UNIT-III

1. Building websites using ASP.NET Core.

Ans: Information about building websites using ASP.NET Core:

- **Structure of Websites:**
    - Websites have multiple pages loaded either statically or dynamically.
    - A web browser uses URLs to make requests to get pages and can interact with the server using different request types like GET, POST, PUT, and DELETE.
- **Browser and Server Roles:**
    - In many cases, the web browser acts as a presentation layer, and most processing happens on the server.
    - Some client-side JavaScript may be used for specific presentation features, like carousels.
- **ASP.NET Core Technologies:**
    - **Razor Pages and Razor Class Libraries:** Used for dynamically generating HTML, especially for simpler websites.
    - **MVC (Model-View-Controller):** An implementation of the MVC design pattern for developing complex websites.
    - **Blazor:** Enables building UI components using C# and .NET, replacing JavaScript-based frameworks like Angular or React.
        - Blazor WebAssembly runs C# code in the browser, like a JavaScript framework.
        - Blazor Server runs C# code on the server, dynamically updating the web page.
        - Not limited to websites; can also create hybrid mobile and desktop apps.

2. New features in ASP.NET Core.

Ans: Following are the new features that have been included for every version:

- ASP.NET Core 1.0 (2016) focused on implementing a minimum API suitable for building modern cross-platform web apps and services for Windows, macOS, and Linux.
- ASP.NET Core 1.1 (2016) focused on bug fixes and general improvements to features and performance.
- ASP.NET Core 2.0 (2017) focused on adding new features such as Razor Pages, bundling assemblies into a Microsoft.AspNetCore. All metapackage, targeting .NET Standard 2.0, providing a new authentication model, and performance improvements.
- ASP.NET Core 2.1 (2018, LTS) focused on adding new features such as SignalR for real-time communication, Razor class libraries for reusing web components, ASP.NET Core Identity for authentication, and better support for HTTPS.
- ASP.NET Core 2.2 (2018) focused on improving the building of RESTful HTTP APIs, updating the project templates to Bootstrap 4 and Angular 6, an optimized configuration for hosting in Azure, and performance improvements.
- ASP.NET Core 3.0 (2019) focused on fully leveraging .NET Core 3.0 and .NET Standard 2.1, which meant it could not support .NET Framework, and it added useful refinements.
- ASP.NET Core 3.1 (2019, LTS) focused on refinements like partial class support for Razor components and a new <component> tag helper.
- ASP.NET Core 5.0 (2020) focused on bug fixes, performance improvements using caching for certificate authentication, HPACK dynamic compression of HTTP/2 response headers in Kestrel, nullable annotations for ASP.NET Core assemblies, and a reduction in container image sizes.
- ASP.NET Core 6.0 (2021) d focused on productivity improvements like minimizing code to implement basic websites and services, .NET Hot Reload, and new hosting options for Blazor, like hybrid apps using .NET MAUI.

3. Understanding web development.

Ans: Developing for the web means developing with Hypertext Transfer Protocol (HTTP).

- **Understanding HTTP:** To communicate with a web server, the client, also known as the user agent, makes calls over the network using HTTP. As such, HTTP is the technical underpinning of the web. So, when we talk about websites and web services, we mean that they use HTTP to communicate between a client (often a web browser) and a server. A client makes an HTTP request for a resource, such as a page, uniquely identified by a Uniform Resource Locator (URL), and the server sends back an HTTP response.
- **Using Google Chrome to make HTTP requests:**
    - **Step 1:** Start Google Chrome.
    - **Step 2:** Navigate to More tools | Developer tools.
    - **Step 3:** In Chrome's address box, enter https://dotnet.microsoft.com/learn/aspnet
    - **Step 4:** In Developer Tools, in the list of recorded requests, scroll to the top and click on the first entry, the row where the Type is document.
    - **Step 5:** On the right-hand side, click on the Headers tab, and you will see details about Request Headers and Response Headers. Definitions of the key terms present there:
        - **GET:** Request method to retrieve a resource from the server.
        - **200 OK:** Status code indicating a successful response.
        - **Request Headers:** Browser-sent information including accepted formats, compression algorithms, and preferred languages.
        - **accept:** Lists accepted formats with quality values, indicating preferences.
        - **accept-encoding:** Lists supported compression algorithms.
        - **accept-language:** Lists preferred human languages with specified quality values.
        - **Response Headers:** Server-sent information, e.g., content-encoding indicates GZIP compression.
    - **Step 6:** Close the browser.

