Unit 3

Overview of Asp.Net

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

Difference between Asp and Asp.Net

What do you mean by ASP?

ASP stands for Active Server Pages. It is a framework to develop web pages. Microsoft introduced ASP as the first server-side scripting language in 1998 that was used for executing scripts with extension .asp on a web server. ASP is not a language. Rather it is a technology.

What do you mean by ASP.NET?

ASP.NET came as a successor to ASP in 2002. A typical ASP.NET application includes files (web.config) and global file (.asax), Code behind file (.aspx.vb), is written in .aspx file, which code-behind code for a web page is written in a separate class file, .aspx.vb. It is used to separate the user interface display from the UI processing. Web.config is a text file used to store application-level settings required for state management, security, etc. Global file is an optional file containing application-level events such as Application_Init, Application_Start, Application_BeginRequest, Application_EndRequest, Application_AuthenticationRequest, Application_Error, Session_Start, Session_End etc.

Following are the head to head differences between the ASP and ASP.NET in detail.

Factor	ASP	ASP.NET
1. Object-Oriented Programming	ASP does not support object-oriented programming.	ASP.NET supports object-oriented programming.
2. Purpose	The server-side scripting engine is also known as a scripting language. Code is written using VB script and HTML for developing web applications.	It supports many programming languages like VB, C#, Python and many more for developing web applications.
3. File Extension	ASP Pages have the file extension .asp.	ASP.NET Pages have the file extension .aspx.
4. Inheritance	ASP does not support the concept of inheritance.	ASP.NET inherit the class written in code behind.
5. Compilation/ Interpretation	ASP has interpreted language.	ASP.NET is Compiled and Interpreted language.
6. Debugging	ASP debugging is difficult as ASP scripts are interpreted.	In ASP.NET debugging is relatively easy.
7. Validation	ASP does not have the built-in facility of validation.	ASP.NET provides built-in validation and controls.
8. Exception Handling	In ASP, exception handling is very poor.	ASP.NET supports the concept of exception handling.
9. Execution	ASP works on IIS.	ASP.NET works on non-Microsoft platforms also. Cassini, a Microsoft web server, can be integrated with Apache.
10. XML (Extensible Mark Up Language)	ASP does not support XML.	ASP.NET support XML.
11. ADO (Active X Database objects)	ADO is a simple COM object and has fewer features.	ADO.NET provides full support to get data from multiple data sources through numerous built-in classes.

12. Code Behind	ASP does not support code-behind.	ASP.NET supports and separates HTML code and .NET language code.
13. Configurable	ASP does not support configurable files.	ASP.NET support configurable file. I.e. it uses a web.config file.
14. Custom Controller	ASP does not have a provision for custom controls.	ASP.NET support @register directive to create custom controls

Architecture of Asp.Net

ASP.NET (Active Server Pages .NET) is a popular framework for building web applications and web services developed by Microsoft. It's a part of the larger .NET ecosystem and provides a robust and scalable platform for web development. ASP.NET applications follow a specific architecture that includes various components and layers:

Client-Side: This is the user's web browser or client device. It interacts with the server-side components to request and display web pages and other resources.

Web Server: The web server, often IIS (Internet Information Services) for Windows-based deployments, receives HTTP requests from clients and routes them to the appropriate components within the ASP.NET application.

ASP.NET Runtime: The ASP.NET runtime is responsible for managing the execution of ASP.NET applications. It includes the Common Language Runtime (CLR), which allows ASP.NET to run managed code, and the ASP.NET HTTP runtime, which processes incoming HTTP requests and manages the ASP.NET page life cycle.

Web Forms/Page Framework: ASP.NET Web Forms is a framework for building web applications using a visual, event-driven programming model. Developers can create web pages that closely resemble desktop user interfaces, making it easier to build complex and interactive web applications.

MVC (Model-View-Controller): ASP.NET also supports the Model-View-Controller (MVC) architectural pattern, which provides a more structured and testable way to build web applications. In MVC, the application is divided into three main components:

Model: Represents the application's data and business logic.

