STARTING DOTNET CORE

- -Open Source and Cross platform development
- -Latest version is ASP.NET Core 7.0 released on FEB-24-2023-
- -.NET was primarily released in 2000 as .NET 1.0
- -In 2016 MS came up with .NET Core 1.0 which allowed cross-platform development (that was not available in previous normal .NET framework).

-FEATURES:

1. Cross Platform and open Source

2. Fast Development

3. .NET Core now also works on other editors such as VScode etc

4. Based on MVC architecture.

-WHAT IS ASP.NET CORE..?

1- New version of ASP.NET

2- Open-Source FW for cross platform development

3- It's rewritten from scratch, initially launched as ASP.NET-5 and then renamed to ASP.NET-CORE-1.0

4- It's cross platform, high-performance

5- It has modular components (i.e everything is separated into modules) that gives flexibility.

-DIFFERENCE B/W .NET & .NET CORE FRAMEWORK-

1- .NET FW is old |.NET Core is new

2- Not open source | Open Source

3- Only For Windows | Cross Platform

4- Needs framework to be installed on windows | doesn't requires intallation

5- .NET dosen't support dependency injection | Allowed dependency injection.

6- Used to build WinForms,WPF for developing desktop and webapps

| Dosen't support desktop application development but focuses on web,windows mobile with ASP.NET and windows universal apps

7- Has single packaged installation and runtime enviroment for windows

|Needs to be installed independently

8- No support for Microservices | Supports Microservices

9- Lower performance | High scalability and performance

10- Dosent support Mobile Development | supports Mobile Development (Compatible with Xamarin)

11- Too heavy for CLI | Lightweight CLI (there's always an option of IDE in both)

12- All libraries are pack and shipped together | Shipped as Nugget packages

---

-DIFFERENCE B/W ASP.NET & ASP.NET CORE-

- ASP.NET was the first version of web-adapted .NET FW

|ASP.NET CORE is an improved version and continuation of ASP.NET

- ASP.NET CORE is opensource & crossplatform
- BOTH ARE BASED ON MVC ARCHITECTURE
- ASP.NET CORE has cloud and microservices support and it provides modularity(seperate files for evrything)
- ASP.NET was launched in 2002 based on .NET FW
- ASP.NET CORE is new standard version and it works on both .NET and .NET CORE FW
- ASP.NET CORE has no support for WebForms.
- ASP.NET CORE has no support for global.asx and web.config but appsetting.json is integrated.
- ASP.NET CORE requires VS-2015 or newer versions.

-MVC IN ASP.NET CORE 6-

-MVC is an architectural design pattern and not a programming language or framework.-

- Model : Business entities(table classes)

- View : UI and presentation logics.

- Controller : Business logics or bridge b/w Model and View.

-MIDDLEWARE-

- ASP.NET CORE introduced middleware,it's nothing but a component which is executed on every request in application

- It defines and control how application responds to HTTP requests

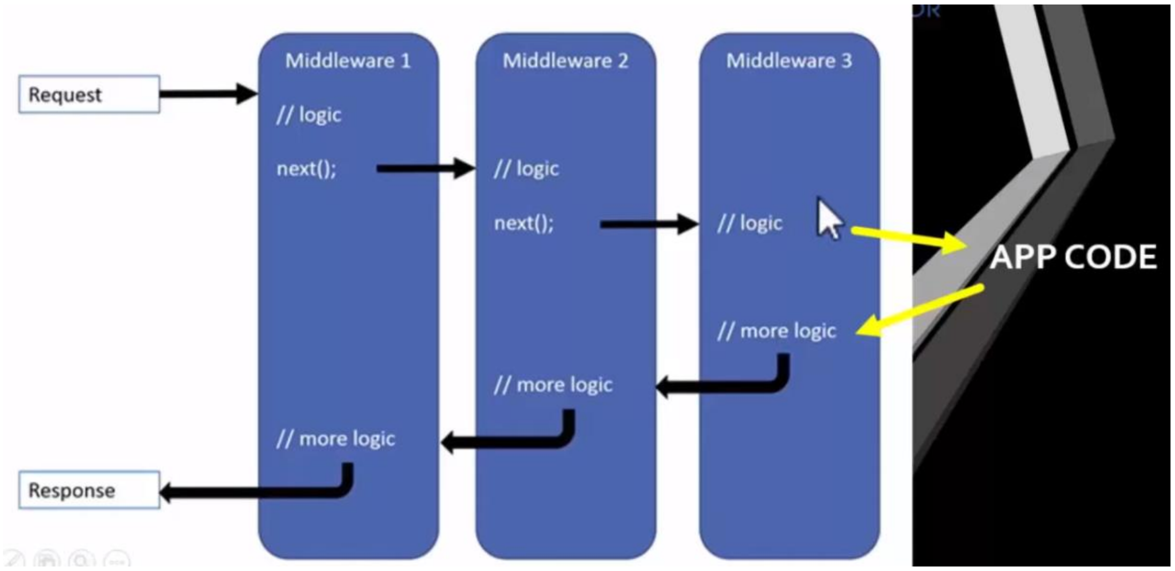
- Executed in program.cs file.

- Authentication and Authorization is also done here.

- 2 TYPES of METHODS to run a middleware

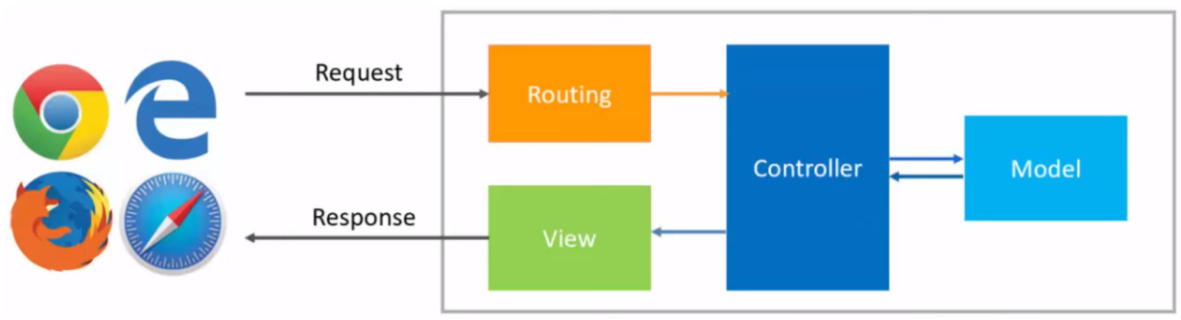
1- RUN() middleware defined using app.Run will never call next middleware and It only requires 1 paramerter (i.e Context). Server is started on app.run() method.

2- next() middleware Allows middleware to pass control to next item(middleware) in the pipeline and it needs 2 parameters (i.e Context and Next)



-ROUTING-

- Routing is a mechanism that checks the request from user and map that to Action method and controller in the Application.
- The Routing in ASP.NET Core MVC application is a mechanism in which it will inspect the incoming Requests (i.e. URLs) and then mapped that request to the controllers and their action methods.



-2 TYPES OF ROUTING-

-CONVENTION-BASED ROUTING

In Conventional Based Routing, the route is determined based on the conventions defined

in the route templates which will map the incoming Requests (i.e. URLs) to controllers and their action methods.

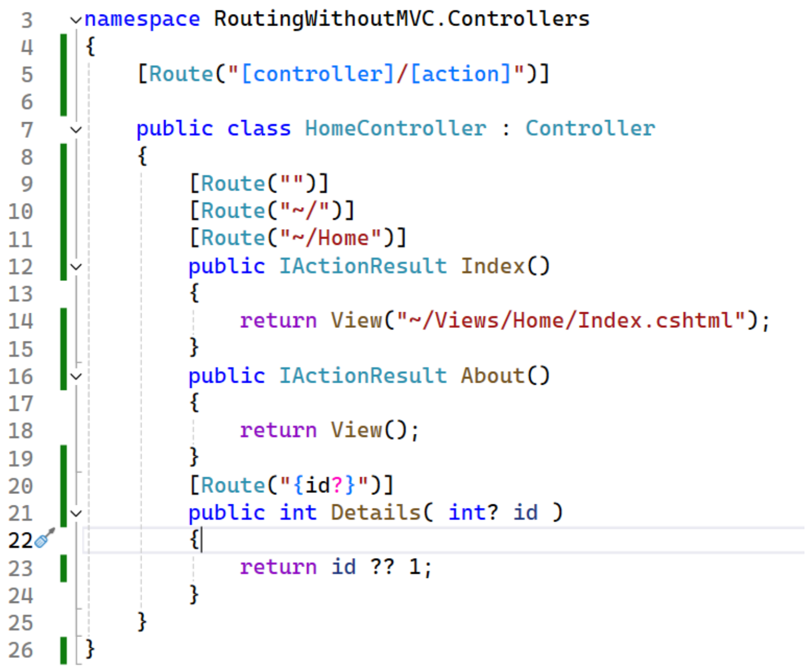
{controller=Home}/{action=Index}/{id?}

--ATTRIBUTE-BASED ROUTING

we define routes for each action

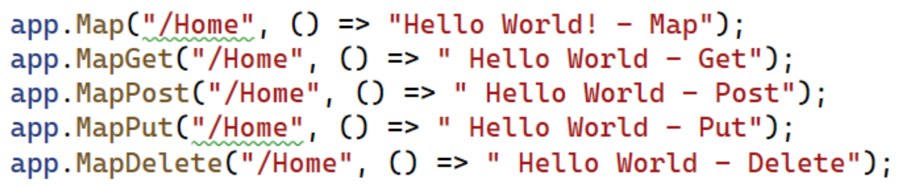
method above it with attribute

like this

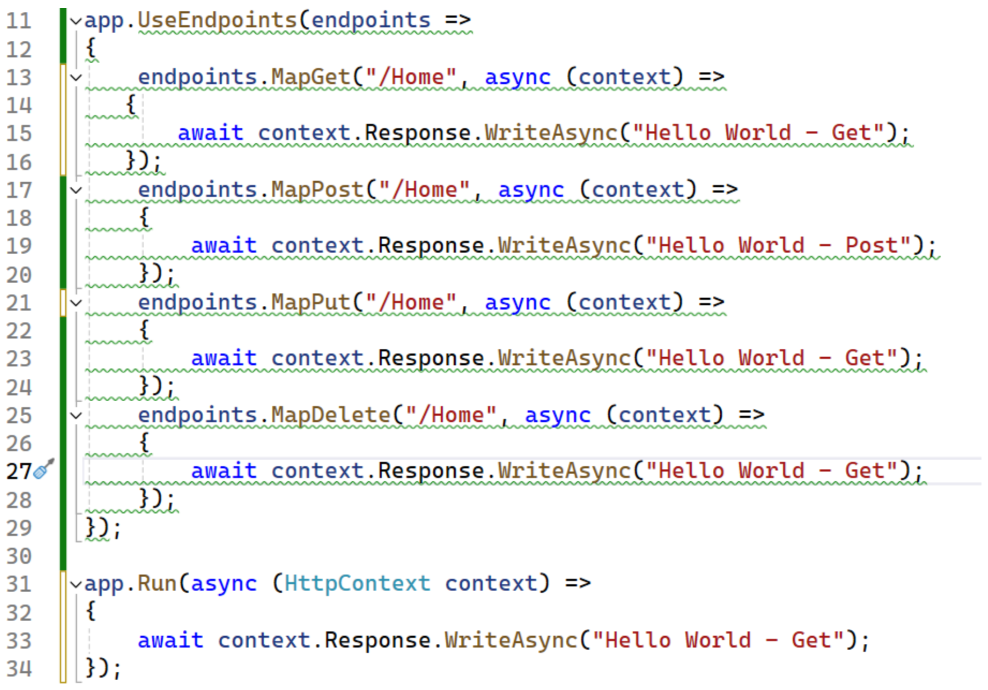


```
namespace RoutingWithoutMVC.Controllers
{
    [Route("[controller]/[action]")]

    public class HomeController : Controller
    {
        [Route("")]
        [Route("~/")]
        [Route("~/Home")]
        public IActionResult Index()
        {
            return View("~/Views/Home/Index.cshtml");
        }
        public IActionResult About()
        {
            return View();
        }
        [Route("{id?}")]
        public int Details( int? id )
        {
            return id ?? 1;
        }
    }
}
```

