie=objdal.GetEmps().ToList();
TempData["ie"] =ie;
TempData.Keep();     returnView();     }
```

```
        [HttpPost]
publicActionResultIndex(FormCollectionf,stringgender,M
odels.Employeee1)
        {
Models.Employeem=newModels.Employee();
m.Eno=f[0];
m.Ename=f[1];
m.Dname=f[2];
m.Gender=gender;
stringval="";
foreach (varitemine1.Hobbies)
{
if (f[item].ToString().Contains("true"))
 {
    val=val+","+item;
  }
}
m.Hobby=val;
m.Salary=e1.Salary;
inti=objdal.AddEmp(m);
if (i==1)
          {
returnRedirectToAction("Index");
          }
returnView();
        }   }
```

index.cshtml:-

```
@model Swiggy.Models.Employee

@{
ViewBag.Title="Index";
Layout="~/Views/_AdminLayout.cshtml";
}
@using(Html.BeginForm("Index","Employee",FormMethod.Post
))
{
@Html.Label("Enter Eno")
@Html.TextBoxFor(model=>model.Eno)
<br/>
@Html.Label("Enter Ename")
@Html.TextBoxFor(model=>model.Ename)
<br/>
@Html.Label("Select Dept")
@Html.DropDownList("dept", TempData["lidept"]
asList<SelectListItem>)
<br/>
@Html.Label("Select Hobby")
foreach (stringiteminTempData["hobby"] asList<string>)
   {
@Html.CheckBox(item)@item
   }
<br/>
<div>
@Html.Label("Select Gender")
<inputid="Gender"name="Gender"value="Male"type="radio">
      Male
<inputid="Gender"name="Gender"value="FeMale"type="radio"
>
      Female
</div>
<br/>
```

```
@Html.Label("Enter Salary")
@Html.TextBoxFor(model=>model.Salary)
<br/>
<inputtype="submit"value="Save"/>
<table>
<tr>
<td>Eno</td>
<td>Ename</td>
<td>Gender</td>
<td>Hobby</td>
<td>Dname</td>
<td>Salary</td>
</tr>
@foreach (variteminTempData["ie"]
asIEnumerable<Swiggy.Models.Employee>)
        {
        <tr>
<td>@item.Eno</td>
<td>@item.Ename</td>
<td>@item.Gender</td>
<td>@item.Hobby</td>
<td>@item.Dname</td>
<td>@item.Salary</td>
</tr>
        }
</table>

