Chapter 6
Razor Views

MVC

Model View Controller is an architectural pattern that separates application code into three main categories Models, Views and Controllers. Any request comes that actually first reaches the controller action method and then the controller invokes the respective business model to perform required operation. Once operation is completed the result is sent back to controller and then controller further invokes view for displaying result.

Benefits of MVC

- Separation of concern
- Each components (MVC) developed independently
- Debugging is easy
- Unit testing each individual component is easier

View

View is web page (.cshtml) that is responsible for containing presentation logic that merges data along with static design code (HTML)

Razor code block

Razor code block is a C# code block that contains one or more lines of C# code that contain any statements and local functions

Razor Expression

@Expression or @(Expression) Razor expression is a C# expression accessing field property or method that returns a value.

Example

```
@using ViewsExample.Model

@{
    string appTitle = "ASP.Net Core App";
    string header = "Welcome to Home Page";

    Person p = new Person()
            {
                Id = 1, Name = "John", DOB =
Convert.ToDateTime("1989-11-24")
            };
}

<!Doctype html>
<html>
    <head>
```

```
        <title>@appTitle</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <h1>@header</h1>
        <h2>Hello @p.Name</h2>
        @{
          double age = Math.Round( (DateTime.Now - p.DOB).TotalDays /
365);
        }

        <h3>@age years old</h3>
    </body>
</html>
```

Razor If
When you want to show some content based on some condition then you can use Razor if or Razor if else
Syntax

```
@if(condition){
      c#/html code
}
```

```
 @if(condition){
      c#/html code
}
else
{
      c#/html code
}
```

Example

```
@using ViewsExample.Model

@{
    string appTitle = "ASP.Net Core App";
    string header = "Welcome to Home Page";

    Person p = new Person()
            {
                Id = 1, Name = "John", DOB = null
            };
}

<!Doctype html>
<html>
    <head>
```

```
        <title>@appTitle</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <h1>@header</h1>
        <h2>Hello @p.Name</h2>
    @if(p.DOB.HasValue){
          double age = Math.Round((DateTime.Now - p.DOB).Value.TotalDays 
/ 365);
        <h3>@age years old</h3>
    }
    else
    {
        <h3>Date of birth is unknown</h3>
    }
</body>
</html>
```

Razor Switch
It is same like switch case in C# when ever any case matches with condition we can display some html content
Syntax

@switch(variable){
case value1: C#/HTML code; break;

case value2: C#/HTML code; break;

default : C#/HTML code; break;

}

Example

```
@using ViewsExample.Model

@{
    string appTitle = "ASP.Net Core App";
    string header = "Welcome to Home Page";

    Person p = new Person()
            {
                Id = 1, Name = "John", DOB = null, PersonGender = 
Gender.Female
            };
}


<!Doctype html>
<html>
    <head>
        <title>@appTitle</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <h1>@header</h1>
        <h2>Hello @p.Name</h2>
    @if(p.DOB.HasValue){
          double age = Math.Round((DateTime.Now - p.DOB).Value.TotalDays
/ 365);
        <h3>@age years old</h3>
    }
    else
    {
        <h3>Date of birth is unknown</h3>
    }

    @switch(p.PersonGender)
    {
        case Gender.Male :
            <p>19 November International Men's Day</p>
            ;
            break;

        case Gender.Female:
            <p>8 March International Women's Day</p>
            ;
            break;
    }
</body>
</html>
```

Razor Foreach
Razor foreach is used to iterate through collection of and print or display element in form of HTML code
Syntax
@foreach(var variable in collection)
{
  C#/HTML code here
}

Razor - for
For loop is used when you want to display data only for some number of time like you have collection of 5 elements and you want to display only 2 elements then you can use for loop

@for(initialization,condition,increament)
{
C#/HTML Code
}

Example

```
@using ViewsExample.Model

@{
    string appTitle = "ASP.Net Core App";
    string header = "Welcome to Home Page";

    List<Person> people = new List<Person>()
    {
       new Person(){Name = "John", DOB = DateTime.Parse("2000-05-06"),
PersonGender = Gender.Male},
       new Person(){Name = "Radha", DOB = DateTime.Parse("2005-04-03"),
PersonGender = Gender.Female},
       new Person(){Name = "Abdul", DOB = DateTime.Parse("2000-12-20"),
PersonGender = Gender.Male},
       new Person(){Name = "Chaya", DOB = DateTime.Parse("1993-12-24"),
PersonGender = Gender.Female}
    };
}


<!Doctype html>
<html>
    <head>
        <title>@appTitle</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <h1>@header</h1>
        <h2>Hello </h2>
        @foreach(Person p in people)
        {
            <div>
                @p.Name
                @p.DOB
                @p.PersonGender
            </div>


        }
        <br />
        @for(int i = 0; i < 2; i++ )
        {
            Person person = people[i];
            <div>
            @person.Name
            @person.DOB
            @person.PersonGender
            </div>
        }
</body>
</html>
```