View: Defines the presentation layer, responsible for rendering the user interface.

Controller: Manages the interaction between the model and view, handling user input and controlling the flow of the application.

Web API: ASP.NET Web API is a framework for building HTTP-based RESTful services. It allows developers to create APIs that can be consumed by web applications, mobile apps, and other clients.

SignalR: SignalR is a library for adding real-time functionality to web applications. It enables bi-directional communication between the server and clients, making it possible to build features like chat applications, live updates, and notifications.

Routing: ASP.NET uses routing to map URLs to specific resources or controller actions. This allows for clean and user-friendly URLs, making it easier to organize and navigate web applications.

Authentication and Authorization: ASP.NET provides built-in support for authentication and authorization, allowing developers to secure their applications using various authentication providers, such as Windows Authentication, Forms Authentication, and OAuth.

State Management: ASP.NET offers various options for managing state in web applications, including client-side state management (e.g., cookies, local storage) and server-side state management (e.g., session state, application state, and caching).

Data Access: ASP.NET applications often interact with databases and other data sources. ADO.NET and Entity Framework are commonly used for data access and ORM (Object-Relational Mapping).

Error Handling and Logging: ASP.NET provides mechanisms for handling errors gracefully and logging relevant information for troubleshooting and monitoring purposes.

Middleware: ASP.NET Core, the latest version of ASP.NET, introduced the concept of middleware, which allows developers to configure and add custom components to the request pipeline. Middleware can handle cross-cutting concerns such as authentication, routing, and caching.

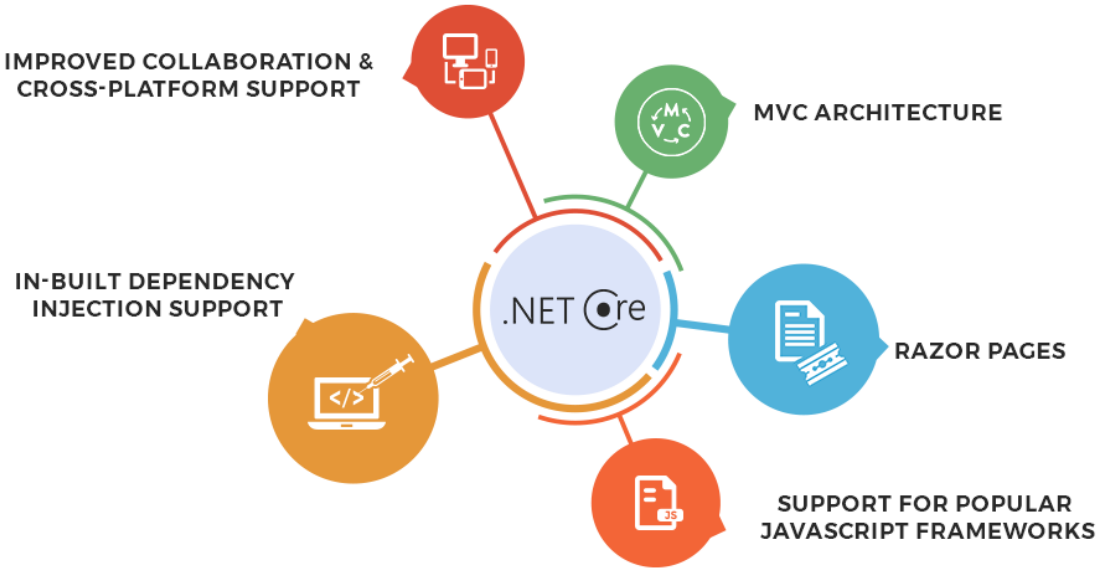
Dependency Injection: ASP.NET Core emphasizes dependency injection, making it easier to manage and inject dependencies into application components, promoting modularity and testability.

Hosting: ASP.NET applications can be hosted on various platforms, including Windows servers, cloud services (e.g., Azure), and containers (e.g., Docker). ASP.NET Core also supports cross-platform hosting.