4. Understanding ASP.NET Core.

Ans: Microsoft ASP.NET Core is part of a history of Microsoft technologies used to build websites and services that have evolved over the years:

- **Active Server Pages (ASP) - 1996:**
    - First attempt for server-side website code.
    - Used ASP files with a mix of HTML and VBScript language.
- **ASP.NET Web Forms - 2002:**
    - Came with .NET Framework.
    - Aimed at non-web developers, allowing visual development with drag-and-drop.
    - Used Visual Basic or C# for event-driven code.
    - Not recommended for new projects; ASP.NET MVC is preferred.
- **Windows Communication Foundation (WCF) - 2006:**
    - Released for building SOAP and REST services.
    - SOAP is complex, so use it only for advanced features like distributed transactions.
- **ASP.NET MVC - 2009:**
    - Introduced to cleanly separate concerns of web development into models, views, and controllers.
    - Improved code reuse and unit testing.
- **ASP.NET Web API - 2012:**
    - Allowed developers to create simpler and more scalable HTTP services (REST services) compared to SOAP.
- **ASP.NET SignalR - 2013:**
    - Enabled real-time communication in websites.
    - Abstracted technologies like WebSockets for features like live chat or real-time updates.
- **ASP.NET Core - 2016:**
    - Released in 2016, combining modern implementations of .NET Framework technologies.
    - Includes MVC, Web API, and SignalR.

- o Supports newer technologies like Razor Pages, gRPC, and Blazor.
- o Runs on modern .NET and is cross-platform, meaning it can run on different operating systems.

5. Classic ASP.NET versus modern ASP.NET Core.
Ans:

ASP.NET Core	ASP.NET
Cross-platform, runs on Windows, Linux, and macOS	Windows only
High performance, optimized for modern cloud-based apps	Good performance, but not as optimized as ASP.NET Core
Highly modular, you only include what you need	Less modular, includes a lot of libraries by default
Uses a more flexible configuration system that can read from JSON, XML, or INI files	Primarily uses the Web.config file for configuration
Built-in support for dependency injection	Requires additional packages or frameworks for dependency injection
Can be hosted in IIS, Nginx, Apache, Docker, or self-host in your own process	Primarily hosted in IIS
Uses a middleware pipeline for handling requests and responses	Uses HttpHandlers and HttpModules for handling requests and responses
Unified framework for MVC and Web API	Separate frameworks for MVC and Web API
Built-in support for environment-based configuration	Requires additional packages or custom solutions for environment-based configuration
Command line tools and Visual Studio integration	Primarily Visual Studio integration
Can be updated via NuGet packages	Updates typically come with .NET Framework updates
Fully open source	Partially open source

6. Creating an empty ASP.NET Core project.
Ans:

Step 1: Open visual studio 2022.

Step 2: Click on create a new project.

Step 3: Select ASP.NET Core Empty and click Next.

Step 4: Give the project name as *PracticalApps* and click Next.

Step 5: Select .NET 6.0 (Long Term Support) as the Framework and click Create.

Step 6: In Solution Explorer, toggle Show All Files.

Step 7: Expand the *obj* folder, expand the *Debug* folder, expand the *net6.0* folder, select the *PracticalApps.GlobalUsings.g.cs* file, and observe the implicitly imported namespaces include all the ones for a console app or class library, as well as some ASP.NET Core ones, such as Microsoft.AspNetCore.Builder, as shown in the following code:

```
// <auto-generated/>
global using global::Microsoft.AspNetCore.Builder;
global using global::Microsoft.AspNetCore.Hosting;
global using global::Microsoft.AspNetCore.Http;
global using global::Microsoft.AspNetCore.Routing;
global using global::Microsoft.Extensions.Configuration;
global using global::Microsoft.Extensions.DependencyInjection;
global using global::Microsoft.Extensions.Hosting;
global using global::Microsoft.Extensions.Logging;
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Net.Http.Json;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Step 8: Collapse the *obj* folder, open *Program.cs* file and observe the following:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

- **WebApplication.CreateBuilder(args):** Sets up a builder for the web application, taking command-line arguments.
- **builder.Build():** Builds the web application using the configured builder.
- **app.MapGet("/", () => "Hello World!"):** Maps a GET request to the root ("/") URL and responds with "Hello World!".
- **app.Run():** Runs the web application.

Step 9: Add `Console.WriteLine("This executes after the web server has stopped!");` below the Run method. Stop the server by pressing Ctrl + C and you would get an output just like below:

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7119
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5184
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\PracticalApps\PracticalApps\
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...
This executes after the web server has stopped!

D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\PracticalApps\PracticalApps\bin\Debug\net6.0\Practical
Apps.exe (process 13640) exited with code 0.
Press any key to close this window . . .
```

7. Testing and securing the website.

Ans: To test the functionality of the ASP.NET Core Empty website project and enable encryption of all traffic between the browser and web server for privacy by switching from HTTP to HTTPS, the following steps are to be followed:

- In the toolbar, switch the Web Browser (Microsoft Edge) to Google Chrome.
- Navigate to Debug | Start Without Debugging.
- The first time you start a secure website, you will be prompted that your project is configured to use SSL, and to avoid warnings in the browser you can choose to trust the self-signed certificate that ASP.NET Core has generated. Click Yes.
- When you see the Security Warning dialog box, click Yes again.