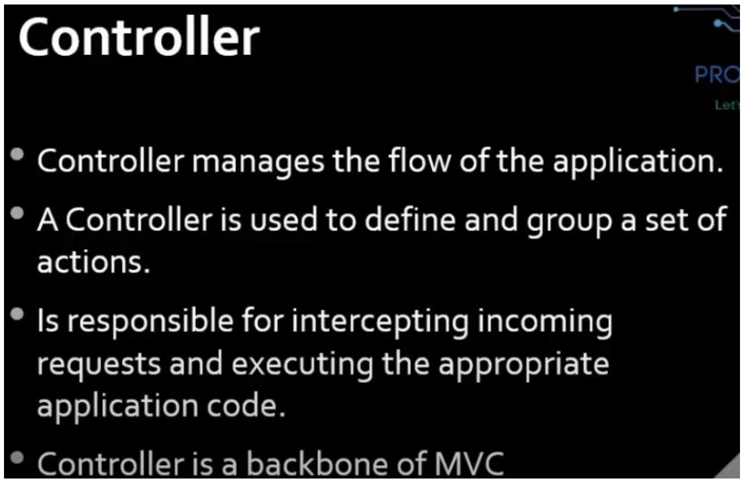
Map methods (Get, Put, Post, Delete)



```
app.Map("/Home", () => "Hello World! - Map");
app.MapGet("/Home", () => " Hello World - Get");
app.MapPost("/Home", () => " Hello World - Post");
app.MapPut("/Home", () => " Hello World - Put");
app.MapDelete("/Home", () => " Hello World - Delete");
```



```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/Home", async (context) =>
    {
        await context.Response.WriteAsync("Hello World - Get");
    });
    endpoints.MapPost("/Home", async (context) =>
    {
        await context.Response.WriteAsync("Hello World - Post");
    });
    endpoints.MapPut("/Home", async (context) =>
    {
        await context.Response.WriteAsync("Hello World - Get");
    });
    endpoints.MapDelete("/Home", async (context) =>
    {
        await context.Response.WriteAsync("Hello World - Get");
    });
});

app.Run(async (HttpContext context) =>
{
    await context.Response.WriteAsync("Hello World - Get");
});
```

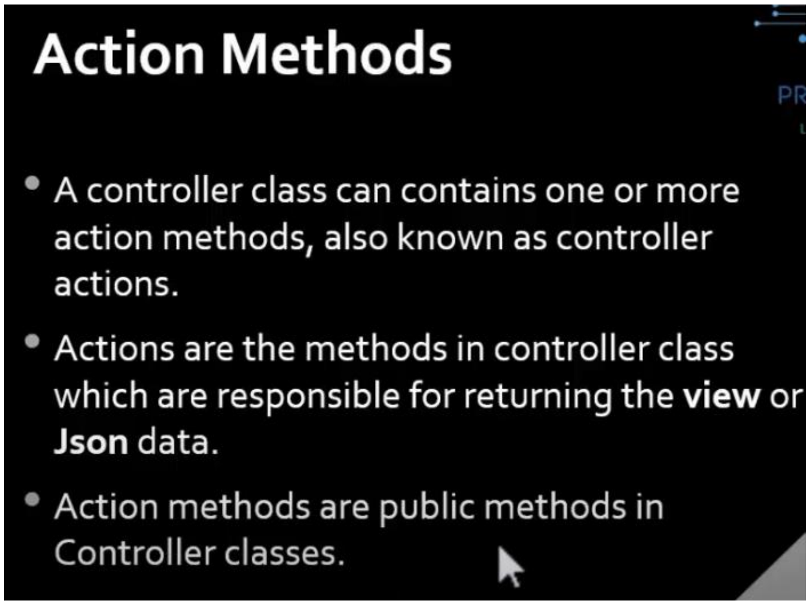


Controller

- Controller manages the flow of the application.
- A Controller is used to define and group a set of actions.
- Is responsible for intercepting incoming requests and executing the appropriate application code.
- Controller is a backbone of MVC

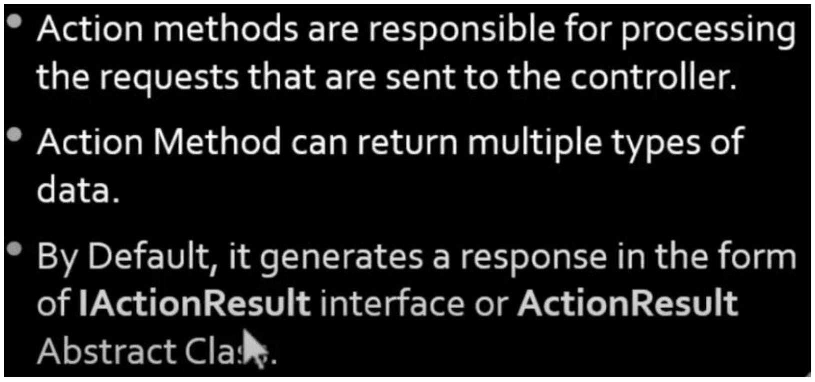
EndPoint

The MapGet method is used to define an endpoint. An endpoint is something that can be Selected, by matching the URL and HTTP method. Executed, by running the delegate.



Action Methods

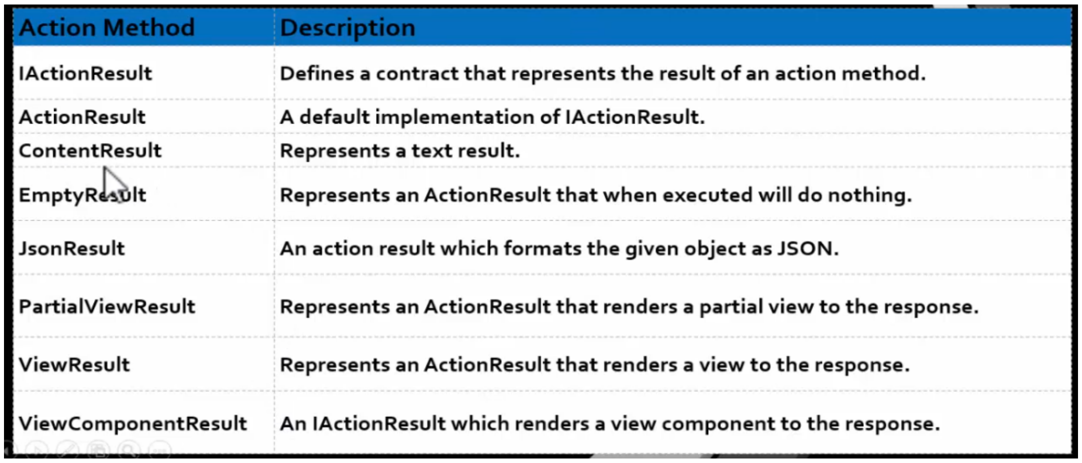
- A controller class can contains one or more action methods, also known as controller actions.
- Actions are the methods in controller class which are responsible for returning the **view** or **Json** data.
- Action methods are public methods in Controller classes.



- Action methods are responsible for processing the requests that are sent to the controller.
- Action Method can return multiple types of data.
- By Default, it generates a response in the form of **IActionResult** interface or **ActionResult** Abstract Class.

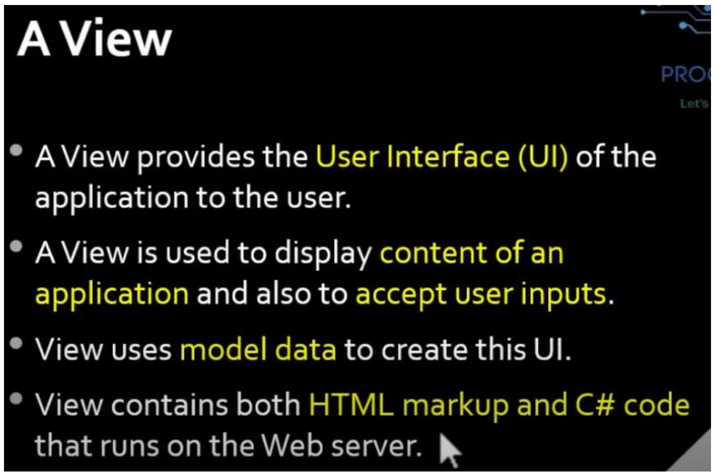
***Important Point*:** IActionResult is implemented by many classes like ViewResult, PartialViewResult, JsonResult etc

In that IActionResult is an interface and ActionResult is an abstract class that other action results inherit from.



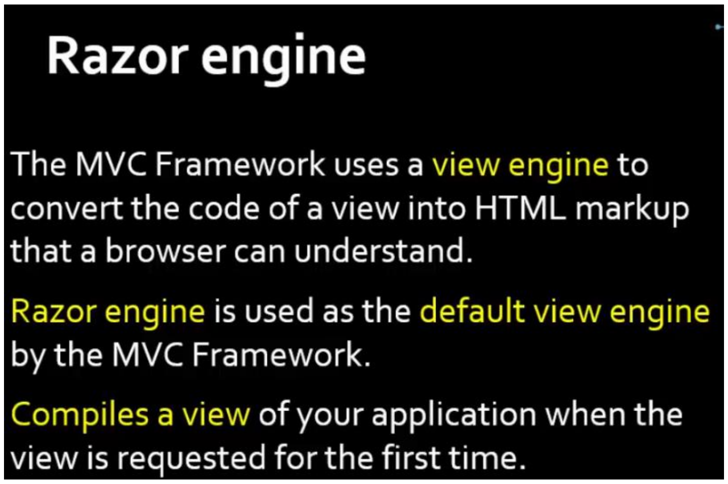
Action Method	Description
IActionResult	Defines a contract that represents the result of an action method.
ActionResult	A default implementation of IActionResult.
ContentResult	Represents a text result.
EmptyResult	Represents an ActionResult that when executed will do nothing.
JsonResult	An action result which formats the given object as JSON.
PartialViewResult	Represents an ActionResult that renders a partial view to the response.
ViewResult	Represents an ActionResult that renders a view to the response.
ViewComponentResult	An IActionResult which renders a view component to the response.

Watch video 13 for all action method implementations.



A View

- A View provides the User Interface (UI) of the application to the user.
- A View is used to display content of an application and also to accept user inputs.
- View uses model data to create this UI.
- View contains both HTML markup and C# code that runs on the Web server.

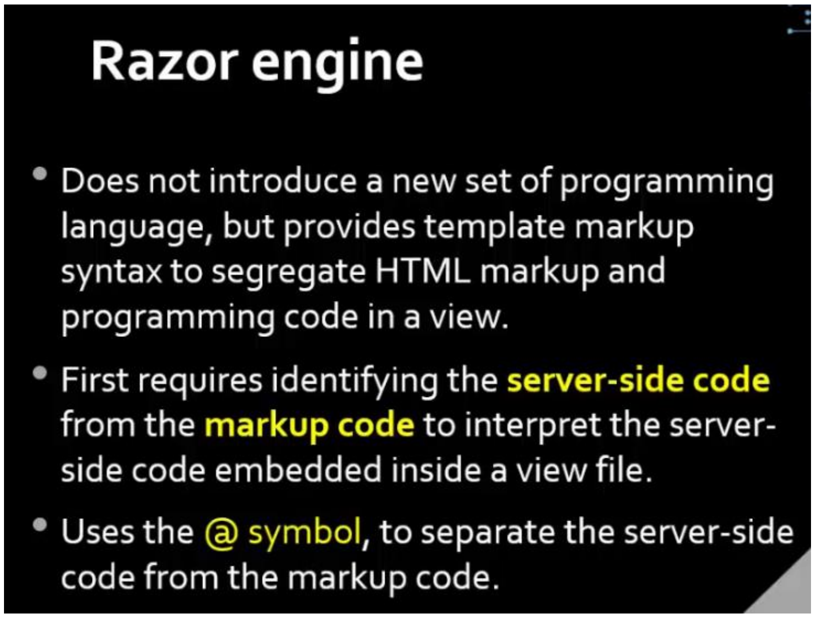


Razor engine

The MVC Framework uses a view engine to convert the code of a view into HTML markup that a browser can understand.

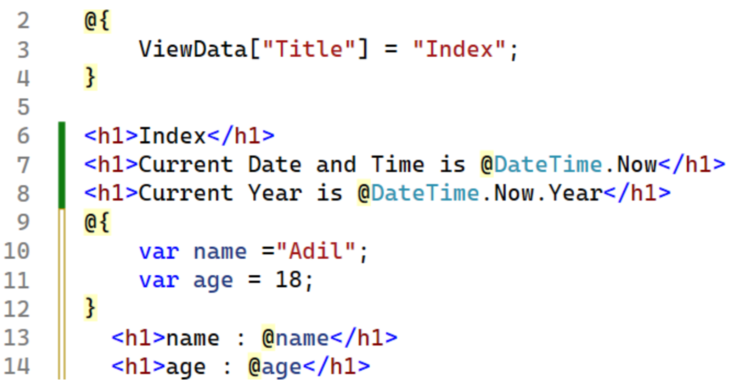
Razor engine is used as the default view engine by the MVC Framework.