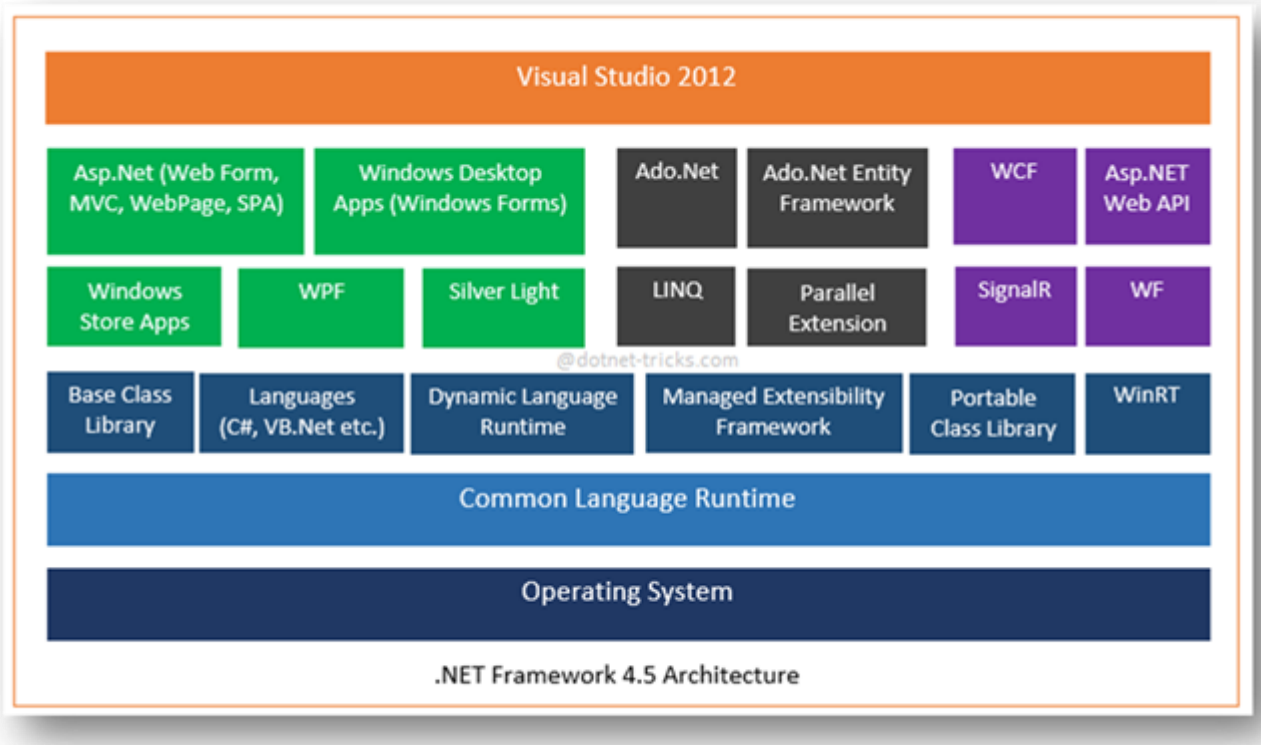
Front-End Technologies: ASP.NET applications often use front-end technologies like HTML, CSS, JavaScript, and popular front-end frameworks like Angular, React, or Vue.js to create rich and interactive user interfaces.

ASP.Net is a platform produced by Microsoft for web development. It is used for creating web-based applications. The First ASP.NET was published in the year 2002.
The first version of ASP.Net deployed was 1.0. The latest version of ASP.Net is version 4.6. ASP.Net is designed to work with the HTTP protocol. This is the standard protocol used overall in every web application. With ASP.NET Core, you can:

- Build web applications and services, IoT apps, and mobile backends.
- It supports development tools on Windows, macOS, and Linux.
- Deploy to the cloud or on-premises. Run-on .NET Core or .NET Framework.



Here, the basic architecture of ASP.Net:-



1. **Language** – A Collection of languages exists for the .net framework. The .NET framework can work with different programming languages such as C#, VB.NET, C++, and F#, which can be helpful in developing web applications.