}
```

Enter Eno

Enter Ename

Enter Salary

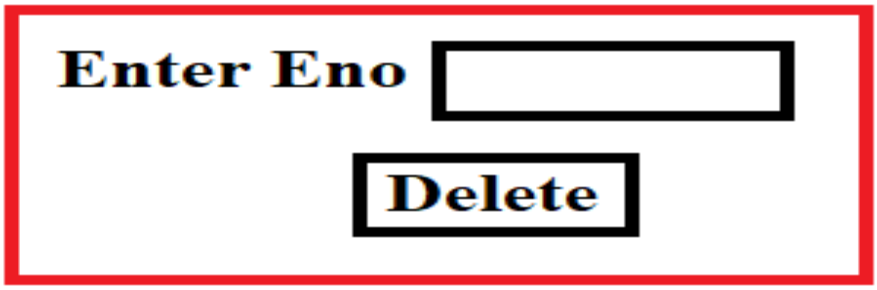
Enter Address

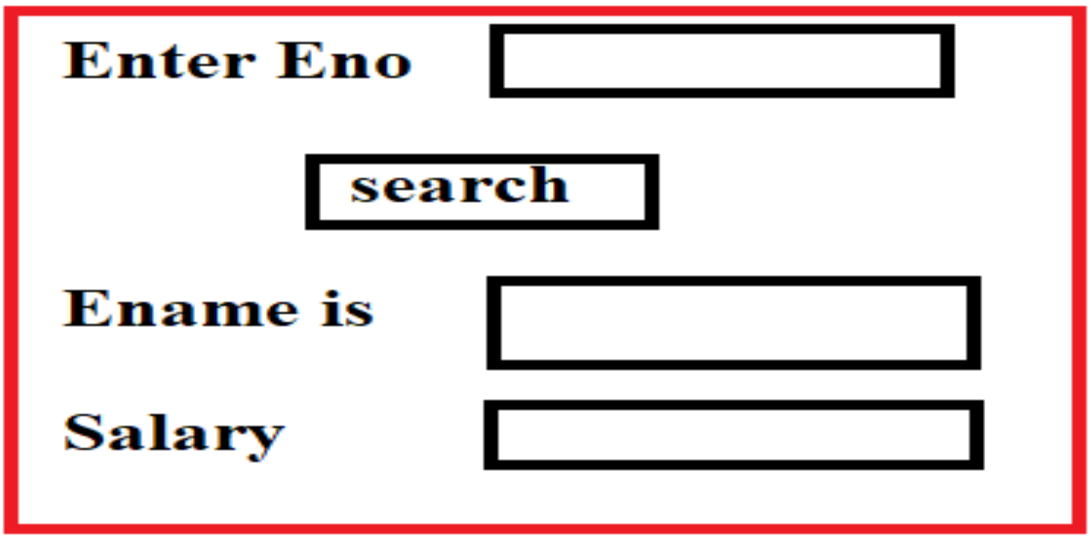
save update

Eno	Ename	Salary	Address		
101	anil	20000	Hyd	Delete	Edit
102	sunil	30000	Chennai	Delete	Edit
103	ajay	25000	Banglore	Delete	Edit

Requirement:-

1. when user clicks on save button save emp details in emp table and display emp details in Html Table
2. when user clicks on Delete Button Delete the record
3. when user clicks on Edit button displayt the record in Textboxes
4. when user clicks on update button update emp details





SATHYATECHNOLOGIES

Chapter-17

DataAnnotations

By.B.Kannababu

KANNABABU

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.

Data Annotation can be used after adding following namespace.

usingSystem.ComponentModel.DataAnnotations

usingSystem.ComponentModel;

1. **Required**

Specifies that Input field cannot be empty.

Example:

```
[Required(ErrorMessage = "Name is Required")]
public string Name { get; set; }
```

2. **DisplayName**

Specifies the Display Name for a Property.

Example:

```
[DisplayName("Enter Your Name: ")]
public string Name { get; set; }
```

3. **StringLength**

Specifies minimum and maximum length for a property.

Example:

```
[StringLength(50, MinimumLength = 3)]
public string Name { get; set; }
```

4. Range

Specifies a range of numeric value.

Example:

[Range(1,120, ErrorMessage ="Age must be between 1-120 in years.")]

publicint Age { get; set; }

5. ScaffoldColumn

Specifies a field for hiding from editor forms.

Example:

[ScaffoldColumn(false)]

publicint Id { get; set; }

6**. DisplayFormat**

Specifies a display format for a property like Date Format, Currency format etc.

Example:

[DisplayFormat(DataFormatString = "{0:dd/MM/yyyyhh:mm:sstt}")]

publicSystem.DateTime? HireDate{ get; set; }

7. ReadOnly

It set a property to read-only.

Example:

[ReadOnly(true)]

public string Name { get; private set; }

8. MaxLength

Specifies max length of string.

Example:

```
[MaxLength(50)]
public string Name { get; set; }
```

9. CreditCard

specifies that a data field value is credit card number.

Example:

```
[DataType(DataType.CreditCard)]
public string Name { get; set; }
```

10. Compare

compare with other input fields.

Example:

```
[System.ComponentModel.DataAnnotations.Compare("Email",
ErrorMessage = "Email Not Matched")]
public string ConfirmEmail { get; set; }
```

11. EmailAddress

specifies that an input field value is well-formed Email Address using Regular Expression.

Example:

```
[Required(ErrorMessage = "Email ID is Required")]
[DataType(DataType.EmailAddress)]
[MaxLength(50)]
[RegularExpression(@ "[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}",
ErrorMessage = "Incorrect Email Format")]
public string Email { get; set;}
```

12. Phone

specifies that an input field value is well-formed phone number using Regular Expression.

Example:

```
[DataType(DataType.PhoneNumber)]
[RegularExpression(@"^\(?([0-9]{2})[-. ]?([0-9]{4})[-. ]?([0-9]{3})[-. ]?([0-9]{3})$", ErrorMessage = "Not a valid Phone number")]
public string Name { get; set; }
```

Output format: **91-1234-567-890**

13. Url

specifies URL validation.

Example:

