Introduction to ASP.NET Core MVC Framework

The ASP.NET Core framework is growing in popularity among developers and is also anticipated to continue to do so. On the .NET platform, ASP.NET is a well-known web development framework for constructing web applications. **ASP.NET Core MVC** is a robust framework for building online programs and APIs using the **Model-View-Controller** architectural pattern.

What is MVC?

MVC stands for Model View and Controller. It is an **Architectural Design Pattern**, which means it is used at an application's architecture level. So, you need to remember that **MVC is not a Programming Language**. **MVC is not a Framework**. **It is a Design Pattern**. When we design an application, first, we create the architecture of that application, and MVC plays an important role in the architecture of that particular application.

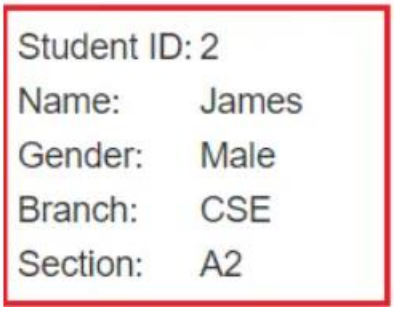
MVC Design Pattern is basically used to develop **Interactive Applications**. An interactive application is an application where there is user interaction involved, and based on the user interaction, some event handling occurs. The most important point you need to remember is that it is not only used for developing Web-Based Applications, but we can also use this MVC Design Pattern to develop Desktop or Mobile-Based applications.

The **MVC (Model-View-Controller) Design Pattern** was introduced in the **1970s,** dividing an application into 3 major components. They are **Model, View, and Controller.** The main objective of the MVC Design Pattern is the separation of concerns/components. It means the **Domain Model and Business Logic** are separated from the **User Interface (i.e., view)**. As a result, maintaining and testing the application becomes simpler and easier.

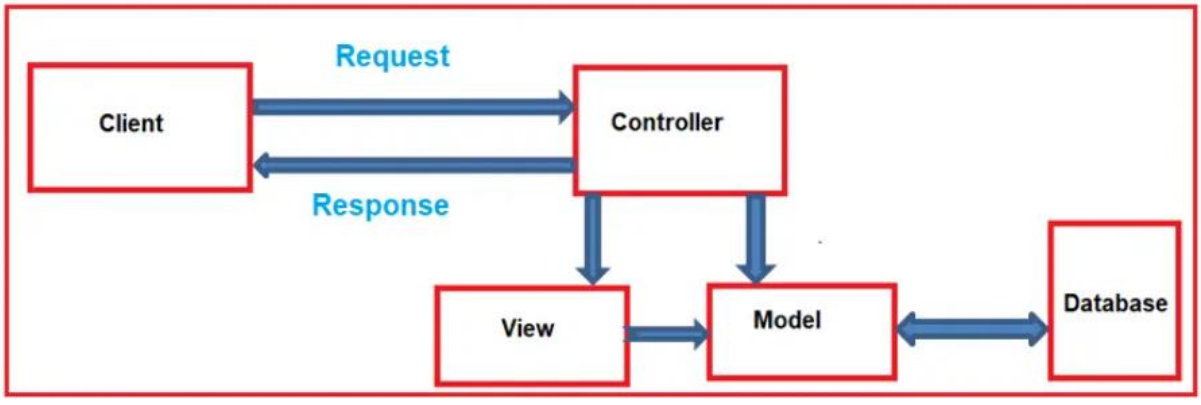
So, in simple words, we can say that the Model-View-Controller (MVC) is an architectural design pattern that separates an application code into three main groups of components: **Models, Views, and Controllers.** This pattern helps to achieve **separation of concerns**.

How Does MVC Design Pattern Work in ASP.NET Core?

Let us see an example to understand how the MVC pattern works in the ASP.NET Core MVC application. For example, we want to design an application where we need to display the student details on a web page, as shown below.



So, when we request something like "**https://localhost:7132/Student/Details/2**" from a web browser, the following things happen to handle the request.



The controller is the MVC design pattern component that handles the incoming request. In order to handle the request, the controller components do several things, as follows. The controller component creates the model that is required by a view. The model is the component in the MVC design pattern which basically contains classes that are used to store the domain data or, you can say, business data. In the MVC design pattern, the Model component also contains the required logic in order to retrieve the data from a database. Once the model is created by the controller, then the controller selects a view to render the domain data or model data. While selecting a view, it is also the responsibility of the controller to pass the model data.

In the MVC design pattern, the only responsibility of the view is to render the model data. So, in MVC, the view is the component responsible for generating the necessary HTML to render the model data. Once the view generates the HTML, that HTML is sent to the client over the network who initially made the request.

So, the three major components of an ASP.NET Core MVC Application are Model, View, and Controller. Let us discuss each of these components of the MVC design pattern in detail.