- In Chrome, show Developer Tools, and click the Network tab.
- Enter the address *http://localhost:5184/*, or whatever port number was assigned to HTTP, and observe that the response is Hello World! in plain text, from the cross-platform Kestrel web server.
- Enter the address *https://localhost:7119/*, or whatever port number was assigned to HTTPS, and observe if you clicked No when prompted to trust the SSL certificate, then the response is a privacy error. You will see this error when you have not configured a certificate that the browser can trust to encrypt and decrypt HTTPS traffic.
- At the command line or in TERMINAL, press *Ctrl + C* to shut down the web server.
- If you need to trust a local self-signed SSL certificate, then at the command line or in TERMINAL, enter the *dotnet dev-certs https --trust* command.
- Trusting the HTTPS development certificate was requested, and you might be prompted to enter your password and a valid HTTPS certificate may already be present.
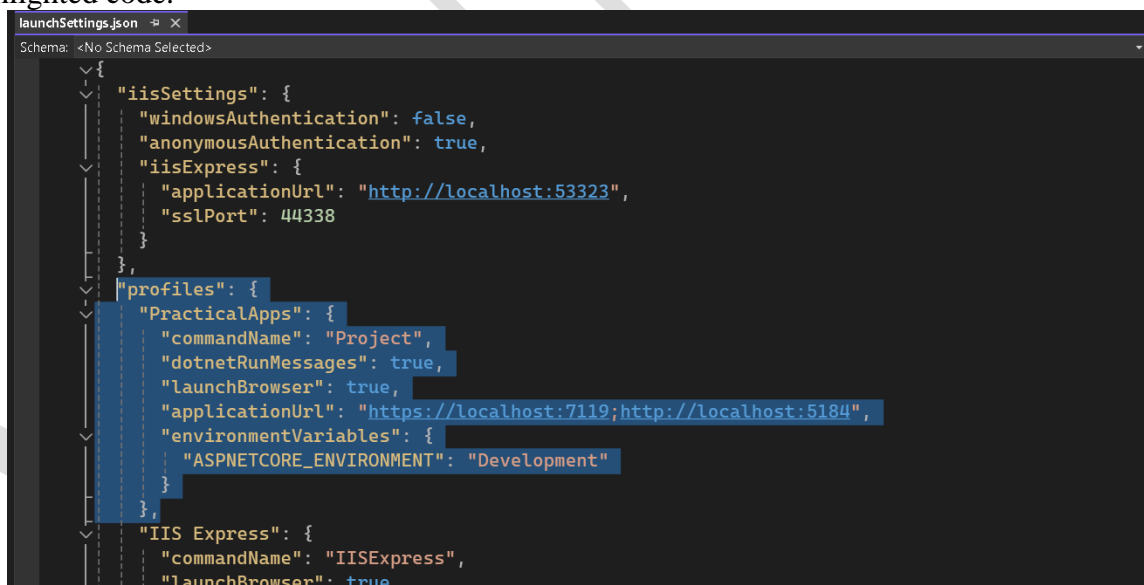
8. Controlling the hosting environment.

Ans: In earlier versions of ASP.NET Core, the project template set a rule to say that while in development mode, any unhandled exceptions will be shown in the browser window for the developer to see the details of the exception.

```
if (app.Environment.IsDevelopment()) {
    app.UseDeveloperExceptionPage();
}
```

With ASP.NET Core 6 and later, this code is executed automatically by default, so it is not included in the project template. ASP.NET Core can read from environment variables to determine what hosting environment to use, for example, DOTNET_ENVIRONMENT or ASPNETCORE_ENVIRONMENT. You can override these settings during local development:
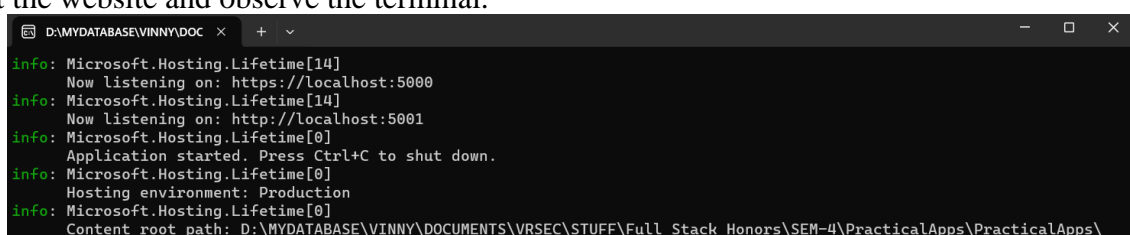
- Expand the folder named *Properties*, open the file named *launchSettings.json* and observe the highlighted code.