```
[Url][Required]
public string URL { get; set; }
```

14. RegularExpression

specifies that input field is matched with desired Regular Expression.

Example:

```
[RegularExpression(@"[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}", ErrorMessage = "Incorrect Email Format")]
public string Email { get; set; }
```

COMPLETE PROGRAMMING EXAMPLE

Let's understand all these with complete programming example.

Step 1: Create StudentModel.cs

```
usingSystem.ComponentModel.DataAnnotations;
usingSystem.ComponentModel;
usingSystem.Web.Mvc;
namespaceFormValidation.Models
{
   [Bind(Exclude = "Id")]
public class StudentModel
   {
       [ScaffoldColumn(false)]
publicint Id { get; set; }

       [Required(ErrorMessage = "Name is Required")]
       [StringLength(50,MinimumLength =3)]
public string Name { get; set; }

       [Required(ErrorMessage ="Email ID is Required")]
       [DataType(DataType.EmailAddress)]
       [MaxLength(50)]
       [RegularExpression(@"[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}",
ErrorMessage = "Incorrect Email Format")]
public string Email { get; set; }

       [Required(ErrorMessage ="Confirm Email is Required")]
       [DataType(DataType.EmailAddress)]
       [System.ComponentModel.DataAnnotations.Compare("Email",
ErrorMessage = "Email Not Matched")]
public string ConfirmEmail { get; set; }

       [Required(ErrorMessage ="Age is Required")]
```

```
[Range(1,120, ErrorMessage ="Age must be between 1-120 in
years.")]
publicint Age { get; set; }
    }
}
```

Create a form in Index.cshtml page:-

```
@{
ViewBag.Title = "Home Page - Student Details";
}
<scriptsrc="~/Scripts/jquery.validate.min.js"></script>
<scriptsrc="~/Scripts/jquery-1.10.2.min.js"></script>
@model FormValidation.Models.StudentModel
<h2>Student Details</h2>
@using (Html.BeginForm("StudentDetails", "Home",
FormMethod.Post))
{
<ol>
<li>
@Html.LabelFor(m =>m.Name)
@Html.TextBoxFor(m =>m.Name)
@Html.ValidationMessageFor(m =>m.Name)
</li>
<li>
@Html.LabelFor(m =>m.Email)
@Html.TextBoxFor(m =>m.Email)
@Html.ValidationMessageFor(m =>m.Email)
</li>
<li>
@Html.LabelFor(m =>m.ConfirmEmail)
@Html.TextBoxFor(m =>m.ConfirmEmail)
@Html.ValidationMessageFor(m =>m.ConfirmEmail)
</li>
<li>
```

```
@Html.LabelFor(m =>m.Age)
@Html.TextBoxFor(m =>m.Age)
@Html.ValidationMessageFor(m =>m.Age)
</li>
</ol>
<input type="submit" value="Save Student Details" />
}

<h3 style="color:green">Student Details</h3>
<h4 style="color:green">
<b>Name: @ViewBag.name<br />
      Email: @ViewBag.email<br />
      Age: @ViewBag.age</b>
</h4>
```

Step 3: Add following code in `HomeController.cs`