Literal Text
There are two ways through which you can print literal text in Views
<text></text> Inside text tag you can write your desired text
@: text after this considered are literal text

Razor - Local Functions
Local functions are callable methods that can accept some arguments and do some processing and return output. These local functions are accessible in the same view. For example when you have some repeated code in view then you can think of using them. You can use the local function inside the method, in the controller or in view as well. Scope of local function is limited to its containing method.
Example

```
@using ViewsExample.Model

@{
    string appTitle = "ASP.Net Core App";
    string header = "Welcome to Home Page";

    List<Person> people = new List<Person>()
    {
       new Person(){Name = "John", DOB = DateTime.Parse("2000-05-06"),
PersonGender = Gender.Male},
       new Person(){Name = "Radha", DOB = DateTime.Parse("2005-04-03"),
PersonGender = Gender.Female},
       new Person(){Name = "Abdul", DOB = DateTime.Parse("2000-12-20"),
PersonGender = Gender.Male},
       new Person(){Name = "Chaya", DOB = DateTime.Parse("1993-12-24"),
PersonGender = Gender.Female}
```

```
    };
}
@*local function example*@

@{
    double? GetAge(DateTime? dateofBirth)
    {
        if(dateofBirth is not null)
        {
            return Math.Round((DateTime.Now - 
dateofBirth.Value).TotalDays / 365);
        }
        else
        {
            return null;
        }
    }
}
<!Doctype html>
<html>
    <head>
        <title>@appTitle</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <h1>@header</h1>
        <h2>Hello </h2>
        @foreach(Person p in people)
        {
            <div>
                @p.Name
                @GetAge(@p.DOB)
                @p.PersonGender
            </div>
        }
        <br />
       </body>
</html>
```

Html.Raw()
If you want to use some html or javascript code as raw then you can use this by using HTMLRaw you can able to execute javascript code else razor page will directly print code whatever you have written as javascript

Example

```
@using ViewsExample.Model

@{
    string alertmesage = "<script>alert('Hi from javascript')</script>";
}

<!Doctype html>
<html>
    <head>
        <title>@appTitle</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        @Html.Raw(alertmesage);
    </body>
</html>
```

ViewData
View Data is a dictionary object which is used to send data from controller to view. View data object automatically created upon receiving data from controller and deleted before sending response to client.

Example
Controller

```
using Microsoft.AspNetCore.Mvc;
using ViewsExample.Model;

namespace ViewsExample.Controllers
{
    public class HomeController : Controller
    {
        [Route("")]
        public IActionResult Index()
        {
            ViewData["appTitle"] = "ASP.Net Core App";
            ViewData["header"] = "Welcome to Home Page";
            List<Person> people = new List<Person>()
            {
                new Person(){Name = "John", DOB =
DateTime.Parse("2000-05-06"), PersonGender = Gender.Male},
                new Person(){Name = "Radha", DOB =
DateTime.Parse("2005-04-03"), PersonGender = Gender.Female},
                 new Person(){Name = "Abdul", DOB =
DateTime.Parse("2000-12-20"), PersonGender = Gender.Male},
                 new Person(){Name = "Chaya", DOB =
DateTime.Parse("1993-12-24"), PersonGender = Gender.Female}
            };
            ViewData["People"] = people;
            return View();
        }
    }
}
```

View

```
@using ViewsExample.Model
@{
    double? GetAge(DateTime? dateofBirth)
    {
        if(dateofBirth is not null)
        {
            return Math.Round((DateTime.Now -
dateofBirth.Value).TotalDays / 365);
        }
        else
        {
            return null;
        }
    }
}
<!Doctype html>
<html>
    <head>
        <title>@ViewData["appTitle"]</title>
        <meta charset="utf-8"/>
    </head>
    <body>
        <h1>@ViewData["header"]</h1>
        <h2>Hello </h2>
        @{
            List<Person>? people = (List<Person>?)ViewData["people"];
        }
        @if(people is not null)
        {
            @foreach(Person p in people)
            {
                <div>
                    @p.Name
```

```
                     @GetAge(@p.DOB)
                     @p.PersonGender
                 </div>
             }
         }
         else
         {
             <p>No user found</p>
         }
</body>
</html>
```

ViewBag
View bag is a property of controller and view that is used to access the view data easily (without type casting). It is not separate from viewdata but it internally uses viewdata to retrieve data. Also it is best practice to use type of object rather than using var keyword in for each loop as type of variable is decided at runtime so while writing code it does not show any intelices for properties of some class for example as well as we need to handle null exception while using view bag.