- Change the randomly assigned port numbers for HTTP to 5000 and HTTPS to 5001.
- Change the environment to Production. Optionally, change *launchBrowser* to false to prevent Visual Studio from automatically launching a browser.
- Start the website and observe the terminal.

- Shut down the web server.
- In *launchSettings.json*, change the environment back to Development.

9. Separating configuration for services and pipeline.

Ans: In more complex web projects, instead of putting all the initialization code in *Program.cs*, it's often better to use a separate Startup class.
- This class has two methods:
  - **ConfigureServices(IServiceCollection services)**: Adds necessary services like Razor Pages, CORS support, or a database context to the dependency injection container.
  - **Configure(IApplicationBuilder app, IWebHostEnvironment env)**: Sets up the HTTP pipeline for handling requests and responses. It uses the **Use** methods on the **app** parameter to define the order in which features are processed.
- These methods are automatically called by the runtime and this separation helps organize and manage the configuration of a web application more effectively.

Ex:

Startup.cs:
```
using IHostingEnvironment = Microsoft.Extensions.Hosting.IHostingEnvironment;

namespace PracticalApps
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
        }

        [Obsolete]
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (!env.IsDevelopment()) {

                app.UseHsts();
            }
            app.UseRouting();

            app.UseHttpsRedirection();

            app.UseEndpoints(endpoints => {
                endpoints.MapGet("/", ()=> "Hello World!");
            });
        }
    }
}
```

Program.cs:
```
using PracticalApps;

Host.CreateDefaultBuilder(args).ConfigureWebHostDefaults(webBuilder =>
{
    webBuilder.UseStartup<Startup>();
}).Build().Run();

Console.WriteLine("This executes after the web server has stopped!");
```

10. Enabling a website to serve static content.

Ans: A website should not only return plain text but also include static HTML pages, CSS for styling, and other resources like images, which are conventionally stored in a directory named *wwwroot* to separate them from dynamic parts of the project. Follow the steps below:

- Create a folder named *wwwroot*.
- Add a new HTML page file to the *wwwroot* folder named *index.html*.
- Modify its content to link to CDN-hosted Bootstrap for styling and use modern good practices such as setting the viewport as shown in the code below:

```html
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1 " />
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css
"
        rel="stylesheet" integrity="sha384-
KyZXEAg3QhqLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2P0p/We"
crossorigin="anonymous">
    <title>Welcome ASP.NET Core!</title>
</head>
<body>
    <div class="container">
        <div class="jumbotron">
            <h1 class="display-3">Welcome to Northwind B2B</h1>
            <p class="lead">We supply products to our customers.</p>
            <hr />
            <h2>This is a static HTML page.</h2>
            <p>Our customers include restaurants, hotels, and cruise
lines.</p>
            <p>
                <a class="btn btn-primary"
                    href="https://www.asp.net/">Learn more</a>
            </p>
        </div>
    </div>
</body>
</html>
```

- To make the website accessible for static files like *index.html* and avoid a 404 Not Found error, you need to explicitly configure static file handling, set default files, and modify the registered URL path that currently returns the plain text "Hello World!" response.
- Make sure to modify the *Startup.cs* file as shown below:

```csharp
using IHostingEnvironment = Microsoft.Extensions.Hosting.IHostingEnvironment;

namespace PracticalApps
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
```

```
        }

        [Obsolete]
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (!env.IsDevelopment()) {

                app.UseHsts();
            }
            app.UseRouting();

            app.UseHttpsRedirection();

            app.UseDefaultFiles();

            app.UseStaticFiles();

            app.UseEndpoints(endpoints => {
                endpoints.MapGet("/hello", ()=> "Hello World!");
            });
        }
    }
}
```

- Start the website.
- Start Chrome and show Developer Tools.
- In Chrome, enter *http://localhost:5000/* and observe that you are redirected to the HTTPS address on port 5001, and the index.html file is now returned over that secure connection because it is one of the possible default files for this website.
- In Developer Tools, observe the request for the Bootstrap stylesheet.
- In Chrome, enter *http://localhost:5000/hello* and observe that it returns the plain text Hello World! as before.
- Close Chrome and shut down the web server.