Introduction to ASP.NET

.NET

.NET is a cross-platform, open source, general purpose development platform for building many different types of applications. With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, and IoT.

.NET Core

.NET Core is an open-source, general-purpose development platform. You can create .NET Core apps for Windows, macOS, and Linux using multiple programming languages. Frameworks and APIs are provided for cloud, IoT, client UI, and machine learning.

Without a doubt, it presents the most advanced, mature, and extensive class libraries, common APIs, multi-language support, and tools. What's better? The Visual Studio 2019 and Visual Studio Code are currently the most advanced and modern developer IDEs. This makes .NET Core one of the most productive platforms in .NET Core development for developers.

.NET Core was first developed by Microsoft. As of now it is managed under the .NET Foundation which is a non-profit open source organization. .The framework has been written in C# and C++ and is licensed under MIT license.

The prime version of .NET Core, 1.0, was first released back in 2016 and without a doubt had very limited functionality. The next version came in August of 2017, .NET Core 2.0. And the story continues after that.

Characteristics of .NET Core

Even though the first version had limited functionalities, the latest version of .NET Core has extensive capabilities with modern and powerful features. Some of the amusing characteristics of .NET Core include open source, cross-platform, modern, flexible, lightweight, fast, friendly, shareable, and built for future software development.

Being Free and Open Source

The entire .NET Core platform is free and open source, giving expert developers a vast playground where they can develop futuristic enterprise-grade applications. You can easily .NET Core source code project is available on Github. A huge number of active developers participate in .NET Core development with a drive to improve the existing features, adding new features, and fixing bugs and issues.

.NET Core is Modern

Shifting away from older frameworks and functionalities, the .NET Core framework is built to solve today's modern needs, some of them include being mobile friendly, building it once and running it everywhere on all platforms, scalable, and high performance.

It is Cross-platform

The framework supports and runs on Windows, macOS, and Linux operating systems.It is also amazingly consistent across architecture including x64, x86, and ARM. You can import and use the same assemblies and libraries on multiple platforms.

.NET Core supports modern language constructs using C# version 8, like object-oriented and modular programming, generics, collections, lambdas, Language Integrated Query (LINQ), and asynchronous programming.

Speed

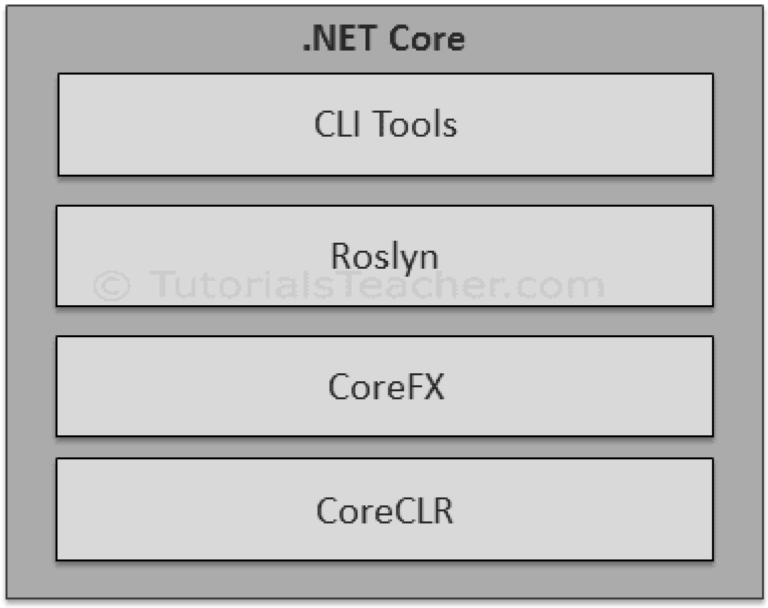
When compared to the .NET Framework and .NET Core 2.2 and previous versions, .NET Core 3.1, the latest version is superiorly fast. Also, it is faster than compared to current server-side frameworks like Java Servlet and Node.js.

Lightweight and Friendly

You will find that the framework is extremely lightweight; it can be included in your app or installed side-by-side user, machine-wide, or on a server. You can deploy .NET Core in Docker containers.

Additionally, the .NET Core framework is aptly compatible with .NET Framework, Xamarin, and Mono, via .NET Standard. You will also get support while working with other popular Web frameworks and libraries like React, Angular, and JavaScript.

.NET Core Composition



CLI Tools: A set of tooling for development and deployment.

Roslyn: Language compiler for C# and Visual Basic

CoreFX: Set of framework libraries.

CoreCLR: A JIT based CLR (Command Language Runtime).

Where to use .NET Core

Command-line applications

At the most basic level, .NET Core is a console applications that can be interested with via a command-line interface, such as Windows Command Prompt or Linux Terminal. So, if you need to build an OS-independent command-line application, .NET Core should be your choice.