Role of Model in MVC Design Pattern:

The Model in an MVC application represents the **state of the application** and any business logic or operations it should perform. **Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application.** Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.

So, in simple words, we can say that the Model is the component in the MVC Design pattern that is used to manage the data, i.e., the state of the application in memory. The Model represents a set of classes used to describe the application's validation, business, and data access logic. So, in our example, the model consists of Student and StudentBusinessLayer classes.

Student.cs

```
namespace ASPNETCoreMVCApplication.Models
{
    public class Student
    {
        public int StudentID { get; set; }
        public string? Name { get; set; }
        public string? Gender { get; set; }
        public string? Branch { get; set; }
        public string? Section { get; set; }
    }
}
```

StudentBusinessLayer.cs

```
namespace ASPNETCoreMVCApplication.Models
{
    public class StudentBusinessLayer
    {
        public IEnumerable<Student> GetAll()
        {
            //logic to return all employees
            return new List<Student>();
        }
        public Student GetById(int StudentID)
        {
            //logic to return an employee by employeeId

            Student student = new Student()
            {
                StudentID = StudentID,
                Name = "James",
```

```
            Gender = "Male",
            Branch = "CSE",
            Section = "A2",
        };

        return student;
    }
    public void Insert(Student student)
    {
        //logic to insert a student
    }
    public void Update(Student student)
    {
        //logic to Update a student
    }
    public void Delete(int StudentID)
    {
        //logic to Delete a student
    }
  }
}
```

In our example, we use the Student class to hold the student data in memory. The StudentBusinessLayer class is used to manage the student data, i.e., going to perform the CRUD operation, Validate the Student data, etc.

So, in short, we can say that a Model in the MVC Design Pattern contains a set of classes used to represent the data and the logic to manage those data. In our example, the Student class is the class that is used to represent the data. The StudentBusinessLayer class is the class that is used to manage the Student data, i.e., Validating the Data, Business Logic, and Persisting the data into the database.

Role of View in MVC Design Pattern:

Views in MVC Application are responsible for presenting content through the user interface. In ASP.NET Core MVC Application, we use the **Razor View Engine** to embed **.NET Code in HTML Markup**. There should be minimal logic (you should not write any business logic, calculation logic, etc.) within views, and any logic in them should only be related to presenting the content.

So, the **View Component in the MVC Design pattern** is used to contain the logic to represent the model data as a user interface with which the end-user can interact. Basically, the view is used to render the domain data (i.e., business data) provided to it by the controller.

For example, we want to display Student data on a web page. The Student model carried the student data to the view in the following example. As already discussed, the one and only responsibility of the view is to render the model data, in this case, student model data. The following code does the same thing.

```
@model ASPNETCoreMVCApplication.Models.Student

<html>
<head>
  <title>Student Details</title>
</head>
<body>
  <br />
  <br />
  <table>
    <tr>
      <td>Student ID: </td>
      <td>@Model.StudentID</td>
    </tr>
```

```
        <tr>
            <td>Name: </td>
            <td>@Model.Name</td>
        </tr>
        <tr>
            <td>Gender: </td>
            <td>@Model.Gender </td>
        </tr>
        <tr>
            <td>Branch: </td>
            <td>@Model.Branch</td>
        </tr>
        <tr>
            <td>Section: </td>
            <td>@Model.Section </td>
        </tr>
    </table>
</body>
</html>
```

Role of Controller in MVC Design Pattern:

Controllers in MVC Design Pattern are the components that handle user interaction, work with the model, and ultimately select a view to render. **In an MVC application, the one and only responsibility of a view is to render the information;** the controller handles and responds to user input and interaction. In the MVC Design Pattern, the controller is the initial entry point and is responsible for selecting which model types to work with and which view to render (hence its name – it controls how the app responds to a given request).

In ASP.NET Core MVC Application, a Controller is a **.cs** (for C# language) file which has some **methods called Action Methods (public methods)**. When a request comes on the controller, it is the controller's action method going to handle those requests.

That means the Controller is the component in an MVC application that is used to handle the incoming HTTP Request. Based on the user action, the respective controller will work with the model, select a view to render the information, and then send the response back to the user who initially made the request. So, Controller is the component that will interact with both the models and views to control the flow of application execution. In our example, when the user issued a request, the following URL **https://localhost:7132/Student/Details/2**

Then that request is mapped to the Details action method of the Student Controller. How will it map to the Student Controller's Details action method that will be discussed in our upcoming articles?

```
using ASPNETCoreMVCApplication.Models;
using Microsoft.AspNetCore.Mvc;

namespace ASPNETCoreMVCApplication.Controllers
{
    public class StudentController : Controller
    {
        public ActionResult Details(int studentId)
        {
            StudentBusinessLayer studentBL = new StudentBusinessLayer();
            Student studentDetail = studentBL.GetById(studentId);

            return View(studentDetail);
        }
    }
}
```

As you can see in the example, the Student Controller creates the Student object within the Details action method. So, here the Student is the Model. To fetch the Student data from the database, the controller uses the StudentBusinessLayer class.

