MVC -

ASP.NET MVC :-

- ASP.NET is a free web framework for building websites and web applications on .NET framework
- ASP.NET MVC 5 is a web framework based on Model - View - Controller (MVC) architecture.
- It is an open-source software from Microsoft
- It's web development framework combines the features of MVC architecture, the most up-to-date ideas and techniques.
- The whole idea behind using the Model View Controller design pattern is that you maintain a sp seperation of concerns.

MVC Architecture :-

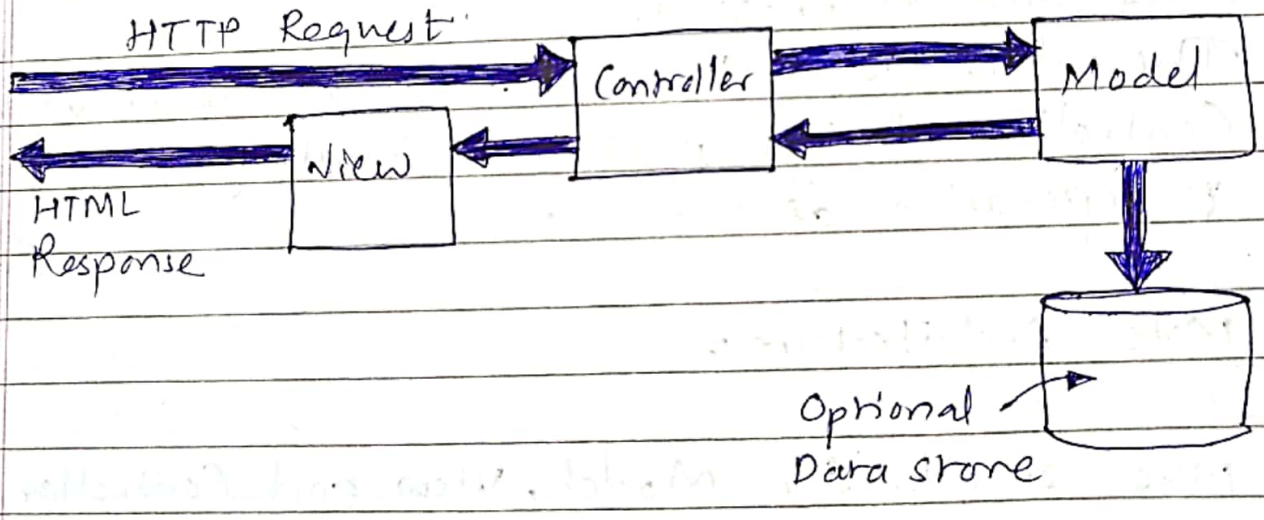
MVC stands for Model, View and Controller. MVC seperates an application into three components.- Models, View and Controller.

- **Model** : Model Represents the data
  A class in C# is used to describe a model :Model objects store data retrieved from the database.

- **View** : View is the User Interface.
  View in MVC is a user Interface. View display model data to the user and also enables them to modify them. View in ASP.NET MVC is HTML .CSS

and some special syntax that makes it easy to communicate with the model and the controller.

- Controller: Controller is the request handler. The controller handles the user request. Typically, the user uses the view and raises an HTTP request, which will be handled by the controller. The controller processes the request and returns the appropriate view as a response.



Controller handles the user interaction request & select appropriate action handler to execute and passes the query string values to the model, which in turn can query database based on information received from controller.

ASP.NET MVC Folder Structure :-

- **App_Data :-**

  The App_Data folder can ~~can~~ contain application data files like LocalDB, .mdf files, XML files, and other data related files. IIS will never serve files from App_Data folder.

- **App_Start :-**

  The App_Start folder can contain class files that will be executed when the application starts. Typically, these would be config files AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs etc. MVC 5 includes ~~BundleConfig~~ BundleConfig.cs, FilterConfig.cs and RouteConfig.cs by default.

- **Content :-**

  The Content folder contains static files like css files, images and icons files. MVC 5 application includes bootstrap.css, bootstrap.min.css, and Site.css by default.

- **Controllers :-**

  The Controllers folder contains class files for the controllers. A Controller handles users request and returns a response. MVC requires the name of all controller files ~~end~~ to end with "Controller".

