8/8/24

ASSIGNMENT 1

Unit - 1 Introduction

Q.1 What is ASP.NET Core ? Explain any four features of ASP.NET Core.

—> ASP.NET core is an open source, cross-platform framework for building modern, cloud-based, internet-connected applications. It is designed to be lightweight, modular and high performance. It can run on windows, macOS and linux, making it a versatile option for developers.

—> Four Key Features of ASP.NET Core :-

1. Cross-Platform Capabilities

Unlike its predecessor, ASP.NET Core can run on windows, macOS and linux. This flexibility allows developers to choose the operating system that best suits their needs.

2. Unified MVC & Web API.

It provides a single programming model for building both web applications and web APIs which simplifies development and reduces code duplication.

3. High Performance.

It is known for its efficient performance, making it ideal for building fast and responsive web applications. It

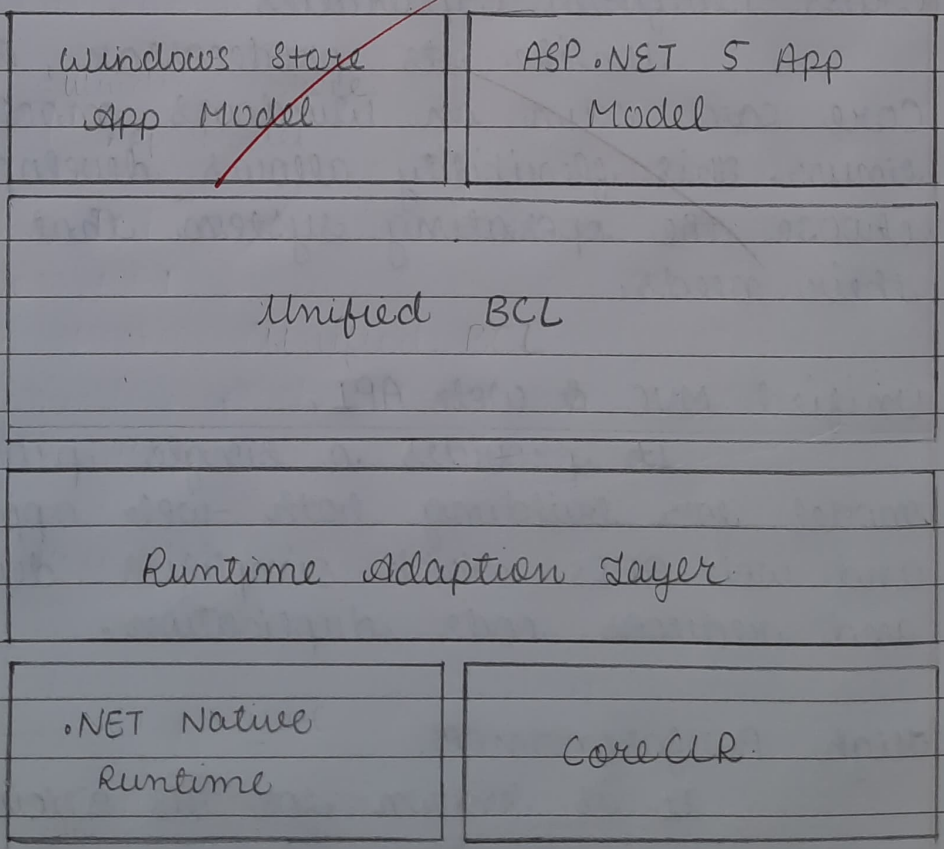
achieves this with a modular design and support for asynchronous programming.

4. Dependency Injection.

It heavily utilizes dependency injection, a design pattern that promotes loose coupling and improves code testability. This allow developers to easily swap out dependencies without modifying core application logic.

Q.2 Draw the .NET Core architecture and explain any four components.