2. **Library** – The .NET Framework includes a set of standard class libraries. The web library is the most common library for web applications in.net. The software library has all the modules that are needed to develop web-based applications.
3. Common Language Runtime - The CLI is a platform, which is used for implementing key activities. Activities include Exception handling and Garbage collection.

Characteristics of ASP.Net

Code Behind Mode – This is the principle of software and code isolation. Maintaining the ASP.Net framework is simple by allowing this separation. An ASP.Net file's general file type is aspx. Suppose we've got a MyPage.aspx web page. The file named MyPage.aspx.cs will be open, which would represent the page's code portion. Therefore, Visual Studio creates separate files for each web page, one for the designing part and the other for the coding.

State Management – ASP.Net has a state management control centre. HTTP is regarded as a stateless protocol. Let's take the example of a shopping cart request. Now, when a client chooses what they want to purchase from the page, they must push the submit button.
The software should recall the purchase items selected by the user. This is known as remembering a request's status at a current time point. The HTTP protocol is stateless. HTTP will not store the data on the cart products when the user goes to the shopping site. To ensure that the cart products are delivered to the checkout site, additional coding must be performed. Such an implementation can become complicated at times. This implementation can sometimes become complicated. But on your side, ASP.Net can do government administration. ASP.Net can, therefore, recall the products in the cart and pass it on to the checkout page.

Caching –ASP.Net is able to implement the Caching framework. It increases application performance. It is possible to store in a temporary location by caching certain pages that are often requested by the user. These pages can be retrieved more quickly, and the user can receive better answers. And caching can significantly enhance an application's performance.

Execution Model of Asp.Net

The execution model of ASP.NET, specifically in the context of ASP.NET Web Forms and ASP.NET Core, describes how the framework processes incoming HTTP requests, handles the application's logic, and generates the responses to be sent back to the client. Below, I'll provide an overview of the execution model for both ASP.NET Web Forms and ASP.NET Core.

ASP.NET Core Execution Model:

HTTP Request Processing: ASP.NET Core uses a middleware-based approach. When an HTTP request is received, it's processed by a pipeline of middleware components. Each middleware component can inspect, modify, or pass the request to the next component.

Routing: ASP.NET Core has built-in routing that maps incoming URLs to controller actions. Routes are configured in the Startup.cs file.

Controller Execution: In ASP.NET Core MVC (Model-View-Controller), controllers handle incoming requests. Controllers are responsible for processing the request, invoking the appropriate methods, and returning responses.

Model Binding: ASP.NET Core automatically binds incoming request data to model classes, making it easier to work with form data, query parameters, and JSON payloads.

Middleware Continuation: After the controller has processed the request, control returns to the middleware pipeline. Additional middleware components can be executed before generating a response.

View Rendering (Optional): In MVC, a controller action typically returns a view. ASP.NET Core uses Razor views or other templating engines to generate dynamic HTML content.

Response Generation: Once the view has been rendered or the controller has generated the response data, the ASP.NET Core runtime packages the response and sends it to the client.

HTTP Response: The HTTP response, including headers and content, is sent to the client's browser, which renders the web page or processes the response data as needed.
Both ASP.NET Web Forms and ASP.NET Core offer different approaches to building web applications, and the execution models reflect these differences. ASP.NET Core provides more flexibility and modularity, making it suitable for modern web development, while Web Forms follows a more event-driven and stateful model. The choice between them depends on the project's requirements and architectural preferences.

Difference between Code behind and .aspx file

In ASP.NET Web Forms, the code for a web page is typically split into two separate files: the .aspx file and the code-behind file (.cs or .vb). These files serve distinct purposes and are part of the Web Forms development model. Here are the key differences between the .aspx file and the code-behind file:

Purpose:

.aspx File: The .aspx file is responsible for defining the structure and layout of the web page. It contains the HTML markup, server controls, and visual elements that determine how the page will be displayed in the browser. It also includes declarative markup for defining event handlers.