Once the controller creates the Student model with the necessary student data, then it passes the Student model to the Details view. The Details view then generates the necessary HTML in order to present the Student data. Once the HTML is generated, then this generated HTML is sent to the client over the network who initially made the request.

Note: The Controller and View depend on the Model in the MVC Design Pattern. But the Model never depends on either View or Controller. This is one of the main reasons for the separation of concerns. This separation of concerns allows us to build and test the model independently of the visual presentation.

Note: **Controllers shouldn't be overly complicated by too many responsibilities**. To keep controller logic from becoming overly complex, push business logic out of the controller and into the domain model.

What is ASP.NET Core MVC?

The ASP.NET Core MVC is a lightweight, open-source, highly testable presentation framework optimized for use with ASP.NET Core. The **ASP.NET Core MVC framework** is used for building Web Apps using the **Model-View-Controller (MVC) Design Pattern**. So, you need to remember that MVC is a Design Pattern, and ASP.NET Core MVC is the Framework based on MVC Design Pattern.

The ASP.NET Core MVC Framework provides us with a patterns-based way to develop dynamic websites with a clean separation of concerns. **This ASP.NET Core MVC framework provides us full control over the markup.** It also supports for Test-Driven Development and also uses the latest web standards such as **HTML 5, Bootstrap, jQuery, etc**.

ASP.NET Core MVC is Microsoft's **Web Application development framework** that is in great demand today. It follows the **Model-View-Controller (MVC) architecture** and incorporates methodologies from Agile Development, along with the most effective aspects of the .NET platform.

ASP.NET Core MVC was introduced by Microsoft in the **latter part of 2015** and has since undergone enhancements to strengthen its functionality further. The latest version available today is the .NET 7.0.

Features of ASP.NET Core MVC:

This framework is bundled with some of the most amazing features. These are:

It is Open Source:

The ASP.NET Core MVC Framework is Open Source which is the main reason behind its popularity. The Entire Source Code of this .NET Core Framework is available at **https://github.com/aspnet,** and you are free to download the source code; even if you want, you can also modify and compile your own version.

Cross-Platform:

The ASP.NET Core MVC Framework is designed from scratch to keep in mind to be Cross-Platform for both development and deployment. ASP.NET Core MVC is cross-platform both for development and deployment. It is available for operating systems such as Windows, Linux & macOS. You can do development works in ASP.NET Core MVC using Visual Studio Editor by Microsoft. Visual Studio works only on Windows and macOS. For Linux, use Visual Studio Code editor.

Full Control Over HTML and HTTP:

With ASP.NET Core MVC, you have complete control over the HTML you generate. This means you can create simple or complex HTML and style it using CSS to display it on the browser. Additionally, you have full control over the HTTP requests that are sent between the server and browser. Creating AJAX requests is also straightforward. ASP.NET Core MVC easily integrates client-side libraries such as jQuery, Angular, React, and Bootstrap.

Extensible Framework:

ASP.NET Core MVC is highly extensible. You can make applications that can be extended to any level in the future. Key features of this framework that gives it the extensible power are:

1. View Components
2. **Tag Helpers**
3. Routing

Testing Made Maintainability:
The ASP.NET Core MVC architecture is an excellent choice if you want a maintainable and testable application. It allows you to divide various parts of your application into separate and independent pieces, which can be tested individually. You can easily integrate testing frameworks like xUnit, MSTest, and MOQ to simulate different scenarios. With ASP.NET Core MVC, maintaining large to very large applications becomes effortless.

Routing
With ASP.NET Core MVC Routing, creating SEO-friendly URLs becomes effortless and manageable from a single location, eliminating the probability of errors. You no longer need to hard-code the URL, as the Routing feature generates it based on your established structure.

API
If you're a C# programmer, you'll be pleased to know that ASP.NET Core MVC APIs can utilize various language features such as the async and await keyword, extension methods, lambda expressions, anonymous and dynamic types, as well as Language Integrated Query (LINQ). That means all the C# language features can be used here.

When to Choose ASP.NET MVC and When to Choose ASP.NET Core MVC?
When to Choose ASP.NET MVC?
1. You are currently working on an existing application developed using ASP.NET MVC and would like to expand the functionalities by adding new features.
2. Your team is familiar with ASP.NET MVC Framework but has yet to gain experience with ASP.NET Core MVC.
3. If you want your application will only be compatible with devices and servers that run on the Windows Operating System.

When to Choose ASP.NET Core MVC?
1. You have a preference for utilizing a framework that is completely open source.
2. You want your application to be able to be developed and hosted on any type of operating system.
3. Your team members have knowledge of ASP.NET Core MVC.
4. You are looking for an application framework that has a long development roadmap ahead of it. If you see the road map of .NET, Microsoft has already provided the road map for the next five years.