→



The .NET core architecture is designed to be modular, cross-platform and high-performance

→ Four Components of .NET Core Architecture :-

1. Application Model (Windows Store App Model)

The Application Model defines the types of applications you can build using .NET core. It provides the framework and infrastructure required to develop different types of applications. It include models like ASP.NET Core for web applications, the Windows Store App Model for building UWP (Universal Windows platform) applications and Console Applications for command-line apps.

2. Unified Base Case library (BCL)

The Base Case library is a set of standard libraries provided by .NET Core. These libraries are a toolbox of reusable blocks for common tasks like data access, networking, threading, I/O. It's the same toolbox on Windows, macOS or linux promoting code or portability.

3. Runtime Application (Adaption) layer.

It includes core runtime features that are necessary for running .NET Core app. This include garbage collection, memory

management and exception handling.

4. CoreCLR (Common Language Runtime)
CoreCLR is the execution engine for .NET Core. It includes JIT compiler, which compiles Intermediate Language (IL) code to machine code at runtime. CoreCLR also includes the CLR, which handles the execution of managed code, garbage collection and more.

Q.3 What is .NET Core? Explain any four characteristics of .NET Core.

—↳ .NET Core is a free, open-source and cross-platform framework developed by Microsoft for building a variety of applications, including web, mobile, gaming, cloud, IOT and more. It is a successor to the .NET framework, designed to be more modular, flexible and optimized for performance, particularly in cloud and microservices environments.

—↳ Four Key Characteristics of .NET Core :-

1. Cross-Platform Compatibility.
It runs seamlessly on Windows, macOS and linux, providing flexibility for developers and deployment options. Also it enables consistent ~~and~~ development and

execution across different operating systems.

2. Open - Source.

It is fully open source, with its source code on GitHub. This transparency allows developers to contribute to the framework, customize it for specific needs and benefit from a community - driven ecosystem.

3. Performance - Oriented

It is designed for high performance, especially in cloud - based environments. Also it is optimized for speed and efficiency & it also offers features like Asynchronous programming and efficient memory management.

4. Modular Architecture.

It allows developers to include only the necessary components for their application which reduces its size and improves startup time. Also offers greater control over the development process and deployment.

Q.4 Write and explain any five advantages of ~~A~~ ASP.NET Core.

→ i) High Performance and Scalability

They are critical for modern web applications, especially those that need to

handle a large number of concurrent users or high traffic. ~~It~~ This ensures that applications can grow with user demand without degrading performance.

ii) Unified Development Model

This unification simplifies the development by providing a consistent programming model for building both web applications and APIs. It promotes code reuse and easier maintenance.

iii) Modular and lightweight.

This modularity results in smaller application sizes and reduced memory footprint, leading to faster application startup, also enhancing security and better overall performance.

iv) Tag Helpers

These make Razor markup more natural by allowing developers to write server-side code using standard HTML syntax, improving the readability and maintainability of views.

v) Flexible Hosting

ASP.NET Core applications can be hosted on IIS or self-hosted in any process, providing flexibility in deployment options.

* Cross-Platform, Dependency Injection, Cloud Ready configuration

Q.5 Write two examples of server rendered UI. Explain advantages and limitations of server rendered UI.

→ Examples of Server Rendered UI are :-

i) E-commerce Product Page : When a user clicks on a product, the server generates the HTML for the product page including product details, images, pricing, reviews. This fully formed HTML is then sent to the client's browser, which renders the page without requiring additional Javascript processing.

ii) Django : A Python-based web framework that follows the MVC pattern (MVT in Django : Model-View-Template). Here, the server generates HTML views that are sent to the client, allowing for the dynamic generation of content based on the request.

→ Advantages of Server-Rendered UI.

(i) Improved SEO : Search engines can easily crawl and index server-rendered pages, leading to better search rankings.

(ii) Faster Initial Load Times : Since the complete HTML is delivered to the client, the initial page load is often faster than client-side rendering.

(iii) Works Well with slow or Offline connections : Server-rendered pages can be displayed even with limited or no internet connectivity.

—↓ Limitations of Server-Rendered UI :~

(i) Increased Server Load : Generating HTML on the server can be computationally expensive, especially for complex pages or high traffic websites.