Compiles a view of your application when the view is requested for the first time.

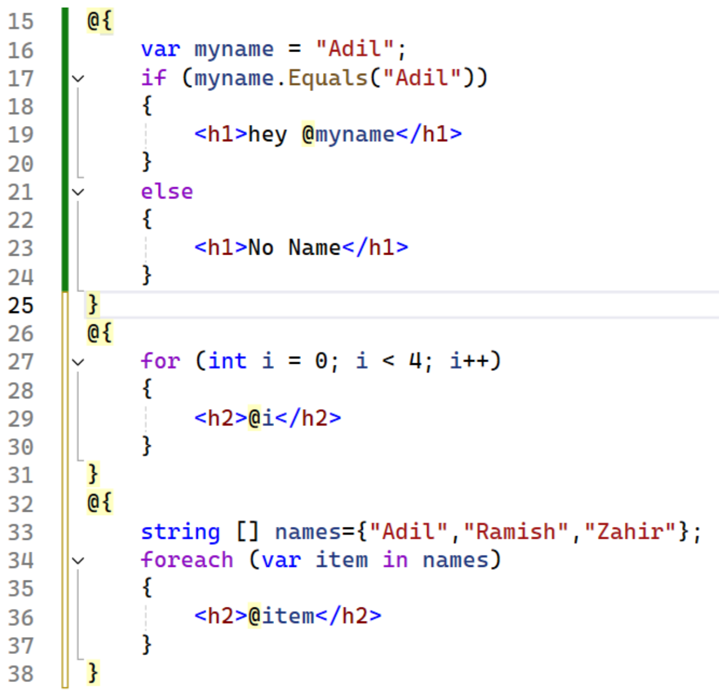


Razor engine

- Does not introduce a new set of programming language, but provides template markup syntax to segregate HTML markup and programming code in a view.
- First requires identifying the **server-side code** from the **markup code** to interpret the server-side code embedded inside a view file.
- Uses the @ symbol, to separate the server-side code from the markup code.

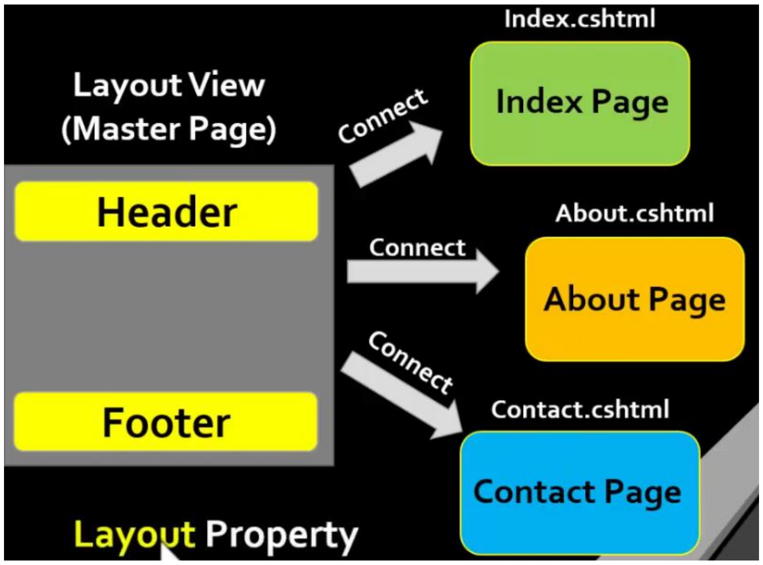


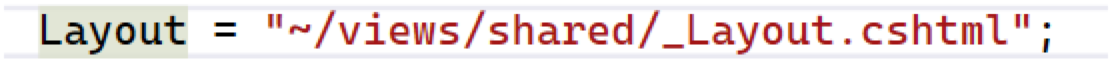
```
@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>
<h1>Current Date and Time is @DateTime.Now</h1>
<h1>Current Year is @DateTime.Now.Year</h1>
@{
    var name ="Adil";
    var age = 18;
}
  <h1>name : @name</h1>
  <h1>age : @age</h1>
```



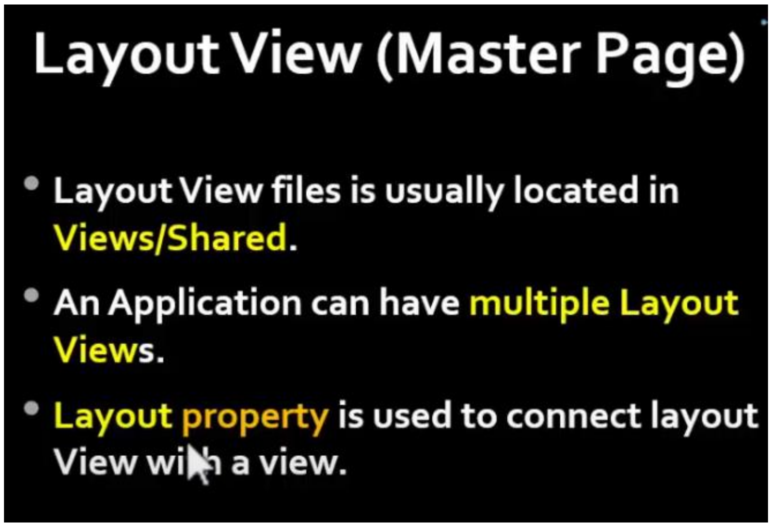
```
@{
    var myname = "Adil";
    if (myname.Equals("Adil"))
    {
        <h1>hey @myname</h1>
    }
    else
    {
        <h1>No Name</h1>
    }
}
@{
    for (int i = 0; i < 4; i++)
    {
        <h2>@i</h2>
    }
}
@{
    string [] names={"Adil","Ramish","Zahir"};
    foreach (var item in names)
    {
        <h2>@item</h2>
    }
}
```

Layout View



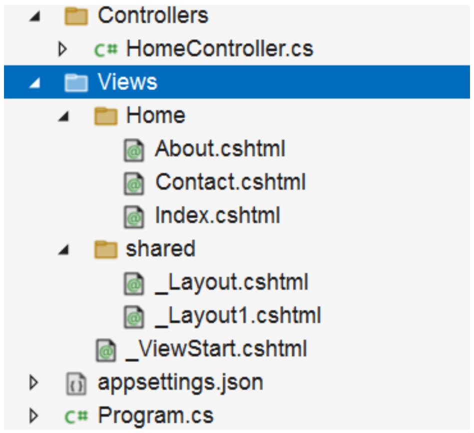


```
Layout = "~/views/shared/_Layout.cshtml";
```



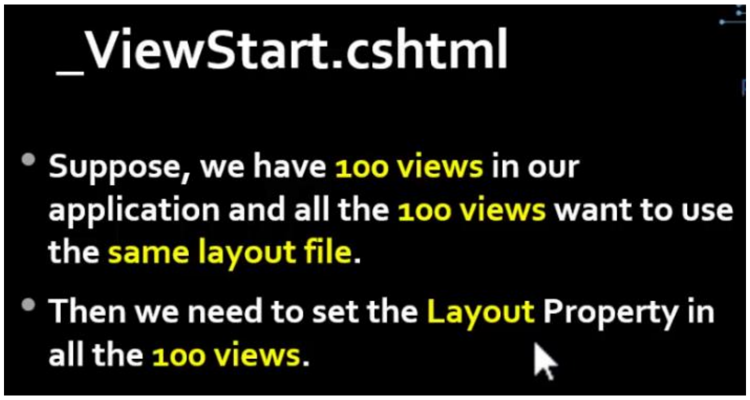
Layout View (Master Page)

- Layout View files is usually located in Views/Shared.
- An Application can have multiple Layout Views.
- Layout property is used to connect layout View with a view.



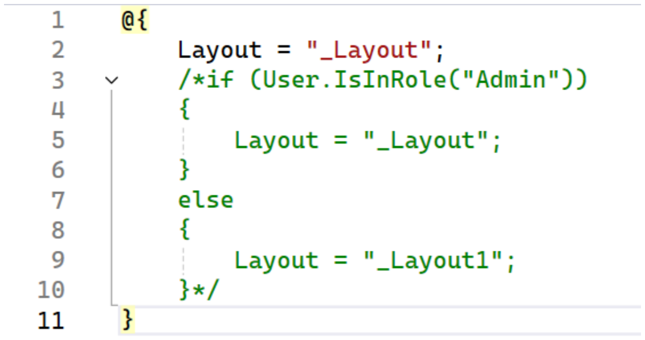
```
Controllers
    HomeController.cs
Views
    Home
        About.cshtml
        Contact.cshtml
        Index.cshtml
    shared
        _Layout.cshtml
        _Layout1.cshtml
    _ViewStart.cshtml
appsettings.json
Program.cs
```

View Start

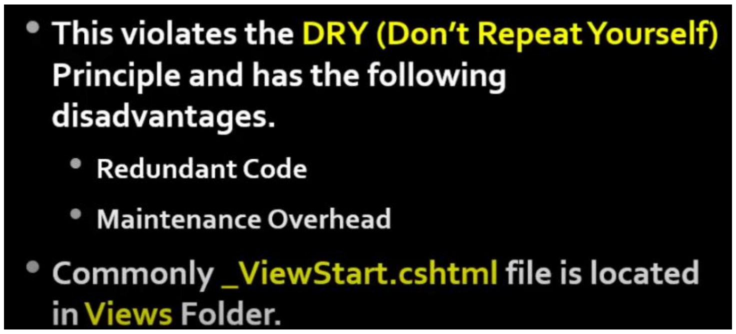


_ViewStart.cshtml

- Suppose, we have 100 views in our application and all the 100 views want to use the same layout file.
- Then we need to set the Layout Property in all the 100 views.

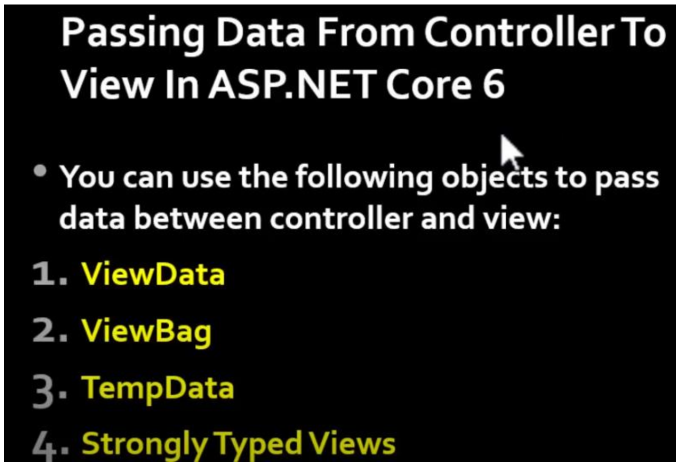


```
@{
    Layout = "_Layout";
    /*if (User.IsInRole("Admin"))
    {
        Layout = "_Layout";
    }
    else
    {
        Layout = "_Layout1";
    }*/
}
```



- This violates the DRY (Don't Repeat Yourself) Principle and has the following disadvantages.
    - Redundant Code
    - Maintenance Overhead
- Commonly _ViewStart.cshtml file is located in Views Folder.

