# Day 5 :- ASP.NET Core Basics ! Mode-View-Controller (MVC) & Razor Pages.

ASP.Net Core is a powerful, open-source, cross-platform framework designed for building modern web applications including MVC applications, web APIs, and Razor Pages.

## * ASP.NET Core Framework Overview

ASP.NET Core is an evolution of the .NET Ecosystem with better ecosystem with better performance, flexibility, and cross-platform capabilities. It can run on windows, mac os, and linux.

Key features of ASP.NET Core

- Cross Platform : Develop applications on Windows, macOS and Linux.
- High performance : optimized for speed and scalability
- Modular middleware pipeline : Uses middleware components to handle HTTP requests.
- Built-in dependency injection : Simplifies the management of services and dependencies.
- Supports MVC & Razor Pages : Helps in developing structured, maintainable web applications.



## * Introduction to Model-View-Controller (MVC) Architecture

MVC is a design pattern used to separate concerns in web applications. It helps improve code organization, scalability, and testability.

Components of MVC architecture

1. Model
   - Represents the data and business logic of the applications.
   - Interacts with the database and processes user input.
   - Example : If you have a Product list, the Model would define how products are stored and retrieved.
   - Example Model :

```
Public class Product {
    Public int Id {get; set;}
    Public string Name {get; set;}
    Public decimal Price {get; set;}
```

2. View
   - Represents the UI (User Interface) of the application.
   - Uses Razor Syntax to dynamically render content.
   - Example View (Index.cshtml) :

```
@model List<Product>

<h2>Product List</h2>
<table>
    @foreach (var product in Model) {
        <tr>
            <td>@Product.Name</td>
            <td>@product.Price</td>
        </tr>
    }
</table>
```

3 Controller

- Handles user requests and responses.
- Fetches data from the model and sends it to the view.
- Example: If a user requests http://localhost/Products, the Controller processes the request and loads the Product List.
- Example Controller:

```
Using Microsoft.AspNetCore.MVC;

Public class ProductController: Controller {
    Public IActionResult Index() {
        List<Product> Products = new List<Product> {
            new Product {Id=1, Name="Laptop", Price=
            new Product {Id=2, Name="Smartphone",
                                        Price=500}
        };
        return View(Products);
    }
}
```



MVC Workflow Summary:

- User makes a request → sent to Controller.
- Controller fetches data from the Model.
- View renders the data dynamically.
- Response is sent to the browser.

3 Lifecycle of an HTTP Request in ASP.NET Core

When a user visits a webpage, the request flows through different layers before reaching final view.

Steps in HTTP Request Pipeline:

- User sends a request → Example: Visiting /Products in a browser.
- Request is processed through Middleware → Middleware filters or modifies the request.
- Routing determines the Controller & Action → Maps URL to the correct Controller (eg., ProductController).
- Controller processes the Request → Calls the Model to fetch data.
- View is rendered and sent as an HTML response.

4 Building a Basic ASP.NET Core MVC Application

Setting up Controllers and Actions
In ASP.NET Core MVC, Controllers handle the incoming HTTP requests and return responses.

Example: Creating a simple Controller

```
Using Microsoft.AspNetCore.Mvc;

Public class HomeController : Controller {
    Public IActionResult Index() {
        return View();
    }
```

- Index() is an Action method that returns the Index.cshtml view

Understanding Razor views & Syntax

Razor is the templating engine used in ASP.NET Core to combine C# code with HTML

Razor Syntax Examples

- Using C# inside HTML:

```
@{
    string message = "Hello, ASP.NET Core!";
}
<h1>@message</p>
```



- Looping through data:

```
@for (int i = 1; i <= 5; i++)
{
    <p> Item @i </p>
}
```

- Displaying a dynamic list (from Controller model):

```
@model List<Product>

<ul>
    @foreach (var product in Model)
    {
        <li>@product.Name - $@product.Price</li>
    }
</ul>
```