(ii) Slower updates : Dynamic content updates require full page reloads, which can be less user-friendly compared to client-side updates.

(iii) Code Duplication : Logic for rendering the UI might be duplicated between the server and client, leading to maintenance challenges.

Q.6 Explain Client-Rendered UI. Discuss advantages and limitations of it.

—↓ Client-Rendered UI refers to the approach where the rendering of the user interface ~~(UI)~~ is handled primarily on the client side, typically within the web browser. This is achieved using JavaScript frameworks and libraries like React, Angular or Vue.js, which manage the DOM and update the UI dynamically based on user interactions.

Here, the server typically sends

a minimal HTML file to the client, along with Javascript files that contain the application logic. Once the browser loads these files, the Javascript takes over to render the UI and handle user ~~experience~~. interactions.

→ Advantages of Client-Rendered UI :~

(i) Enhanced User Experience : It ensures smooth transitions and animations ~~one~~ without full page reloads. Also provides real-time updates and interactive features.

(ii) Improved Performance : Once the initial HTML is loaded, subsequent interactions are handled efficiently on the client side, reducing server load.

(iii) Flexibility : It supports richer user interfaces with complex interactions can be created. Also it is easier to build Single-page applications (SPAs).

→ Limitations of Client-Rendered UI :~

(i) Slower Initial Load : The initial page load can be slower as Javascript needs to be downloaded and executed before the UI is fully rendered.

(ii) SEO Challenges : Search engines might have difficulty indexing dynamic content generated by Javascript.

(iii) Dependency On Javascript : Users with Javascript disabled might have a poor experience.

Q.7 What are Blazor Server and Blazor Web Assembly ? Explain.

—v Blazor is a framework for building interactive web UIs using C#. It offers two primary hosting models :

1. Blazor Server

Blazor Server is a hosting model where the application runs on the server and the UI ~~components~~ updates are sent to the client over a SignalR connection. The browser renders the UI, but all processing is handled on the server. This model is suitable for applications that require frequent server-side processing or access to server-side resources.

- Advantages :~
  (i) Simplified Development
  (ii) Full .NET Core API Support
  (iii) Fast Initial Load.

- Limitations :~
  (i) Server load
  (ii) Latency
  (iii) Offline Support.

2. Blazor WebAssembly (Blazor WASM)

Blazor WebAssembly is a hosting model where the application runs directly in the browser on WebAssembly. The entire application, including the .NET runtime, is downloaded to the client and executed there. This model provides a more traditional client-side development experience.

- Advantages :-
(i) Rich Interactivity
(ii) Reduced Server load.
(iii) Offline Capabilities

- Limitations :-
(i) Larger Initial Download.
(ii) Browser Constraints
(iii) Security Considerations.

Q.8 Write and explain commands to create ASP.NET Core MVC project.

→ To create an ASP.NET Core MVC project, you can use the .NET CLI commands :-

1. Open your preferred command-line interface i.e. Command Prompt, Powershell, Terminal.

2. Navigate to the desired directory where you want to create the project, using 'cd'.

cd /Users/angatshah0511/Desktop

3. Create ~~the~~ a new ASP.NET Core MVC project, using the 'dotnet new' ~~mvc -n MyMvcApp~~ command.

dotnet new mvc -n MyMvcApp.

⇒ This command creates a new ASP.NET Core MVC project. 'dotnet new' creates a new project, 'mvc' specifies the template to use and '-n MyMvcApp' sets the name.

4. Now move/navigate to the newly created project directory.

cd MyMvcApp

⇒ This command changes the current directory to the newly created project directory

5. Now run the project by using the 'dotnet run' command.

dotnet run.

⇒ This command builds and runs the application. It will use the project's default configuration to run the application.

Q.9 write and explain content of launchSettings.json file.

→ The launchSettings.json file in an ASP.NET core project contains settings that are used when the application is executed locally.