DATE / /

- **Fonts:-**
  The fonts folder contains custom font files for your application.

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

- **Scripts:-**
  The Scripts folder contains Javascript or VB Script files for the application. MVC 5 includes javascript files for bootstrap, jquery 1.10, and modernizer by default

- **Views:-**
  The 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.
  The Views folder includes a seperate folder for each controller.

- **Global.asax:-**
  Global.asax file 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.

- **Packages.Config :-**
  Packages.config file is managed by ~~Nuget~~ to NuGet to track what packages and versions you have installed in the application.

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

NOTE :-
-@Html.ActionLink("LinkText", "ActionName", "ControllerName")

Action Method :-

All the public methods of the Controller class are called Action methods. They are like any other normal methods with the following restrictions :

- Action method must be public. It cannot be private or protected.
- Action method cannot be overloaded.
- Action method cannot be a static method.

Routing In MVC :-

ASP.NET introduced Routing to eliminate the needs of mapping each URL with a physical file. Routing enables us to define a URL pattern that maps to request handler. This request handler can be a file or class.

- Route :- Route defines the URL pattern and handler information. All the configured routes of an application stored in RouteTable and will be used by the Routing engine to determine appropriate handler class or file for an incoming request

- Configure a Route :- Every MVC application must configure (register) at least one route configured by the MVC framework by default. You can register a route in RouteConfig.class, which is in RouteConfig.cs under App start folder.

Routing in MVC :-

Route is configured using the MapRoute() extension method of RouteCollection, where name is "Default", url pattern is "{controller}/{action}/{id}" and defaults parameter for controller, action method and id parameter

PAGE No.
DATE / /

Controller in MVC :-

- The controller in MVC architecture handles any incoming URL request.
- The controller is a class, derived from the base class. System.Web.Mvc.Controller.
- Controller class contains public methods called Action Methods.
- Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.
- In ASP.NET MVC, every controller class name must end with a word "Controller".
- For Example, the Home page controller name must be HomeController, and for the student page, it must be the StudentController. Also, every controller class must be located in the Controller folder of the MVC folder structure.

Action Method :-

All the public methods of the Controller class are called Action methods. They are like any other normal methods with the following restrictions:

- Action method must be public. It cannot be private or protected.
- Action method cannot be overloaded
- Action method cannot be a static method.

ActionResult :-

- MVC framework includes various Result classes, which can be returned from an action method.
- The result class represent different types of responses, such as HTML, file, string, JSON, JavaScript etc.
- The ActionResult class is a base class of all the above result classes, so it can be the return type of action method that returns any result.
- However, you can specify the appropriate result class as a return type of action method.

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	Represents a Javascript script.	JavaScript()
JsonResult	Represents JSON that can be used in AJAX	Json()
RedirectResult	Represents a redirection to ~~another route~~ new URL	Redirect()

RedirectToRouteResult	Represents redirection to another route	RedirectToRoute()
PartialViewResult	Represents the partial view Result	PartialView()
HTTPUnauthorizedResult	Represents HTTP 403 response.	

Action Selectors :-

Action Selectors are attributes that are applied on action methods of a Controller. It is used to select correct action method to call as per the request. MVC provides the following action selector attributes :

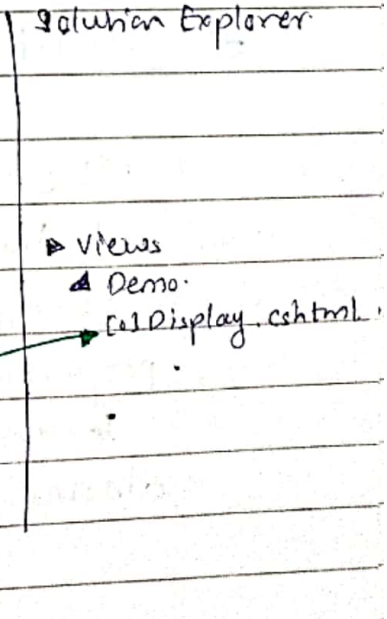
1) ActionName
2) ActionVerbs.

- ActionName =

This attribute allows us to specify a different name for the action method. It is useful when we want to call action by different name.

```
[ActionName("Display")]

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



Example

```
public string Index1()
{
    return "helloworld";
}
public int Index2()
{
    return id;
}
```

Non-Action Methods.