Pass Data between view and controller

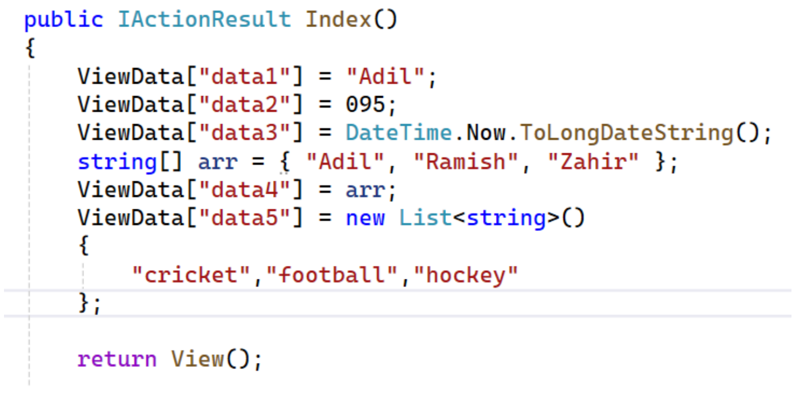


Passing Data From Controller To View In ASP.NET Core 6

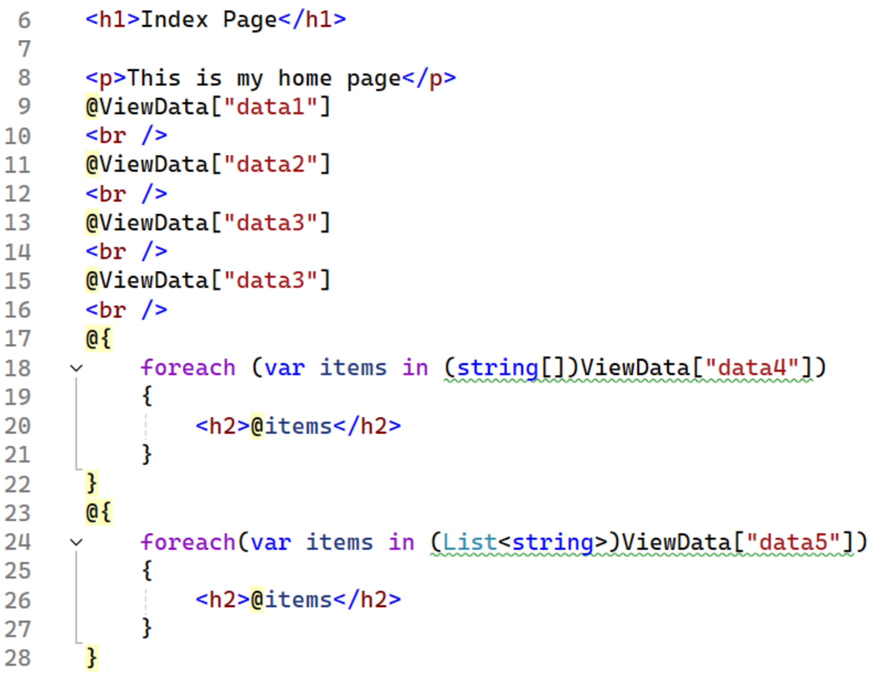
- You can use the following objects to pass data between controller and view:

1. ViewData
2. ViewBag
3. TempData
4. Strongly Typed Views

ViewData

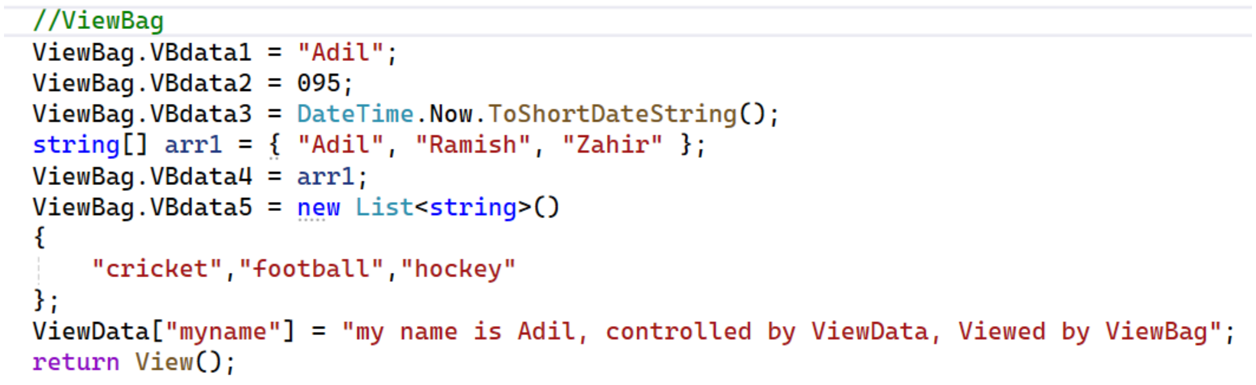


```
public IActionResult Index()
{
    ViewData["data1"] = "Adil";
    ViewData["data2"] = 095;
    ViewData["data3"] = DateTime.Now.ToLongDateString();
    string[] arr = { "Adil", "Ramish", "Zahir" };
    ViewData["data4"] = arr;
    ViewData["data5"] = new List<string>()
    {
        "cricket","football","hockey"
    };

    return View();
```

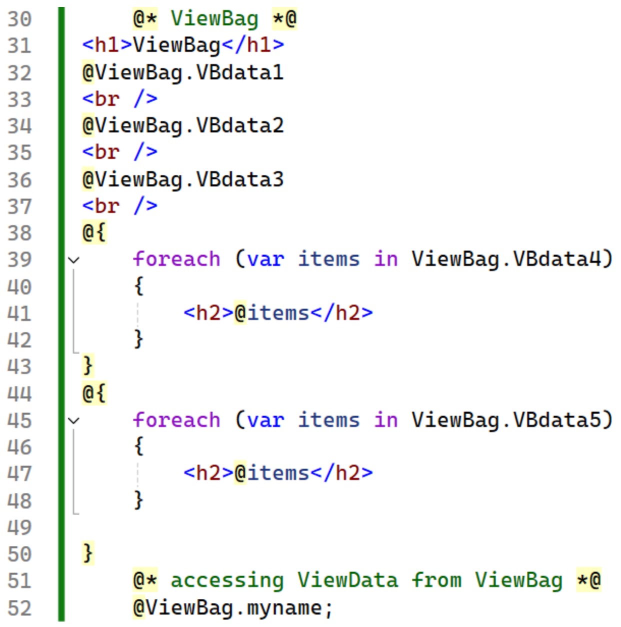


```
<h1>Index Page</h1>

<p>This is my home page</p>
@ViewData["data1"]
<br />
@ViewData["data2"]
<br />
@ViewData["data3"]
<br />
@ViewData["data3"]
<br />
@{
    foreach (var items in (string[])ViewData["data4"])
    {
        <h2>@items</h2>
    }
}
@{
    foreach(var items in (List<string>)ViewData["data5"])
    {
        <h2>@items</h2>
    }
}
```

ViewBag

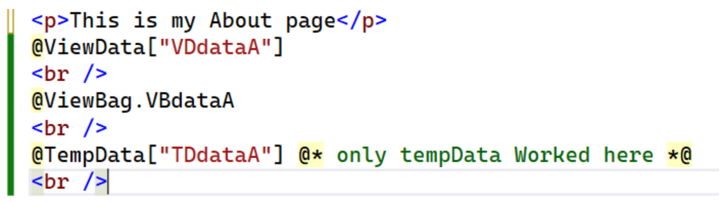


```
//ViewBag
ViewBag.VBdata1 = "Adil";
ViewBag.VBdata2 = 095;
ViewBag.VBdata3 = DateTime.Now.ToShortDateString();
string[] arr1 = { "Adil", "Ramish", "Zahir" };
ViewBag.VBdata4 = arr1;
ViewBag.VBdata5 = new List<string>()
{
    "cricket","football","hockey"
};
ViewData["myname"] = "my name is Adil, controlled by ViewData, Viewed by ViewBag";
return View();
```

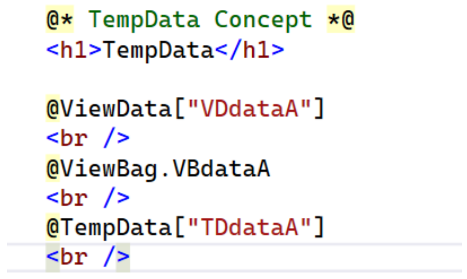


```
    @* ViewBag *@
<h1>ViewBag</h1>
@ViewBag.VBdata1
<br />
@ViewBag.VBdata2
<br />
@ViewBag.VBdata3
<br />
@{
    foreach (var items in ViewBag.VBdata4)
    {
        <h2>@items</h2>
    }
}
@{
    foreach (var items in ViewBag.VBdata5)
    {
        <h2>@items</h2>
    }

}
    @* accessing ViewData from ViewBag *@
    @ViewBag.myname;
```

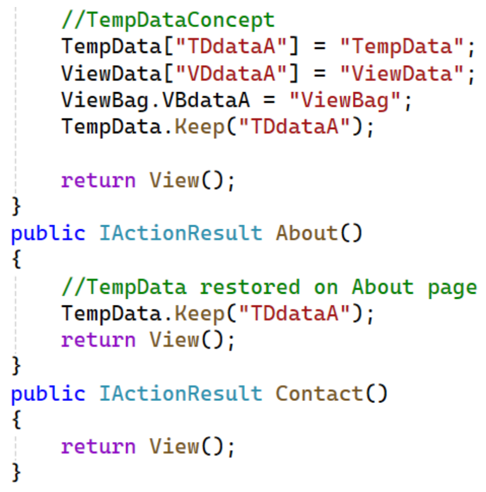
TempData



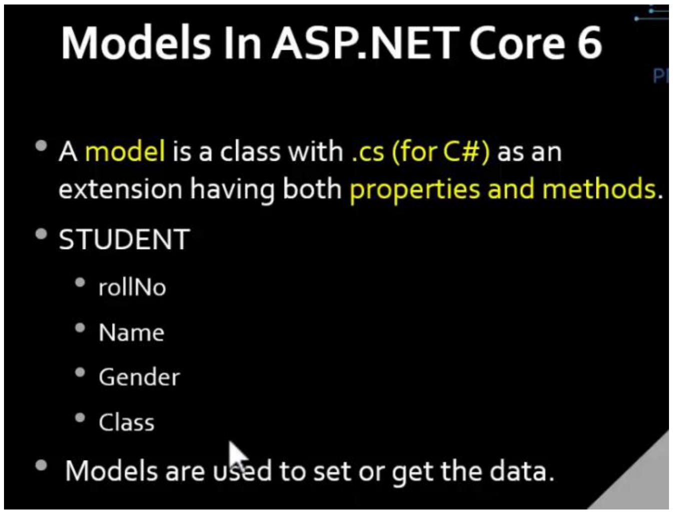
```
<p>This is my About page</p>
@ViewData["VDdataA"]
<br />
@ViewBag.VBdataA
<br />
@TempData["TDdataA"] @* only tempData Worked here *@
<br />
```



```
@* TempData Concept *@
<h1>TempData</h1>

@ViewData["VDdataA"]
<br />
@ViewBag.VBdataA
<br />
@TempData["TDdataA"]
<br />
```



```
    //TempDataConcept
    TempData["TDdataA"] = "TempData";
    ViewData["VDdataA"] = "ViewData";
    ViewBag.VBdataA = "ViewBag";
    TempData.Keep("TDdataA");

    return View();
}
public IActionResult About()
{
    //TempData restored on About page
    TempData.Keep("TDdataA");
    return View();
}
public IActionResult Contact()
{
    return View();
}
```



Models In ASP.NET Core 6

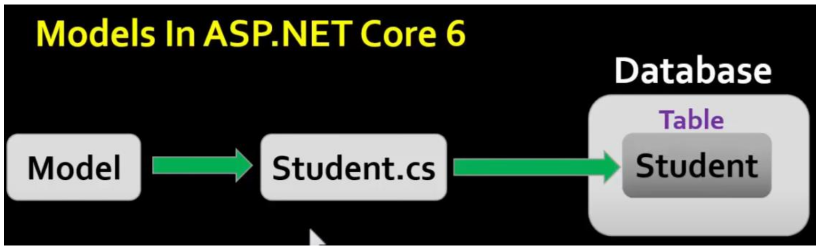
- A model is a class with .cs (for C#) as an extension having both properties and methods.
- STUDENT
  - rollNo
  - Name
  - Gender
  - Class
- Models are used to set or get the data.

```
    public class StudentModel
    {
        public int RollNo { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public int Standard { get; set; }

    }
}
```

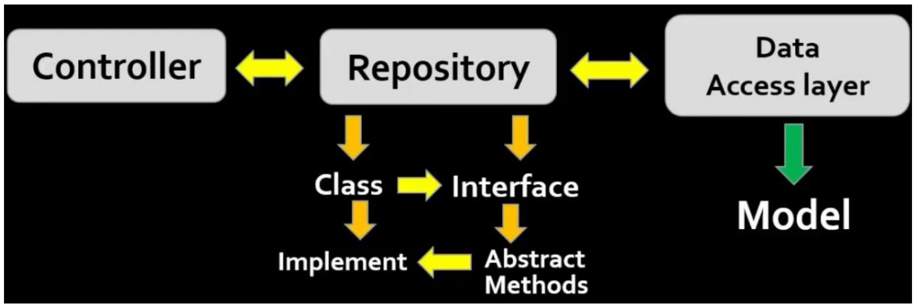
```
public IActionResult Index()
{
    var student = new List<StudentModel>() {
    new StudentModel{RollNo = 1,Name = "Ahmed", Gender = "Male", Standard = 10},
    new StudentModel{RollNo = 2,Name = "Ali", Gender = "Male", Standard = 9},
    new StudentModel{RollNo = 3,Name = "Alina", Gender = "Female", Standard = 8}
  };
  ViewData["MyStudents"]=student;
  return View();
}
```