→ {

"iisSettings" : {

```
    "windowsAuthentication" : false,
    "anonymousAuthentication" : true,
    "iisExpress": {
      "applicationUrl" : "http://localhost:56034",
    "sslPort" : 44390
    }
  },
  "profiles" : {
    "IIS Express" : {
      "commandName" : "IISExpress",
      "launchBrowser" : true,
      "launchUrl" : "weatherforecast",
      "environmentVariables" : {
          "ASPNETCORE_ENVIRONMENT" : "Development"
      }
    },
    "MyMvcApp" : {
      "commandName" : "Project",
      "dotnetRunMessages" : true,
      "launchBrowser" : true,
      "applicationUrl" : https://localhost:5001;
       http://localhost:5000",
      "environmentVariables" : {
          "ASPNETCORE_ENVIRONMENT" : "Development"
      }
    }
  }
}
```

ii) iisSettings : Configures settings for running

the application with IIS (Internet Information Services) Express.

> windowsAuthentication : Indicates whether windows authentication is enabled (here it is false).

> anonymousAuthentication : Indicates whether anonymous authentication is enabled (here it is true).

ii) profiles : Contains different profiles for launching the application. Each profile can have its own settings for how the application should be started.

- IIS Express :-

> commandName : Specifies the command to use for launching.

> launchBrowser : Indicates whether the browser should be launched automatically when the application starts (here it is True).

> launchUrl : The relative URL to open in the browser when the application starts.

- MyMvcApp :-

> dotnetRunMessages : Indicates whether to show messages from dotnet run (here it is true)

> launchBrowser : Indicates whether the browser should be launched automatically when the application starts (here it is True).

> environmentVariables : A dictionary of environment

variables to set for this profile. Here, ASPNETCORE_ENVIRONMENT is set to development.

Q.10 With necessary code snippet, explain endpoint and HTML rendering.

—4 In ASP.NET Core, endpoints and HTML rendering are used to handle incoming HTTP Requests and generate the appropriate HTML Response.

1. ~~Em~~ Endpoint

An endpoint in ASP.NET Core is a route that handles a specific request. Endpoints are defined using routing, which maps incoming requests to corresponding actions in controllers or Razor Pages.

eg:

```
using Microsoft.AspNetCore.Mvc;
namespace MyWebApp.Controllers
{
    public class HomeController : Controller {
        [HttpGet("/")]
        public IActionResult Index() {
            return View();
        }
        [HttpGet("/about")]
        public IActionResult About() {
            return View();
        }
    }
}
```

2. HTML Rendering

HTML rendering in ASP.NET Core refers to the process of converting Razor Views or Razor Pages into HTML content that is sent to the client's browser. Razor Syntax allows mixing C# code with HTML to generate dynamic web pages.

eg:

```
@ {
    ViewData["Title"] = "Home Page";
}
<!DOCTYPE html>
<html>
<head>
    <title> @ViewData["Title"] </title>
</head>
<body>
    <h1> Welcome to My Web App </h1>
    <p> This is the home page. </p>
</body>
</html>
```

Q. "Step-by-step explain ASP.NET Core snippet for controller and view to demonstrate use of "asp-action".

—4 The 'asp-action' attribute is used ~~To~~ in ASP.NET Core Razor views to generate URLs that link to specific methods in a controller. This is often used in HTML forms, hyperlinks, or buttons to direct the user

to another page or action within the application.

Step-1: Create a Controller with Action Methods.

```
using Microsoft.AspNetCore.Mvc;
namespace MyWebApp.Controllers {
    public class ProductsController : Controller {
        public IActionResult Index() {
            return View();
        }
        public IActionResult Details(int id) {
            return View();
        }
        public IActionResult Create() {
            return View();
        }
    }
}
```

- 'ProductsController' handles requests related to products
- 'Index()' returns a view displaying a list of products.
- 'Details(int id)' shows details for a specific product
- 'Create()' displays a form to create a new product.

Step-2: Create views for the controller Actions.