```
public ViewResult Index3()
{
    return View();
}
public ActionResult Index4()
{
    return View();
}
```

Action Methods.

Model Class:-

The model classes represents domain-specific data and business logic in the MVC application.
It represents the shape of the data as public properties and business logic as methods.
In the ASP.NET MVC Application, all the Model classes must be created in the Model folder.

View in ASP.NET MVC :-

- A view is used to display data using the model class object. The Views folder contains all the view files in the ASP.NET MVC application.
- A controller can have one or more action methods, and each action method can return a different view. In short, a controller can render one or more views.
- So, for easy maintenance, the MVC framework requires a seperate sub-folder for each controller with the same name as a controller, under the Views folder.
- The Shared folder contains views, layout views, and partial views, which will be shared among multiple controllers.

Razor View Engine :-

- Razor is a markup syntax that lets you embed server-based code into web pages using C# and VB.Net.
- It is not a programming language. It is a server side markup language.
- The special syntax for razor view maximizes the speed of writing code by minimizing the number of characters.
- You can use it anywhere to generate output like HTML
- It's just that ASP.NET MVC has implemented a view engine that allows us to use Razor inside of an MVC application to produce HTML.

- The razor view uses @ character to include the server-side code instead of the traditional <% %> of ASP.

File Extension	Description.
.cshtml	C# Razor view. Supports C# code with html tags.
.vbhtml	Visual Basic Razor view. Supports Visual Basic code with html tags.
.aspx	ASP.NET web form
.ascx	ASP.NET web control.

Razor Design Goals:-

- Microsoft wanted Razor to be easy to use and easy to learn, and work inside of tools like Visual Studio so that Intellisense is available, the debugger is available, but they wanted Razor to have no ties to a specific technology, like: ASP.NET or ASP.NET MVC.
- Microsoft wanted Razor to be smart, to make a developer's job easier.

Main Razor Syntax Rules for C# :-

- C# files have the extension .cshtml
- Razor code blocks are enclosed in @{ ... }
- Inline expressions (variables and functions) start with @.
- Code statements end with semicolon.
- Variables are declared with the 'var' keyword.
- strings are enclosed with quotations marks.
- C# code is case sensitive.

Transfer data from Controller to view.

- ViewData
- ViewBag
- TempData

Program:- (in .cshtml file)

```
@{var marks = 50;}

<html>
<body>
  <div>
    @if (marks >35)
    {
      <p> Pass </p>
    }
  </div>
  <div>
    @if (marks >70)
    {
      <p> Grade is A </p>
    }
    else if (marks >40)
    {
      <p> Grade is A </p>
    }
    else if (marks >35)
    {
      <p> Grade is A </p>
    }
    else
    {
      <p> Fail </p>
    }
  </div>
```

```
<div>
    <table class="table table-responsive">
        <tr>
            <th> ID </th>
            <th> Name </th>
            <th> City </th>
            <th> Mobile </th>
        </tr>
        @for (var i=0; i<10 ; i++)
        {
            <tr>
                <td> 001 </td>
                <td> Rutwi </td>
                <td> Sangli </td>
                <td> 0123456789 </td>
            </tr>
        }
    </table>
</div>
</body>
</html>
```

Data Sharing Techniques :-

Transfer Data from Controller to View

1) ViewData
2) ViewBag
3) TempData.

1) ViewData :-

- ViewData for simple Value :-

DemoController which contains index method

```
public ActionResult Index()
{
    ViewData["Data"] = "hello";
    ViewData.Add("Info","welcome");

    return View();
}
```

View page in which we are trying to access simple data using ViewData

```
<head>
    <title>Display</title>
</head>
<body>
    <div>
        @ViewData["data"] </br>
```

```
@ViewData["Info"]
</div>
</body>
</html>
```

- ViewData For Complex Value :-

DemoController which contains index method

```
public ActionResult Index()
{
  ViewData["Products"] = new List<string>()
  {
    "TV",
    "Laptop"
    "Mobile"
  };
  return View();
}
```

ViewPage in which we are trying to access complex data using ViewData