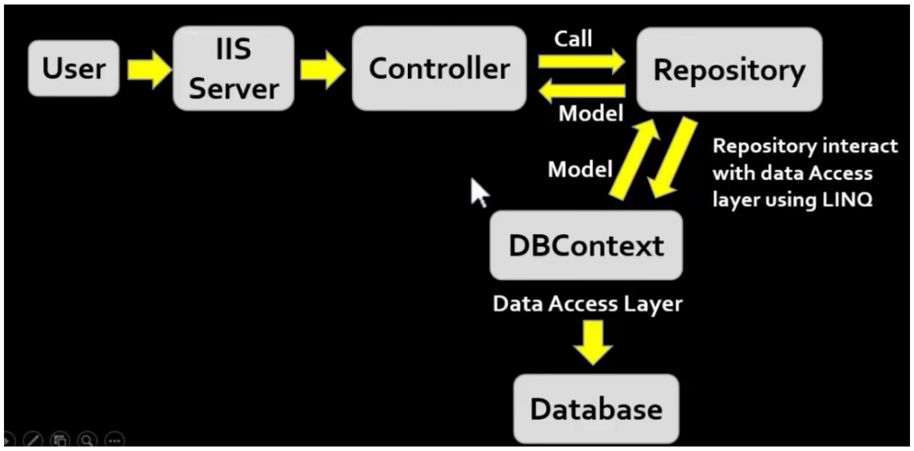
```
<table>
    <tr>
        <td>Roll Number</td>
        <td>Name</td>
        <td>Gender</td>
        <td>Class</td>
        @foreach (var item in students)
        {
            <tr>
            <td>@item.RollNo</td>
            <td>@item.Name</td>
            <td>@item.Gender</td>
            <td>@item.Standard</td>
            </tr>
        }
    </tr>
</table>
```

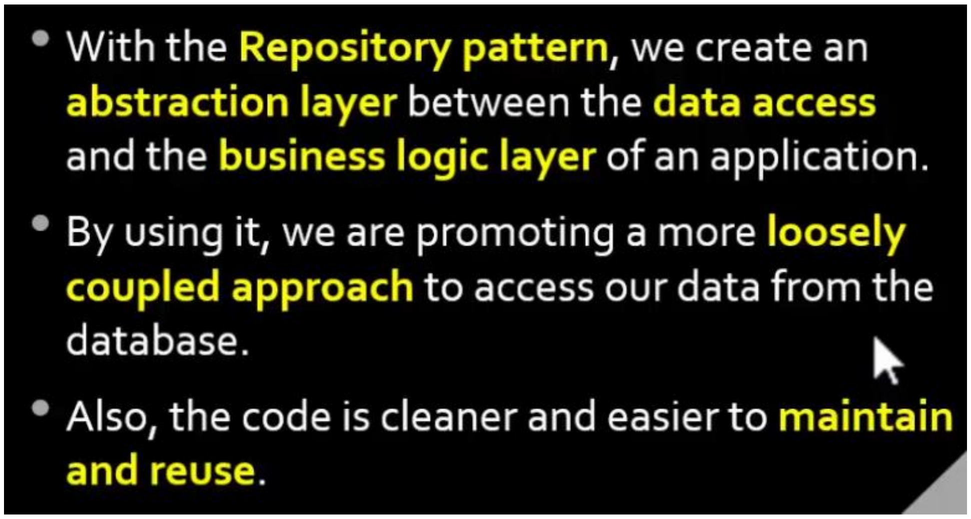
Repository Pattern



For every database table, there is a separate model class

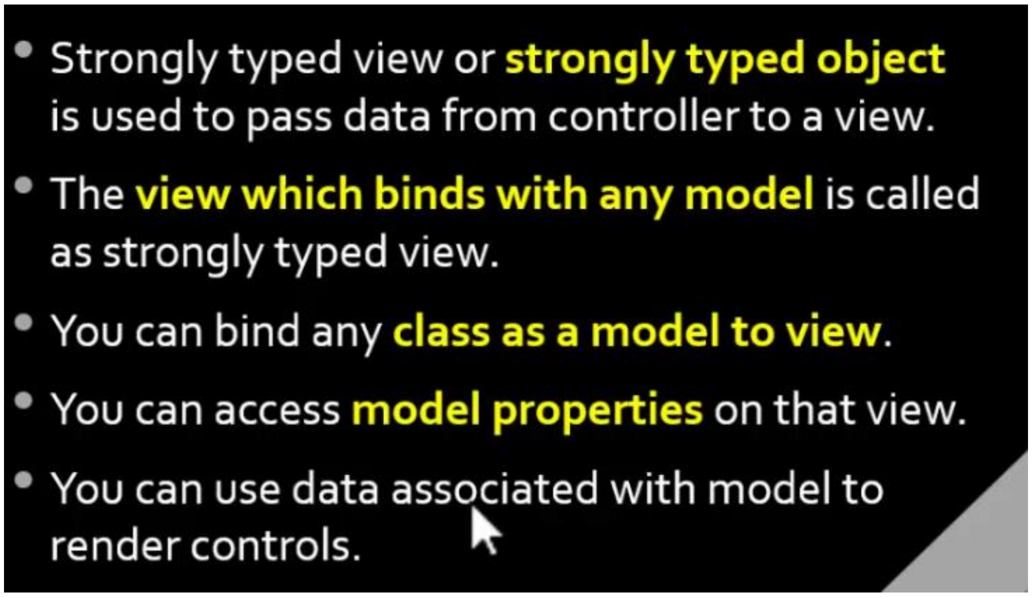




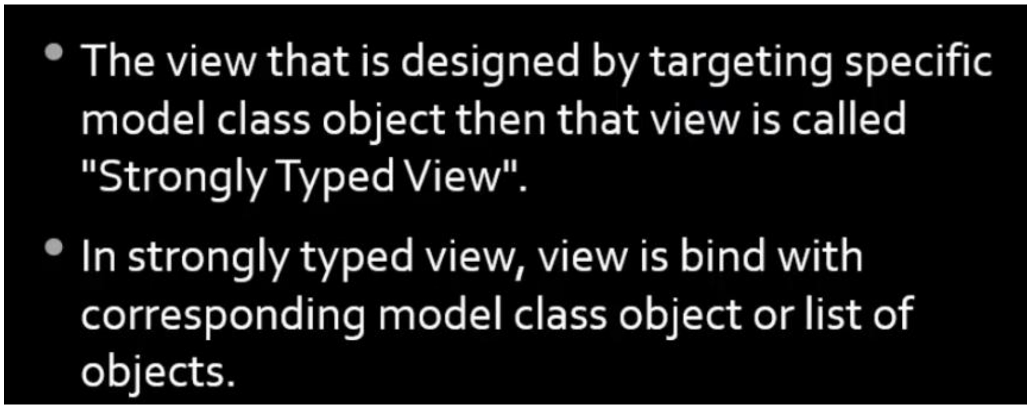


- With the **Repository pattern**, we create an **abstraction layer** between the **data access** and the **business logic layer** of an application.
- By using it, we are promoting a more **loosely coupled approach** to access our data from the database.
- Also, the code is cleaner and easier to **maintain and reuse**.

Strongly Typed View



- Strongly typed view or **strongly typed object** is used to pass data from controller to a view.
- The **view which binds with any model** is called as strongly typed view.
- You can bind any **class as a model to view**.
- You can access **model properties** on that view.
- You can use data associated with model to render controls.



- The view that is designed by targeting specific model class object then that view is called "Strongly Typed View".
- In strongly typed view, view is bind with corresponding model class object or list of objects.

-INSTALLING BOOTSTRAP/JQUERY etc-

- All static files like BS,css,js,jquery are placed inside WWWROOT folder

- Images are also placed here

- click on WWWroot and click on add client side library and type twitter-bootstrap and install.k

- app.UseStaticFiles(): without usiing this method we cannot access any of our static files

- Libman.json file contains all the configuration for clients side libraries

---

--ENITTY FRAMEWORK CORE IN ASP.NET CORE--

- ORM Object Relational Mapping FW

- Redesigned version of EF 6 from scratch

-

--- WRITE THIS COMMAND TO GENERATE CONNECTION ---

install these Nuget packages first

1- Microsoft.EntityFrameworkCore.SqlServer

2- Microsoft.EntityFrameworkCore.Tools

3- Microsoft.EntityFrameworkCore.Design

{

Scaffold-DbContext "server=servername;database'MyDb;trusted_connection=true"Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

}

Scaffold-DbContext "server=PC1-3\SQLEXPRESS; database=hexashop; trusted_connection=true"Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

---

---IF ITS GIVING AN SSL CERTIFICATE WARNING USE THIS STRING ---

Scaffold-DbContext "Server=PC1-3\SQLEXPRESS;Database=hexashop;Trusted_Connection=True;TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

----UPDATE----

(add -force flag at the end of string)

Scaffold-DbContext "server=PC1-3\SQLEXPRESS; database=hexashop; trusted_connection=true"Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -force

--shift the connection string from context file to appsettings.json--

```
 "ConnectionStrings": {
   "dbcs": "PASTE HERE"
 },
```

--register Connection string in program.cs--

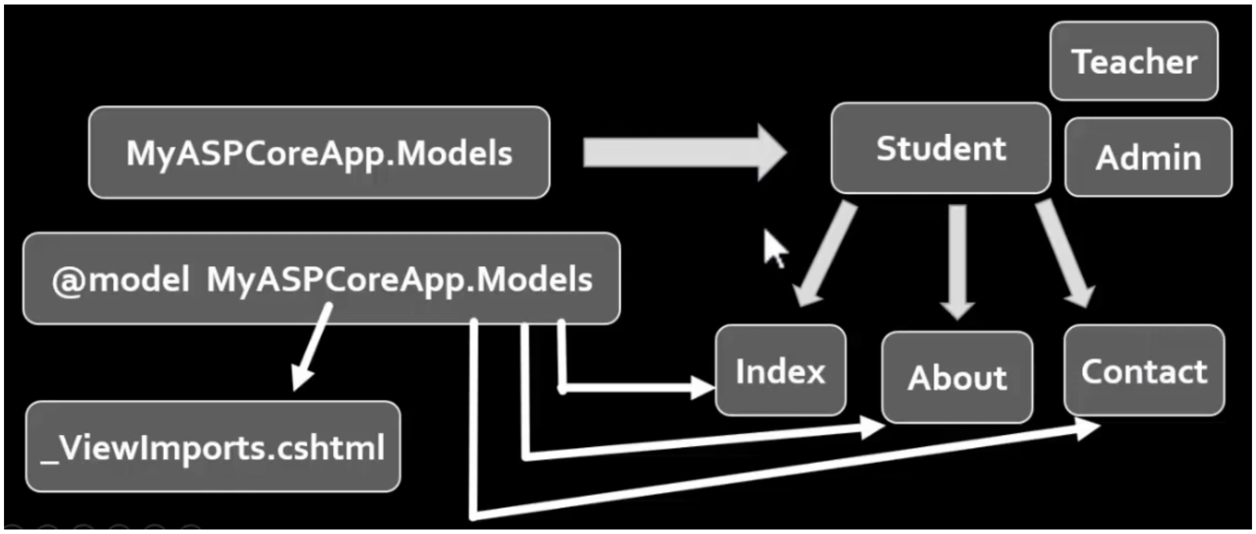
add these in program.cs after builder.Services.AddControllersWithViews();