⇒ Create the "Index" view

```
@{
    ViewData["Title"] = "Products List";
}
<h1>@ViewData["Title"]</h1>
<ul>
    <li>
        <a asp-controller="Products" asp-action=
        "Details" asp-route-id="1">Product 1</a>
    </li>
</ul>
<a asp-controller="Products" asp-action="Create">
Create New Product</a>
```

⇒ Create the "Details" view

```
@{
    ViewData["Title"] = "Product Details";
}
<h1>@ViewData["Title"]</h1>
<p>Details of Product with ID @ViewBag.ProductId</p>
<a asp-controller="Products" asp-action="Index">
Back to Products list</a>
```

Q.12 Explain following code snippet :

```
① using Microsoft.AspNetCore.Mvc;
  using PartyInvites.Models;
② namespace PartyInvites.Controllers {
③     public class HomeController : Controller {
④         public IActionResult Index() {
              return View();
          }
```

```
⑤   [HttpGet]
    public ViewResult RsvpForm() {
        return View();
    }
⑥   [HttpPost]
    public ViewResult RsvpForm(GuestResponse
     guestResponse) {
        //TODO : store response from guest
        return View();
    }
}
}
```

1. Namespaces and Usings.
   - 'using Microsoft.AspNetCore.Mvc': This brings in the ASP.NET Core MVC framework components such as controllers, views and action results. It enables the use of MVC features.
   - 'using PartyInvites.Models': This imports the namespace, which likely contains the 'GuestResponse' model class. This class is used to capture and process guest responses.

2. Namespace Declaration
   This helps organize the code and avoids naming conflicts with other classes.

3. HomeController Class
   It inherits from the 'Controller' base class. It handles incoming HTTP requests and return responses.

4. Index Action Method

This method handles requests to the root URL ("/") or "/Home/Index" by convention. This returns a view associated with the 'Index' action. By default, it will look for a Razor view file named 'Index.cshtml' in the 'Views/Home' directory.

5. RsvpForm Action Method (GET)

The 'HttpGet' specifies that the 'RsvpForm' method should respond to HTTP GET requests. Also this method returns a view that displays the RSVP form to the user. (looks for 'RsvpForm.cshtml')

6. RsvpForm Action Method (POST)

The 'HttpPost' specifies that the 'RsvpForm' method should respond to HTTP post requests. This method is invoked when the user submits the RSVP Form. It accepts a 'GuestResponse' object as a parameter. The object is populated with the form data submitted by user. Then the 'return View()' redirects the user to a "Thank You" page or display a confirmation message.

Q.13 Create necessary classes and demonstrate use of @model. Also discuss required code

—4 ASP.NET

The '@model' directive in ASP.NET Core is used in Razor views to specify the type of data that

the view expects. This allows you to strongly type the view, making it easier to work with model data within the view.

1. Creating the Model Class.

```
// Models / GuestResponse.cs
namespace PartyInvites.Models {
    public class GuestResponse {
        public string Name { get; set; }
        public string Phone { get; set; }
        public bool? WillAttend { get; set; }
    }
}
```

2. Creating the Controller.

```
// Controllers / HomeController.cs
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers {
    public class HomeController : Controller {
        [HttpGet]
        public ViewResult RsvpForm() {
            return View();
        }
        [HttpPost]
        public ViewResult RsvpForm(GuestResponse guest
        Response) {
            if (ModelState.IsValid) {
                // TODO: Store the guest response
                return View("Thanks", guestResponse);
            }
```

```
        else {
            return View();
        }
    }
}
```

3. Creating the View.

```
@model PartyInvites.Models.GuestResponse.
<!DOCTYPE html>
<html>
<head>
    <title> RSVP Form </title>
</head>
<body>
    <h1> RSVP for the Event </h1>
    <form asp-action = "RsvpForm" method = "post">
      <div>
        <label asp-for = "Name" ></label>
        <input asp-for = "Name" />
        <span asp-validation-for = "Name" ></span>
      </div>
```

* Same for Phone and WillAttend.

```
    <button type = "submit"> Submit RSVP </button>
  </form>
</body>
</html>
```

⇒ Finally, create the "Thanks" View to display a confirmation message after form is submitted.