```
<body>
  <div>
    <ol>
      @foreach (var item in (List<string>)ViewData["Products"])
      {
        <li>@item</li>
      }
    </ol>
  </div>
</body>
```

about ViewData

- It is property of ControllerBase class. It is defined in System.Web.MVC
- ViewData is a type of ViewDataDictonary
- It stores value in key value format.
- It can stores complex values but can't return them directly on call back. It requires type casting.
- It is only available for current request
- There are two ways to define it:
  (1) ViewData.Add("key", "value");
  (2) ViewData["key"] = Value;
- ViewData only transfers data from controller to view, not vice-versa. It is valid only during the current request.

2) ViewBag:-

- ViewBag was not available in previous MVC
- It is introduced in MVC 3.0 because of "dynamic" data type
- ViewBag is a property of ControllerBase Class.
- It is defined in System.Web.MVC. It is of type dynamic
- Syntax: ViewBag.Name = value;
- ViewBag can directly return complex values on call back no need of typecast
- But ViewBag also available only for current request

- ViewBag only transfers data from controller to view, not vice versa. ViewBag values will be null if redirection occurs.

- ViewBag for Simple data -

Demo Controller which contains index method.

```
public ActionResult Index()
{
    ViewBag.Products = "pen";
    return View();
}
```

View Page in which we are trying to access simple data using ViewBag

```
<body>
  <div>
    @ViewBag.Products
  </div>
</body>
</html>
```

- ViewBag for Complex Values:-

DemoController which contains index method

```
Public ActionResult Index()
{
   ViewBag.Products = new List<string>()
   {
      "TV"
      "Laptop"
      "Mobile"
   };
   return View();
}
```

Complex data

View page in which we are trying to access complex data using ViewBag

```
<body>
   <div>
      <ol>
         @foreach (var item in ViewBag.Products)
         {
            <li>@item</li>
         }
      </ol>
   </div>
</body>
```

Without typecasting

DATE / / /

3) TempData :-

- TempData is used to transfer data from view to Controller, Controller to view, or from one action method to another action method of the same or a different controller.
- TempData stores the data temporarily and automatically removes it after retrieving a value.
- TempData only works during the current and subsequent request.

• TempData for Complex View:-

Demo Controller which contains index method

```
public ActionResult Index()
{
    TempData["Data"] = "pen";

    TempData["Products"] = new List<string>();
    {
        "pen";
        "pencil";
        "book".
    };
    Return View();
}
```

ViewPage in which we are trying to access complex data using TempData

```
<body>
  <div>
    @TempData["Data"]
  </div>
  <div>
    <ol>
      @foreach (var item in (List<string>)
                                TempData["Products"])
      {
        <li> @item </li>
      }
    </ol>
  </div>
</body>
```

Difference Betn ViewData, ViewBag, TempData

ViewData	ViewBag	TempData
① It is key-value Dictionary collection.	① It is a type object	① It is key-value Dictionary collection.
② ViewData is faster than ViewBag	② ViewBag is slower than ViewData	② NA
③ ViewData is a dictionary object and it is property of ControllerBase class	ViewBag is Dynamic property of ControllerBase class	TempData is dictionary object and it is property of controllerBase class

ViewData is introduced in MVC 1.0 and available in MVC 1.0 and above.	ViewBag is introduced in MVC 3.0 and available in MVC 3.0 and above.	TempData is also introduced in MVC 1.0 and available in MVC 1.0 and above
ViewData also works with .net framework 3.5 and above	ViewBag only works with .net framework 4.0 & above.	TempData also works with .net framework 3.5 & above.
Type Conversion code is required while enumerating	In depth, ViewBag is used dynamic. so there is no need to type Conversion while enumerating	Type conversion code is required while enumerating
It's value becomes null if redirection has occured	Same as ViewData.	TempData is used to pass data between two consecutive request
It lies only during the current request	same as ViewData	TempData only works during the current and subsequent reques

Integrate Model, View and Controller :-

Another way to pass data from controller to view.
Using @model property of model.

1) Create an MVC empty project with name ex DemoMVC.
2) Create model class and define properties to store a data ex. Emp.
3) Create another model class with name EmpBL in which we can initialize the class properties.

In Model
→ Emp.cs (filename)