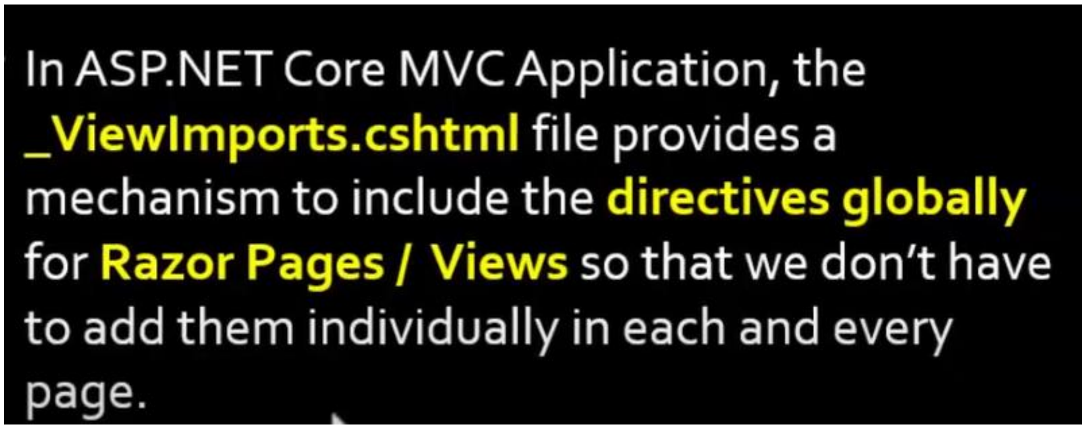
--

```
        var provider = builder.Services.BuildServiceProvider();
        var config = provider.GetService<IConfiguration>();
        builder.Services.AddDbContext<ArtevolContext>(item =>
item.UseSqlServer(config.GetConnectionString("dbcs")));
```

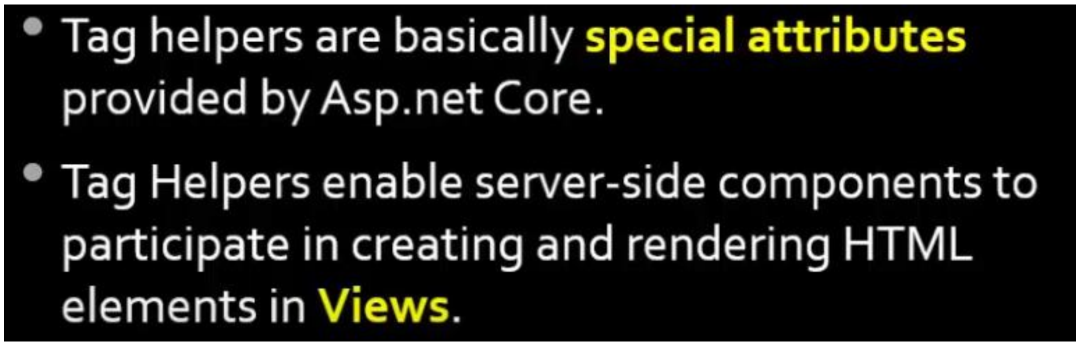
--

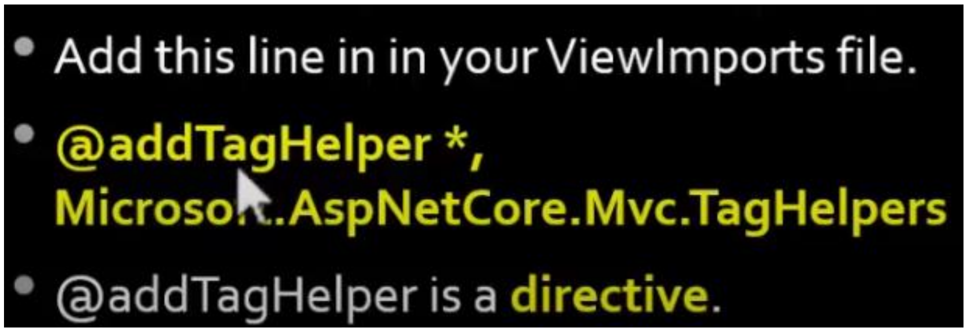
ViewImports

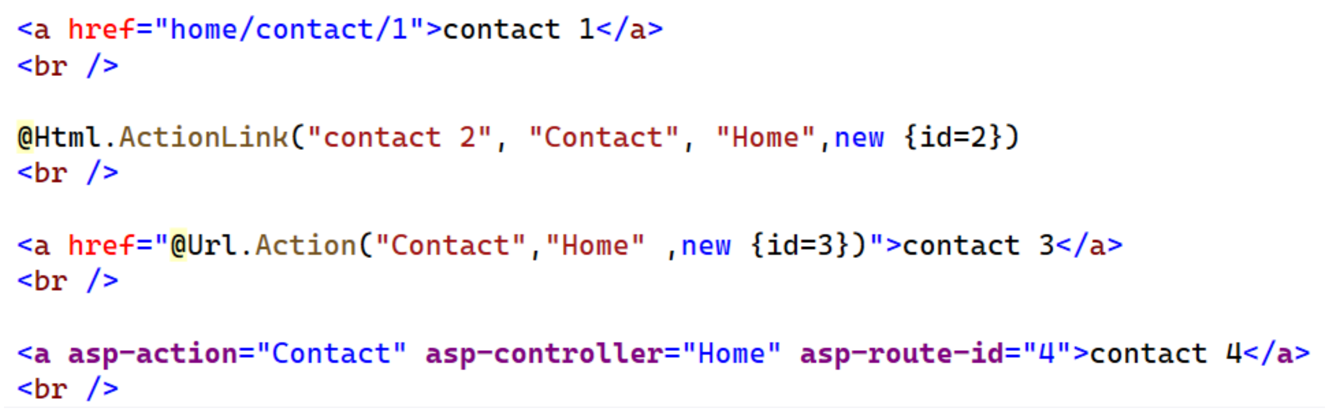



In ASP.NET Core MVC Application, the _ViewImports.cshtml file provides a mechanism to include the directives globally for Razor Pages / Views so that we don't have to add them individually in each and every page.

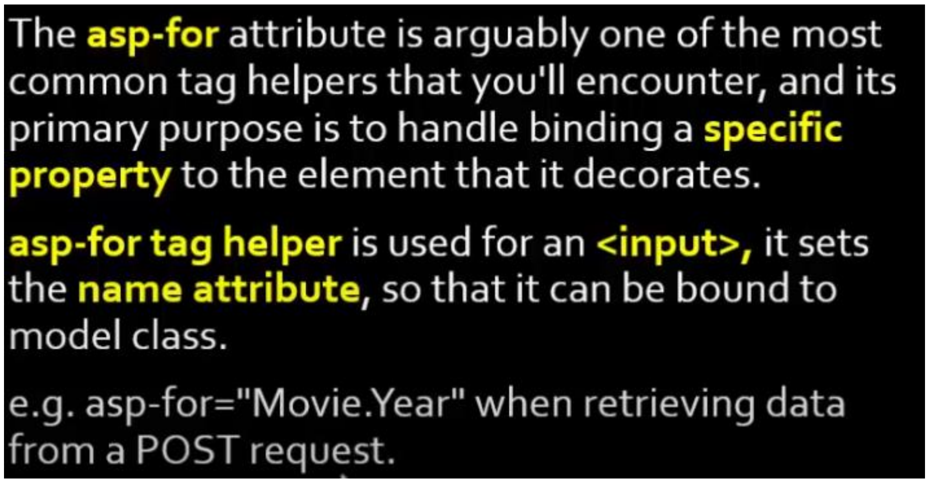
Tag Helpers


- Tag helpers are basically special attributes provided by Asp.net Core.
- Tag Helpers enable server-side components to participate in creating and rendering HTML elements in Views.


- Add this line in in your ViewImports file.
- @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
- @addTagHelper is a directive.

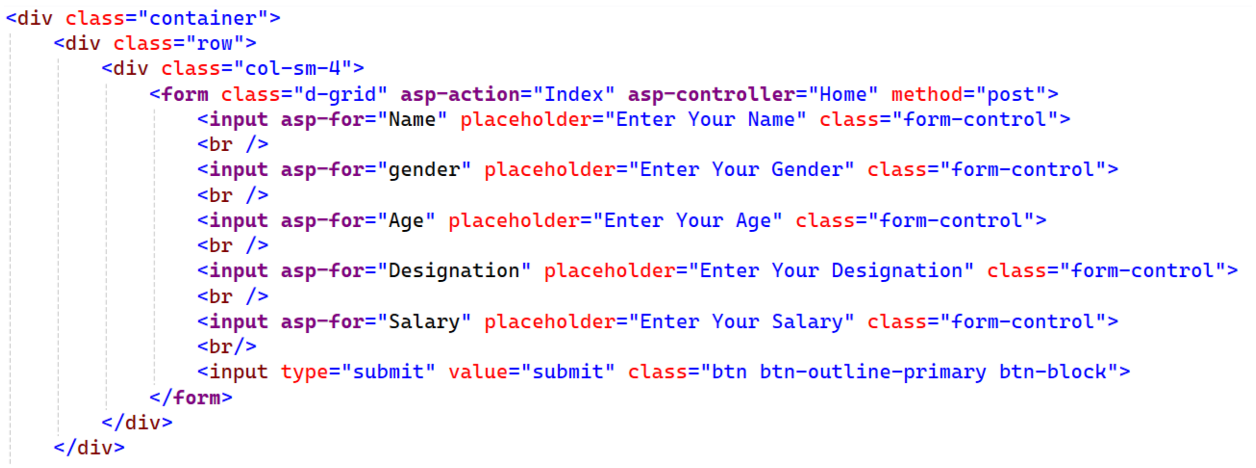

```
<a href="home/contact/1">contact 1</a>
<br />

@Html.ActionLink("contact 2", "Contact", "Home",new {id=2})
<br />

<a href="@Url.Action("Contact","Home" ,new {id=3})">contact 3</a>
<br />

<a asp-action="Contact" asp-controller="Home" asp-route-id="4">contact 4</a>
<br />
```

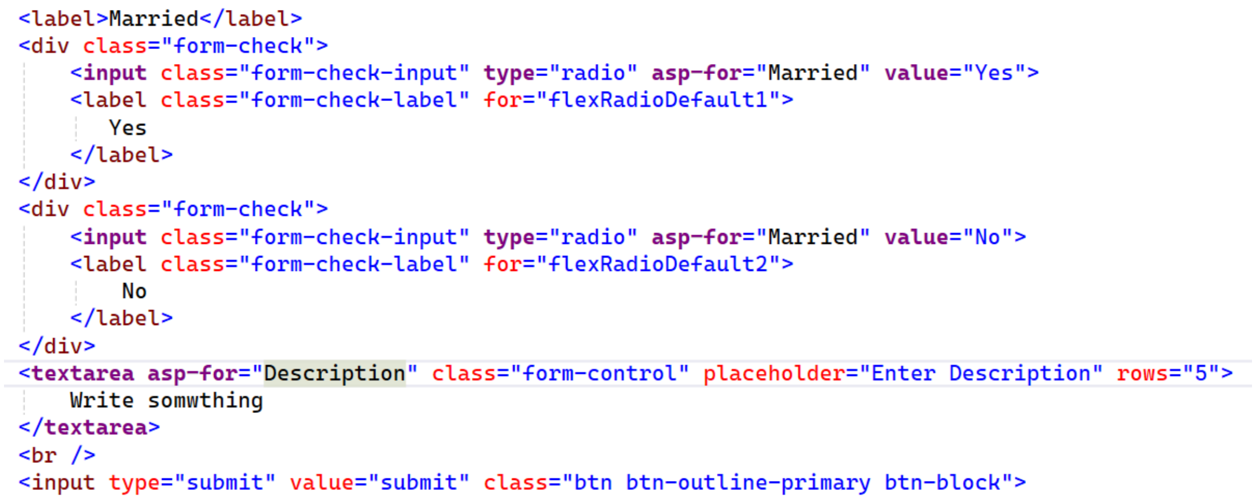
asp-for Tag

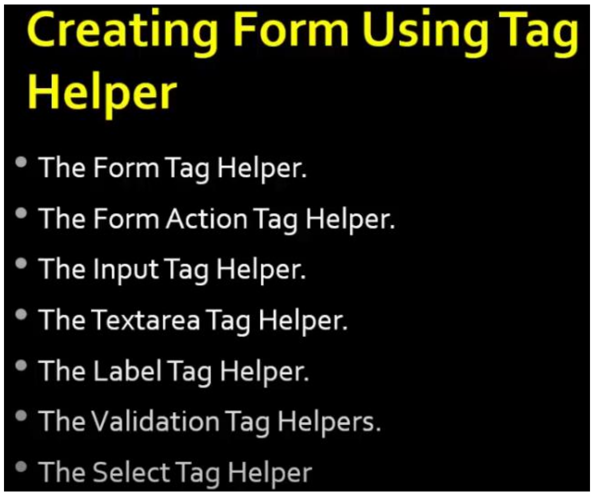

The **asp-for** attribute is arguably one of the most common tag helpers that you'll encounter, and its primary purpose is to handle binding a **specific property** to the element that it decorates.