Code-Behind File: The code-behind file (.cs or .vb) contains the server-side logic and event-handling code associated with the .aspx file. It includes the C# or VB.NET code that responds to user interactions and controls the behavior of the page. This file typically contains event handlers for buttons, data-binding code, and other server-side logic.

Content:

.aspx File: The .aspx file primarily consists of HTML markup and server controls, along with declarative event handling code using attributes like OnClick or OnTextChanged. It can also include data-binding expressions and client-side JavaScript code.

Code-Behind File: The code-behind file contains server-side code written in C# or VB.NET. This code is responsible for handling events raised by server controls (e.g., button clicks, page load) and performing tasks such as database operations, data manipulation, and business logic.

Separation of Concerns:

.aspx File: The .aspx file is concerned with the presentation and structure of the web page. It focuses on the user interface and layout aspects.

Code-Behind File: The code-behind file separates the presentation from the logic. It houses the server-side code responsible for handling user interactions and implementing the functionality of the web page. This separation promotes a clean and maintainable code structure.

Maintainability:

.aspx File: Since the .aspx file primarily deals with markup and user interface elements, it tends to be more concise and easier to understand, making it suitable for designers and front-end developers.

Code-Behind File: The code-behind file contains the application's logic and can be more complex. Developers focus on implementing the server-side functionality, which may include database interactions, data processing, and business rules.

Reusability:

.aspx File: .aspx files can be reused by referencing them in multiple code-behind files. This allows you to have different code-behind files with different logic while using the same .aspx file for layout.

Code-Behind File: Code-behind files can be reused by associating them with different .aspx files. This enables you to separate the logic from the presentation and apply the same logic to different pages.

In summary, the .aspx file in ASP.NET Web Forms is responsible for defining the page's layout and visual elements, while the code-behind file contains the server-side logic and event-handling code. This separation of concerns enhances maintainability, readability, and reusability in web application development.

Implementation of Simple Web Application

Creating a simple web application involves several steps, and I'll provide a basic example using ASP.NET Core and C#. In this example, we'll create a "Hello World" web application that displays a greeting message on a web page. Make sure you have the necessary development tools installed, including Visual Studio or Visual Studio Code, and .NET Core.

Step 1: Create a New ASP.NET Core Web Application

Open Visual Studio or Visual Studio Code.

Create a new project and select "ASP.NET Core Web Application" as the project template.

Choose a project name and location, then click "Create."

In the "Create a new ASP.NET Core web application" dialog, select the "Web Application" template and ensure that ASP.NET Core 3.1 or later is selected.

Click "Create" to generate the project.

Step 2: Define a Simple Controller and View

In the created project, open the "Controllers" folder and create a new controller called "HomeController.cs" if it doesn't already exist. This controller will handle requests and return views.

```
using Microsoft.AspNetCore.Mvc;

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

Step 3: Next, create a "Views" folder (if not already present) in the project's root directory. Inside the "Views" folder, create a folder named "Home" and add a new view file named "Index.cshtml" inside it.

```
<!-- Views/Home/Index.cshtml -->

<!DOCTYPE html>

<html>

<head>

    <title>Hello World</title>
```

```
</head>
<body>
  <h1>Hello, World!</h1>
</body>
</html>
```

Step 4: Configure Routing

In the "Startup.cs" file of your project, ensure that routing is configured correctly in the Configure method. By default, the "MapRoute" method is used for routing.

```
app.UseEndpoints(endpoints =>
{
  endpoints.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Step 5: Run the Application

Build and run the application by pressing F5 in Visual Studio or using the dotnet run command in the terminal.

Your web application will start, and you can access it in your browser at the specified URL (usually https://localhost:5001 or https://localhost:5000).

You should see the "Hello, World!" message displayed on the webpage.

Congratulations! You've created a simple web application in ASP.NET Core. This example demonstrates the basic structure of an ASP.NET Core web application, including controllers, views, and routing. You can expand upon this foundation to build more complex web applications with additional features and functionality.