```
public class Emp
{
    public int ID { get; set; }

    public string Name { get; set; }

    public string Address { get; set; }

    public string Email { get; set; }
}
```

→ EmpBL.cs. (Filename)

```
public class EmpBL
{
    public Emp funEmp()
    {
        Emp obj = new Emp()
        {
            ID = 1;
            Name = "Trupti";
            Address = "Pune";
            Email = "trupti@gmail.com".
        };
        return obj;
    }
}
```

In Controller

```
public ActionResult GetEmpDetails()
{
    EmpBL obj = new EmpBL();
    Emp e = obj.funEmp();
    return View(e);
}
```

In .cshtml

```
@Model intro_mvc_example.Models.Emp
@{
    ViewBag.Title = "Get Emp Details";
}
<h2> Get Emp Details </h2>
<div> class
```

In .cshtml

```
@Model intro_mvc_example.Models.Emp

<h2> Get Emp Details </h2>
<div class = "row">
  @Model.ID
  @Model.Name
  @Model.Address
  @Model.Email
</div>
```

Flow ⇒

Controller → Model → ~~View~~ Controller → View

Result ⇒

Get Emp Details

1 Trupti Pune trupti@gmail.com

HTML Helper class:-

- The HtmlHelper class renders HTML controls in the razor view. It binds the model object to HTML controls to display the value of model properties into those controls.
- @Html is an object of the HtmlHelper class. (@ symbol is used to access server side object in razor syntax). Html is a property of the HtmlHelper class included in base class of razor view WebViewPage
- The HtmlHelper class generates HTML elements. For example, @Html.ActionLink("Create New","Create") would generate anchor tag Create New

Html Helper class:-

There are many extension methods for HtmlHelper class, which creates different HTML controls.

Extension Method	Html control.

HTML helper class

Entity Framework.

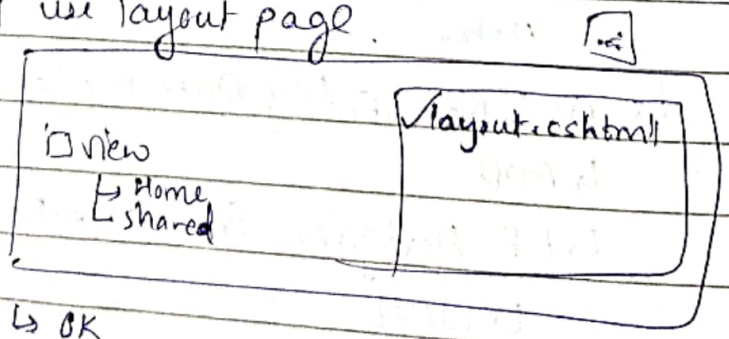
ASP.NET Web application (.NET Framework)
↳ create new project
↳ MVC
↳ Clean Solution.
↳ ~~Build~~ ReBuild solution

Tools
↳ Nuget Package manager
↳ Manage Nuget Package for solution.
↳ Browse
↳ search box "EntityFramework"
↳ Entity Framework by Microsoft.
↳ check ☑ project
↳ Install
↳ Save All
↳ Rebuild.

Right Click on Project
↳ Add
↳ New item
↳ Visual C#
Data
↳ ADO.NET Entity Data Model
↳ ADD.
↳ EF Designer from DataBase.
↳ Next
→

- Create
  - ↳ New Connection
    - ↳ Connection with database as we do.
      - ↳ Next
        - ↳ Select DataBase and table
          - ↳ Finish.
            - ↳ Save All. (edmx file)
              - ↳ Build
                - ↳ Rebuild.
                - ↳ Close (edmx file)

Controller
- ↳ Right click on that
  - ↳ Add
    - ↳ Controller
      - ↳ MVC5 controller with views, using entity framework
        - ↳ Add (window will pop up)
      - ↳ Data Context class
        - ↳ Select file from dropdown list
      - ↳ Model Class
        - ↳ tblEF (Project Name)
          (table name)
      - ↳ ☑ use layout page.



- ↳ OK
- ↳ Add
- ↳ Rebuild Solution
- ↳ SAVE ALL

↳ close controller.

↳ copy controller name

↳ in Route config file.

new {controller = "(copied controller name)"

↳ Rebuild solution

↳ Run