asp-for tag helper is used for an **<input>,** it sets the **name attribute**, so that it can be bound to model class.

e.g. asp-for="Movie.Year" when retrieving data from a POST request.


```
<div class="container">
    <div class="row">
        <div class="col-sm-4">
            <form class="d-grid" asp-action="Index" asp-controller="Home" method="post">
                <input asp-for="Name" placeholder="Enter Your Name" class="form-control">
                <br />
                <input asp-for="gender" placeholder="Enter Your Gender" class="form-control">
                <br />
                <input asp-for="Age" placeholder="Enter Your Age" class="form-control">
                <br />
                <input asp-for="Designation" placeholder="Enter Your Designation" class="form-control">
                <br />
                <input asp-for="Salary" placeholder="Enter Your Salary" class="form-control">
                <br/>
                <input type="submit" value="submit" class="btn btn-outline-primary btn-block">
            </form>
        </div>
    </div>
```

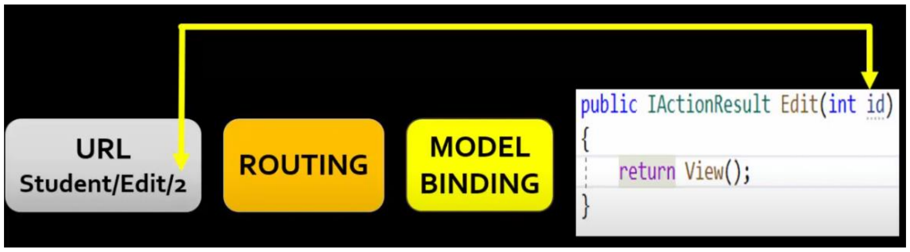

```
<label>Married</label>
<div class="form-check">
    <input class="form-check-input" type="radio" asp-for="Married" value="Yes">
    <label class="form-check-label" for="flexRadioDefault1">
        Yes
    </label>
</div>
<div class="form-check">
    <input class="form-check-input" type="radio" asp-for="Married" value="No">
    <label class="form-check-label" for="flexRadioDefault2">
        No
    </label>
</div>
<textarea asp-for="Description" class="form-control" placeholder="Enter Description" rows="5">
    Write somwthing
</textarea>
<br />
<input type="submit" value="submit" class="btn btn-outline-primary btn-block">
```



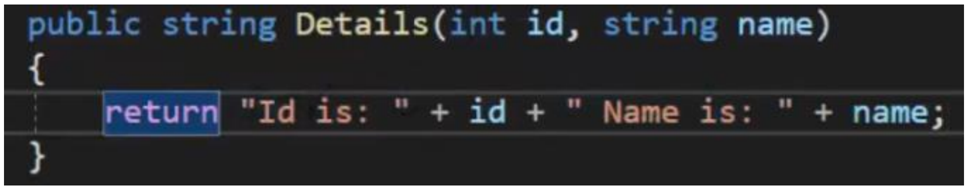
Creating Form Using Tag Helper

- The Form Tag Helper.
- The Form Action Tag Helper.
- The Input Tag Helper.
- The Textarea Tag Helper.
- The Label Tag Helper.
- The Validation Tag Helpers.
- The Select Tag Helper

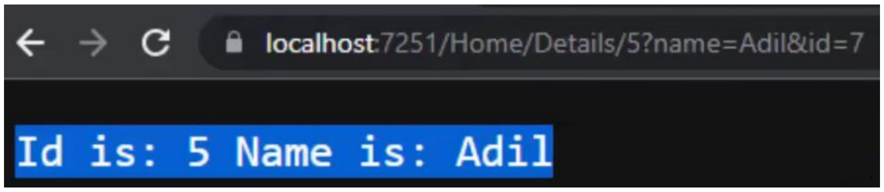
Model binding / Mapping



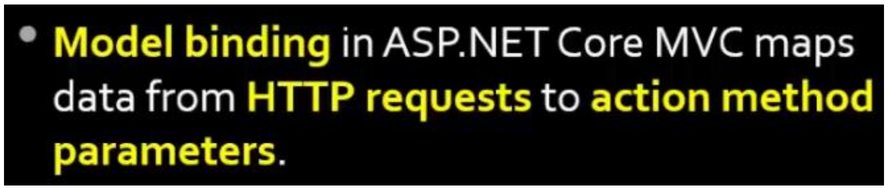
```
public IActionResult Edit(int id)
{
    return View();
}
```



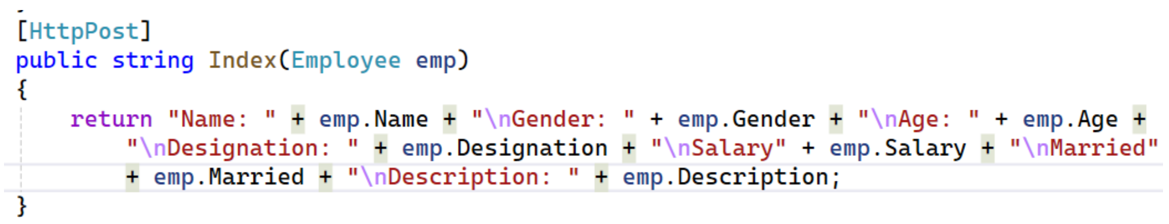
```
public string Details(int id, string name)
{
    return "Id is: " + id + " Name is: " + name;
}
```



```
Id is: 5 Name is: Adil
```



- **Model binding** in ASP.NET Core MVC maps data from **HTTP requests** to **action method parameters**.



```
[HttpPost]
public string Index(Employee emp)
{
    return "Name: " + emp.Name + "\nGender: " + emp.Gender + "\nAge: " + emp.Age +
        "\nDesignation: " + emp.Designation + "\nSalary" + emp.Salary + "\nMarried"
        + emp.Married + "\nDescription: " + emp.Description;
}
```

Validation tag helpers

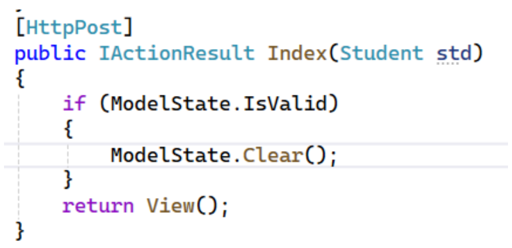
Model

```
public class Student
{
    [Required(ErrorMessage ="Please write your Name")]
    [StringLength(15,MinimumLength = 3,ErrorMessage ="write between 3 to 15 characters")]
    //[MinLength(3)]
    //[MaxLength(15)]
    public string Name { get; set; }

    [Required(ErrorMessage = "Please write your Email")]
    //[EmailAddress]
    [RegularExpression("^([\\w\\.\\-]+)@([\\w\\-]+)((\\.(\\w){2,3})+)$",ErrorMessage ="Invalid Email")]
    public string Email { get; set; }

    [Required(ErrorMessage = "Please Enter your Age")]
    [Range(10,50,ErrorMessage ="Age Between 10 to 50")]
    public int? Age { get; set; }

    [Required(ErrorMessage = "Please Enter your password")]
    [RegularExpression("^(?=.*[A-Za-z])(?=.*\\d)(?=.*[@$!%*#?&])[A-Za-z\\d@$!%*#?&]{8,}$",
        ErrorMessage ="uppercase,lowercase,8 characters,1 special character")]
    public string Password { get; set; }

    [Required(ErrorMessage = "Please Enter Confirm password")]
    [Compare("Password",ErrorMessage = "uppercase,lowercase,8 characters,1 special character")]
    [Display(Name = "Confirm Password")]
    public string ComparePassword { get; set; }

    [Required(ErrorMessage = "Please Enter your Phone Number")]
    [RegularExpression("^((\\+92)|(0092))-{0,1}\\d{3}-{0,1}\\d{7}$|^\\d{11}$|^\\d{4}-\\d{7}$",
        ErrorMessage = "Invalid Phone number")]
    public string PNumber { get; set; }

    [Required(ErrorMessage = "Please Enter Wesite URL")]
    [Url(ErrorMessage= "Invalid URL")]
    public string WebsiteURL { get; set; }
```

View

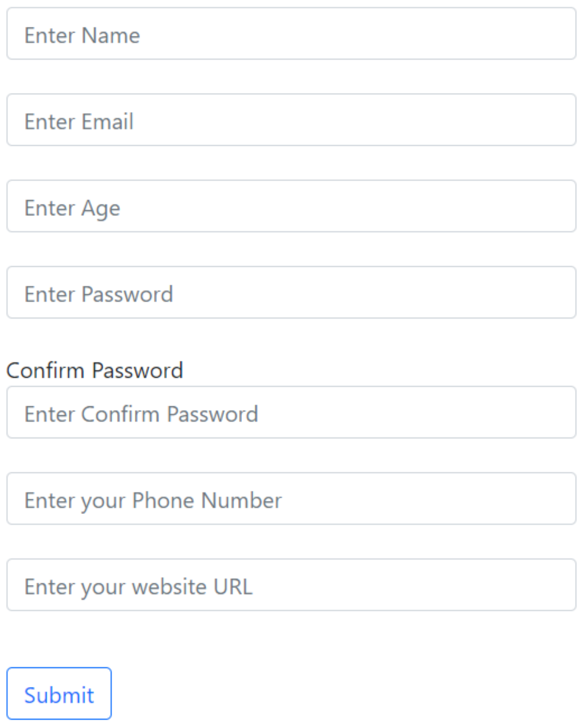
```
<div class="container">
    <div class="row">
        <div class="col-sm-4">
            <form asp-controller="Home" asp-action="Index" method="post">
                <input asp-for="Name" class="form-control" placeholder="Enter Name">
                <span asp-validation-for="Name" style="color: red"></span>
                <br />
                <input asp-for="Email" class="form-control" placeholder="Enter Email">
                <span asp-validation-for="Email" style="color: red"></span>
                <br />
                <input asp-for="Age" class="form-control" placeholder="Enter Age">
                <span asp-validation-for="Age" style="color: red"></span>
                <br />
                <input asp-for="Password" type="password" class="form-control" placeholder="Enter Password">
                <span asp-validation-for="Password" style="color: red"></span>
                <br />
                <label asp-for="ComparePassword"></label>
                <input asp-for="ComparePassword" type="password" class="form-control" placeholder="Enter Confirm Password">
                <span asp-validation-for="ComparePassword" style="color: red"></span>
                <br />
                <input asp-for="PNumber" class="form-control" placeholder="Enter your Phone Number">
                <span asp-validation-for="PNumber" style="color: red"></span>
                <br />
                <input asp-for="WebsiteURL" class="form-control" placeholder="Enter your website URL">
                <span asp-validation-for="WebsiteURL" style="color: red"></span>
                <br />

                <div asp-validation-summary="All" style="color:red;"></div>

                <input type="submit" name="submit" class="btn btn-outline-primary" />
            </form>
        </div>
    </div>
</div>
```

Controller

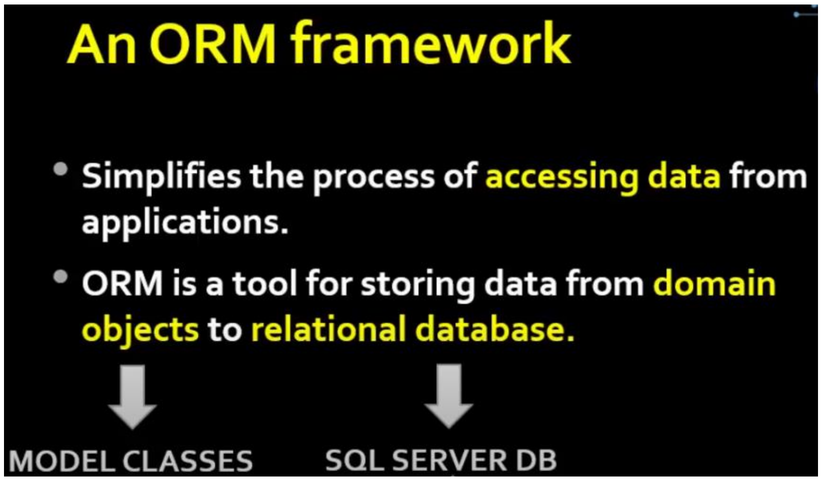


```
[HttpPost]
public IActionResult Index(Student std)
{
    if (ModelState.IsValid)
    {
        ModelState.Clear();
    }
    return View();
}
```