```
ttpPost]
public ActionResult StudentDetails(StudentModelsm)
{
if (ModelState.IsValid)
 {
  ViewBag.name = sm.Name;
ViewBag.email = sm.Email;
ViewBag.age = sm.Age;
return View("Index");
 }
else
 {
  ViewBag.name = "No Data";
ViewBag.email = "No Data";
ViewBag.age = "No Data";
return View("Index");
 }
}
```

Student Details

1. **Name** Name is Required
2. **Email** Email ID is Required
3. **ConfirmEmail** Confirm Email is Required
4. **Age** Age is Required

Save Student Details

Student Details

Name: No Data
Email: No Data
Age: No Data

SATHYA TECHNOLOGIES

Chapter-18

Filters in MVC

By.B.Kannababu

KANNABABU

•Filters are used to execute some logic either before or after executing ActionMethods

- Filters can be applied to an action method or controller in a declarative or programmatic way.
- Declarative means by applying a filter attribute to an action method or controller class
- Programmatic means by implementing a corresponding interface.

Authorization Filters:

It is used to implement authorization and authentication for action filters Result Filters:

Result filters contains logic that gets executed before or after a view result gets executed. E.g. if you want to change view before its get render to browser.

Exception Filters:

Exception filters are used to handle error, caused by either controller action or controller action results, we can also use it for logging the exceptions.

Authorize Filters:-

- MVC provides 2 Filters to perform Authetication and Authorization

Q)What is Authentication?

Authetication is the process of checking usercredentialsUsercredentials means username and password Any user who is having username and password then that user is called as Autheticated user

Q)What is Authorization?

Authorization is the process of assigning the roles and Responsibilites for the Autheticated users In MVC we can provide security in different ways:-

- Windows Authnetication
- Forms Autehtication
- Passport Autehtication

- Open Autehtication

Windows Authentication:-It is a process of providing acces to the webpages present in the website based on Windows O.S user credentials

Steps to work with Windows Authentication:-

Declaring Authetication mode in web.config file

<authetication mode="windows"/>

Apply Authorize filter for controller Actionmethod

[Authorize(users="servername\\username")]

Forms Authetication is cookie based Authetication where usernames and password will be save in Browser in the form of cookie

Ex:-

```
usingFormsAutheticationDemo.Models;
usingSystem.Web.Security;
public class HomeController : Controller
{
[HttpGet]
public ActionResult Login()
{
return View();
}
[HttpPost]
public ActionResult
Login(FormsAutheticationDemo.Models.Loginobjmodellogin)
{
if(objmodellogin.UserName=="Admin" ||
objmodellogin.UserName=="Anil"
&&objmodellogin.Password=="Admin" ||
objmodellogin.Password=="Anil")
{
FormsAuthentication.RedirectFromLoginPage(objmodellogin.UserN
ame,false);
```

```
}
else
{
Response.Write("invalid user”);  }
return View();
}
public ActionResult Index()
{       return View();    }
}
```

Model class:-

```
public class Login
{
public string UserName { get; set; }
public string Password { get; set; }}
}
```

web.config:-

```
<authentication mode="Forms">
<forms defaultUrl="~/Home/Index" loginUrl="~/Home/Login">
<credentials>
<user name="Admin" password="Admin"/>
<user name="Anil" password="Anil"/>
</credentials>
</forms>
</authentication>
<authorization>
<deny users="Anil"/>
</authorization>
</system.web>
```

Caching:-

```
using System;
usingSystem.Web.Mvc;
namespace WebApplication2.Controllers
{
public class HomeController : Controller
{
[OutputCache(Duration=10)]
public ActionResult Index()
{
ViewData["Message"] = System.DateTime.Now.ToString() ;
return View();
}
}
}
```

ExceptionFilter:-

```
using System;
using System.Web.Mvc;
namespace WebApplication4.Controllers
{
    public class MyExceptionFilter : FilterAttribute, IExceptionFilter
    {
        public void OnException(ExceptionContext filterContext)
        {
            if (!filterContext.ExceptionHandled &&
filterContext.Exception is DivideByZeroException)
            {
                filterContext.Result = new
RedirectResult("/Home/ErrorPage");
                filterContext.ExceptionHandled = true;
            }
        }
    }
```

```
[MyExceptionFilter]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        int a = 10;
        int b = 0;
        int c = a / b;
        ViewBag.Message = c;
        return View();
    }
    public ActionResult ErrorPage()
    {
        return View();
    }
}
}
```

Index.cshtml:-

```
@{
    ViewBag.Title = "Index";
}
<h1>

</h1>
```

ErrorPage.cshtml:-

```
@{
    ViewBag.Title = "ErrorPage";
}

<h2>ErrorPage</h2>

Denominator must not be 0
```

Filter Type	Description	Built-in Filter	Interface
Authorization filters	Performs authentication and authorizes before executing action method.	[Authorize], [RequireHttps]	IAuthorizationFilter
Action filters	Performs some operation before and after an action method executes.		IActionFilter
Result filters	Performs some operation before or after the execution of view result.	[OutputCache]	IResultFilter
Exception filters	Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline.	[HandleError]	IExceptionFilter