Example
View

```
@using ViewsExample.Model
@{
    double? GetAge(DateTime? dateofBirth)
    {
        if(dateofBirth is not null)
        {
            return Math.Round((DateTime.Now -
dateofBirth.Value).TotalDays / 365);
        }
        else
        {
            return null;
        }
    }
}
<!Doctype html>
<html>
    <head>
        <title>@ViewBag.appTitle</title>
        <meta charset="utf-8"/>
        <link href="~/StyleSheet.css" rel="stylesheet">
    </head>
    <body>
        <div class="page-content">
            <h1>@ViewBag.header</h1>
            @foreach(Person p in ViewBag.people)
                {
                    <div class="box float-left w-50">
                    <h3>@p.Name</h3>
                            <table class="table w-100">
                                <tbody>
                                    <tr>
                                        <td>Gender</td>
                                        <td>@p.PersonGender</td>
                                    </tr>
                                    <tr>
                                        <td>Age</td>
                                        <td>@GetAge(@p.DOB)</td>
                                    </tr>
                                    <tr>
                                        <td>Date Of Birth</td>
<td>@p.DOB.Value.ToString("MM/dd/yyyy")</td>
                                    </tr>
                                </tbody>
                            </table>
                    </div>
                }
        </div>
</body>
</html>
```

Strongly Typed View
Strongly typed view is a view that is bound to a specific model class. It is mainly used to access the model/object collection easily in the view. To specify the model you need to write @model ObjectName like that you need to mention. While passing data from controller to view you need to pass objects in view while returning.
Ex return View(objectName);
Controller

```
using Microsoft.AspNetCore.Mvc;
using ViewsExample.Model;

namespace ViewsExample.Controllers
{
    public class HomeController : Controller
    {
        [Route("")]
        public IActionResult Index()
        {
            ViewData["appTitle"] = "ASP.Net Core App";
            ViewData["header"] = "Welcome to Home Page";
            List<Person> people = new List<Person>()
            {
                new Person(){Name = "John", DOB =
DateTime.Parse("2000-05-06"), PersonGender = Gender.Male},
                new Person(){Name = "Radha", DOB =
DateTime.Parse("2005-04-03"), PersonGender = Gender.Female},
                new Person(){Name = "Abdul", DOB =
DateTime.Parse("2000-12-20"), PersonGender = Gender.Male},
                new Person(){Name = "Chaya", DOB =
DateTime.Parse("1993-12-24"), PersonGender = Gender.Female}
            };
            return View("Index", people);
        }
    }
}
```

View

```
@using ViewsExample.Model
@model IEnumerable<Person>

@{
    double? GetAge(DateTime? dateofBirth)
    {
        if(dateofBirth is not null)
        {
            return Math.Round((DateTime.Now -
dateofBirth.Value).TotalDays / 365);
        }
        else
        {
            return null;
        }
    }
}
<!Doctype html>
<html>
    <head>
        <title>@ViewBag.appTitle</title>
        <meta charset="utf-8"/>
```

```
         <link href="~/StyleSheet.css" rel="stylesheet">
     </head>
     <body>
         <div class="page-content">
             <h1>@ViewBag.header</h1>

             @foreach(Person p in Model)
                 {
                     <div class="box float-left w-50">
                     <h3>@p.Name</h3>
                             <table class="table w-100">
                                 <tbody>
                                     <tr>
                                         <td>Gender</td>
                                         <td>@p.PersonGender</td>
                                     </tr>
                                     <tr>
                                         <td>Age</td>
                                         <td>@GetAge(@p.DOB)</td>
                                     </tr>
                                     <tr>
                                         <td>Date Of Birth</td>
 <td>@p.DOB.Value.ToString("MM/dd/yyyy")</td>
                                     </tr>
                                 </tbody>
                             </table>
                     </div>
                 }
         </div>
 </body>
 </html>
```

Benefits of Strongly Typed View

- You will get intellisense while accessing model class properties
- You will have only one model per view
- Property name checked at compile time
- Easy to identify which model is being accessed in the view

ViewMode

View model is also called a wrapper model that is used to combine multiple entity classes and display results into strongly typed views. Strongly typed view is assigned with one model class only to bind multiple model classes to a strongly typed view. We can use ViewModel. It is nothing but creating one class which is considered as view model class add other class as property into it and bind it with a view that's it.

_ViewImports.cshtml
_ViewImports.cshtml is a special file in the Views folder or its subfolder which executes automatically before execution of a view. It is mainly used to import common namespaces that are imported in a view.

SharedViews
In a real world scenario we come across a situation where we want to have the same UI components across multiple controllers and view that part we can keep it in the shared view folder for example menu, footer etc will come under SharedView.

Questions

1. What is the MVC pattern?
2. Explain the role of the various components of the MVC pattern?
3. Explain the differences between ViewData and ViewBag
4. Explain strongly-typed views.