Output



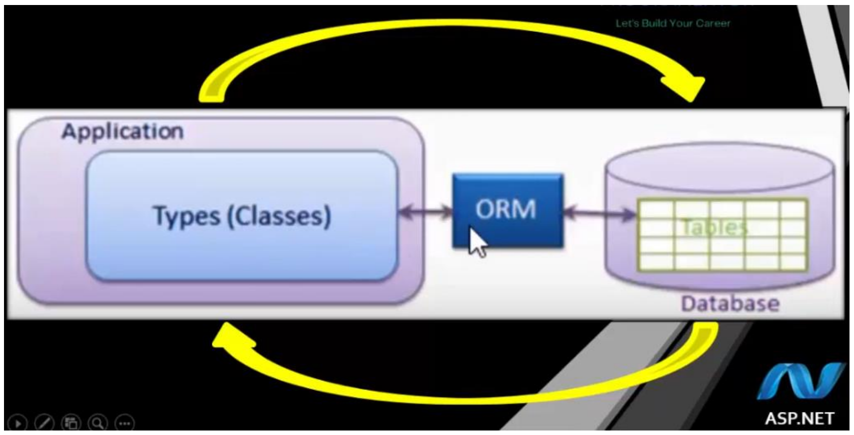
ORM - (OBJECT RELATIONAL MAPPING)

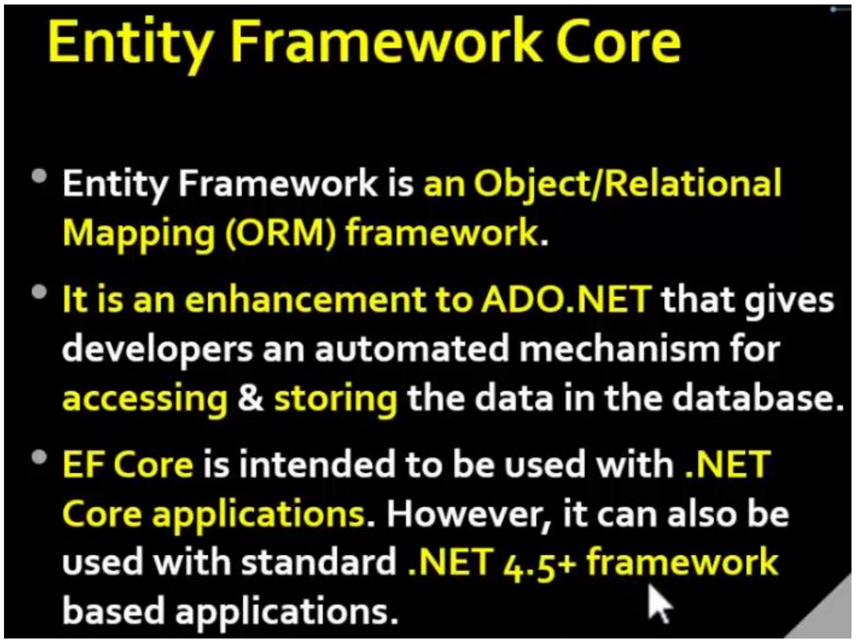


An ORM framework

- Simplifies the process of accessing data from applications.
- ORM is a tool for storing data from domain objects to relational database.

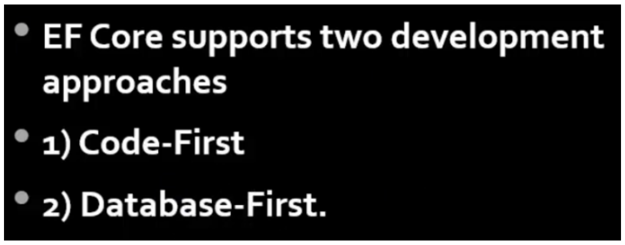
MODEL CLASSES
SQL SERVER DB



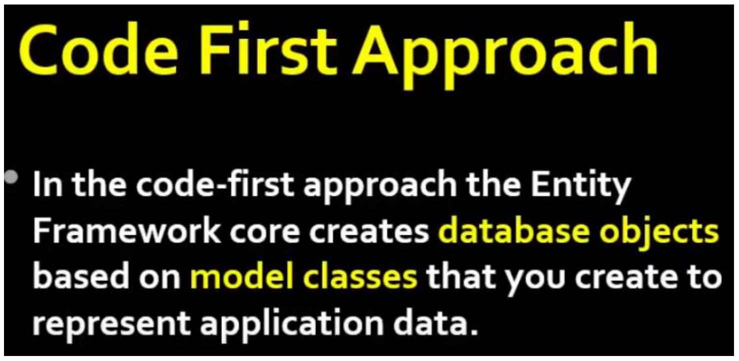


Entity Framework Core

- Entity Framework is an Object/Relational Mapping (ORM) framework.
- It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.
- EF Core is intended to be used with .NET Core applications. However, it can also be used with standard .NET 4.5+ framework based applications.

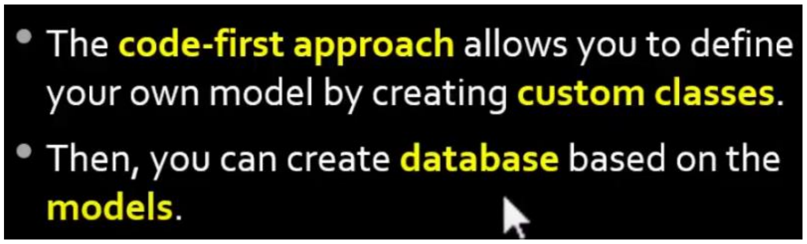


- EF Core supports two development approaches
- 1) Code-First
- 2) Database-First.

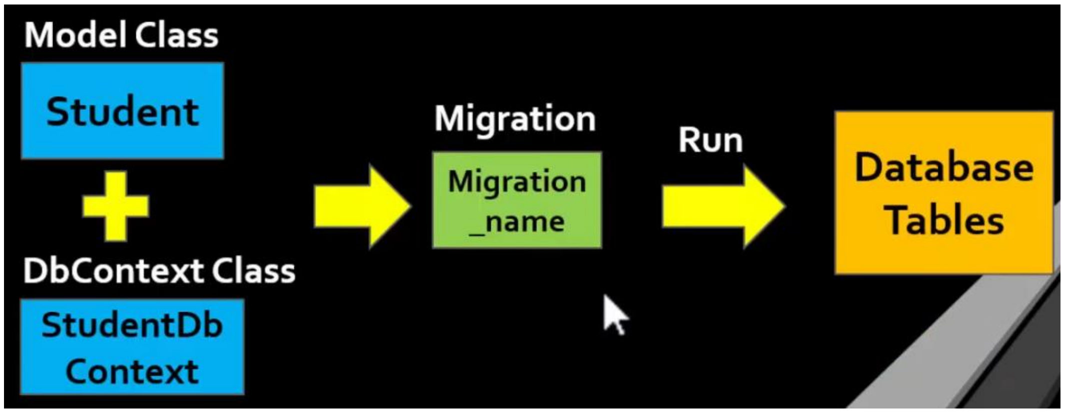


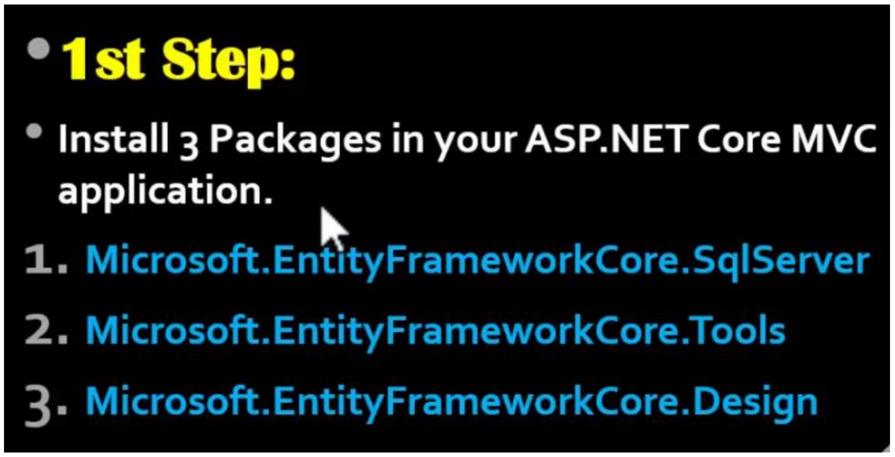
Code First Approach

- In the code-first approach the Entity Framework core creates database objects based on model classes that you create to represent application data.



- The **code-first approach** allows you to define your own model by creating **custom classes**.
- Then, you can create **database** based on the **models**.

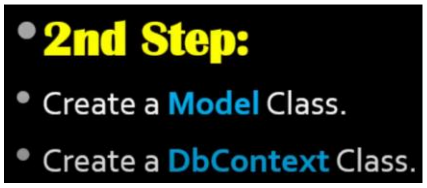




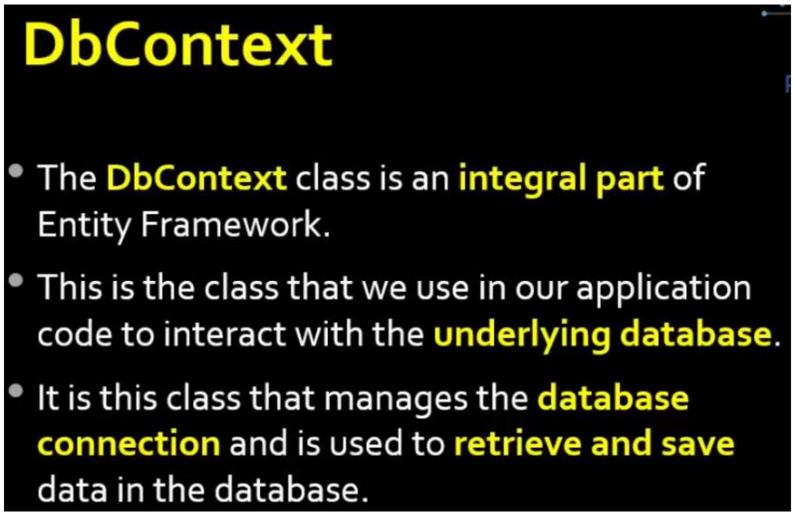
- **1st Step:**
- Install 3 Packages in your ASP.NET Core MVC application.

1. Microsoft.EntityFrameworkCore.SqlServer
2. Microsoft.EntityFrameworkCore.Tools
3. Microsoft.EntityFrameworkCore.Design

Dependencies > Manage NuGet Packages> brows > search for the packages

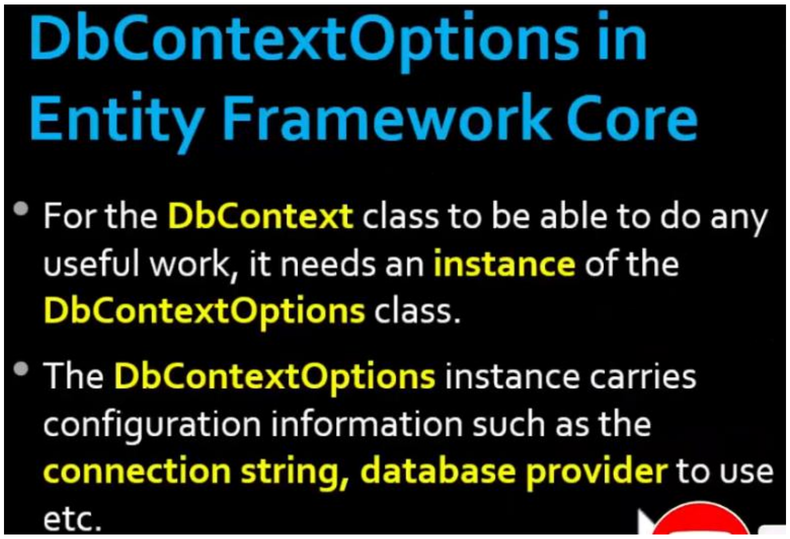


- **2nd Step:**
- Create a Model Class.
- Create a DbContext Class.



DbContext

- The **DbContext** class is an **integral part** of Entity Framework.
- This is the class that we use in our application code to interact with the **underlying database**.
- It is this class that manages the **database connection** and is used to **retrieve and save** data in the database.



DbContextOptions in Entity Framework Core

- For the **DbContext** class to be able to do any useful work, it needs an **instance** of the **DbContextOptions** class.
- The **DbContextOptions** instance carries configuration information such as the **connection string, database provider** to use etc.

Base keyword call parent class constructor