> Model : The 'GuestResponse' class represents the data being collected.
> Controller : The 'HomeController' manages the display and processing of the RSVP form.
> View : The '@model' directive is used to ~~display~~ strongly type the view, allowing you to work directly with the model's properties within the Razor template.

Q.14 With suitable example code snippet of of controller, model and view, explain validation.

—> Validation in ASP.NET Core ensures that the data submitted by users meets certain criteria before it is processed. ASP.NET Core MVC provides built-in validation attributes that can be applied to model properties and these validations are automatically enforced when the model is bound to the user input.

1. Model with Validation Attributes.

```
using System.ComponentModel.DataAnnotations;
namespace PartyInvites.Models {
    public class GuestResponse {
        [Required(ErrorMessage = "Please Enter Your name")]
        public string Name { get; set; }
* Now same code as Question - 13 (pg: 23)
    }
}
```

- '[Required]' : This attribute ensures that the property is not left empty. If it is, a validation error is triggered.
- '[EmailAddress]' : This attribute checks that the input is a valid email address.
- 'ErrorMessage' : Custom error messages are specified using the property of the attribute.

2. Controller to Handle Form Submission.
The controller handles the form submission and checks if the model is valid before proceeding.

3. View with Validation Messages.
In the razor view, use the '@model' directive to bind the view to the 'GuestResponse' model. The view should ~~to~~ also include validation messages to inform users of any input errors.

☆ For example, the code is same as Question-13.

Q.15 Develop ASP.NET Core MVC view which uses Bootstrap. With code snippet, explain any five classes of bootstap.css.

-4 To integrate Bootstrap into an ASP.NET core MVC view, you can utilize Bootstrap's CSS classes to enhance the UI's appearance and functionality.

1. Setting Up Bootstrap in ASP.NET Core.
You can add it via a CDN in your '_Layout.cshtml' file.

2. Creating the MVC view.

```
@model PartyInvites.Models.GuestResponse
<!DOCTYPE html>
<html>
<head>
    <title> RSVP Form </title>
</head>
<body>
    <div class = "container mt-5">
        <h1 class = "text-center"> RSVP </h1>
        <form asp-action = "RsvpForm" method = "post">
            <div class = "form-group">
                <label asp-for = "Name" class = "font-
                weight-bold"> </label>
                <input asp-for = "Name" class = "form-
                control" />
                <span asp-validation-for = "Name"
                class = "text-danger"> </span>
            </div>
            <button type = "submit" class = "btn btn-
            primary btn-block"> Submit RSVP </button>
        </form>
    </div>
</body>
</html>
```

—↓ i) 'container': Creates a responsive, fixed-width container for centering and padding content.

ii) 'mt-5': Adds a ~~top-margin~~ margin top of 3 rem (48 px) to an element. (vertical spacing)

iii) 'form-group': Groups form controls and label with appropriate spacing.

iv) 'form-control': Styles form inputs and textareas with consistent padding and border.

v) 'btn and btn-primary': Creates buttons with predefined styles. 'btn' is a base class and 'btn-primary' adds the primary button style.

Q.16 Create sample Program.cs file for ASP.NET core MVC project and explain any three statements from it.

—↓ The 'Program.cs' file is crucial for configuring and starting the application. It sets up the web host and configures services and middleware.

—↓ 'Program.cs' file

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
```

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
namespace YourNamespace {
    public class Program {
        public static void Main(string[] args) {
            CreateHostBuilder(args).Build().Run();
        }
        public static IHostBuilder CreateHostBuilder
        (string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

-4 i) CreateHostBuilder Method : Responsible for creating and configuring the foundation of the app. It establishes the basic building blocks for hosting the application.

ii) Main Method : This method serves as the entry point for your application. It initiates the creation and execution of the host responsible for handling incoming requests.

iii) UseStartup<Startup> Method : Tells the web server to use a specific class (Startup) for configuring how the application works.

End