Of course, you can do an OS-independent command-line application in mono as well. But why bother, if .NET Core gives you a much better project file structure and system libraries that are way easier to work with?

Web applications

ASP.NET Core, the set of libraries that enable development of web applications on top of .NET Core, is way easier to work with than mono equivalent, while it can do everything mono can and more.

Through trial and error, Microsoft developers made it possible to set up basic web applications in minutes with easily configurable middleware pipilene.

Also, with revamped version of SignalR library and server-side Blazor, you can now maintain a persistent real-time two-way communication between your client and server components with very little code required.

Background services

Any application that runs continuously in the background and doesn’t have GUI can be classed as a service. And .NET Core allows you to build such applications just as easily as you can build some basic console apps.

Desktop applications

This point would be controversial, as .NET Core doesn’t have in-built way of developing cross-platform GUI-enabled desktop applications.

However, if you only intend to run your desktop app on Windows, then .NET Core 3 provides support for this. And, just like with everything else in .NET Core, the experience of creating this application type has been vastly improved compared to how it was done in .NET Framework.

On the other hand, if you want to make your desktop application for multiple operating systems, .NET Core may still be the best choice. There is a great cross-platform JavaScript framework for building desktop applications called Electron.js and with Electron.NET NuGet package, you can integrate it with your .NET Core app.

The unique selling point of this approach is that your entire GUI is done in HTML, CSS and JavaScript, just like a web page. The only difference is that all of it is hosted on your machine inside the desktop application itself. And, instead of being displayed in the browser, it’s displayed in its own window.

Because front-end technologies have massively evolved over the years, the possibilities of what you can do inside of your GUI are limitless. You can use absolutely any JavaScript library or framework available. Therefore, if you have seen how easy it can be to create web pages with an impressive layout, you can do it just as easily inside your desktop application.

First of all, your back-end code will be compiled. So, if you don't want to have your code accessible to the users, you won't have to. Likewise, in this scenario, your .NET code can do any background logic it would normally do on the server of a web application.

Essentially, Electron.NET application is not radically different from a standard ASP.NET application. The only differences are that both back-end and front-end parts are hosted in the same process.

Mono

Microsoft Released .NET framework it opened a vast possibility of creating powerful, flexible and robust software. But that was only intended for the Windows platform. What about the Others? — Then comes the Mono as an open source project in 2001 which brings the .NET Compatible framework classes to other platforms including its own c# compiler and CLR(Common Runtime Language). Mono supports almost all .NET features and still increasing its features. The stated purpose of Mono is not only to be able to run Microsoft .NET applications cross-platform, but also to bring better development tools to Linux developers.

Mono can be run on many software systems including Android, most Linux distributions, BSD, macOS, Windows, Solaris, and even some game consoles such as PlayStation 3, Wii, and Xbox 360.

Mobile app development

Using .NET for mobile app development was always focus for Xamarin since its inception, this is where you will be using mono. That is, of course, unless you opt for writing the apps in their native languages, which would be either Kotlin or Java for Android and either Swift or Objective-C for iOS.

If you do opt to go native, however, you will be only able to write your code for one OS at a time. With mono-based Xamarin tools, on the other hand, you can have a single code base, which you can compile for both Android and iOS.

.NET Core, on the other hand, doesn't come with anything that will allow you to write mobile apps. It's never been intended for this purpose.

Game development

Unity gaming engine uses mono inside of it, so mono is the official way of getting your .NET code into games.

.NET Core, on the other hand, was never intended for gaming. So, unless you want to build a text-based adventure or some relatively basic browser-based game, .NET Core is of no use.

Compiled code inside browser

WebAssembly allows you to run a compiled code inside browser. Unlike other technologies, such as Silverlight, Java and Flash that required vendor-specific browser plugins, WebAssembly is a web standard. And as such, it's supported by all major browsers.

Mono has in-built capability to be compiled into WebAssembly-compatible packages. And, as such, it was a natural choice for the base framework behind client-side Blazor.

.NET Core doesn't have such capability natively. Although it was initially used for Blazor project, it was abandoned in favour of mono.

Multi-platform desktop applications

Desktop applications on operating systems other than Windows was the reason why mono was developed in the first place. Since then, countless frameworks and libraries were built around it to accommodate GUI-based application development on Windows, Linux, Mac OS and even Solaris.

Having said that, there are situations where .NET Core is a better choice for desktop application development. More on this later.

The main scenarios where mono would be a better choice for desktop application development than .NET Core is if you have limited knowledge of JavaScript or if you want your application to run on Solaris, which .NET Core doesn't support.

ASP.NET

ASP.NET is an open-source, server-side web-application framework designed for web development to produce dynamic web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, applications and services.. ASP.NET extends the .NET developer platform with tools and libraries specifically for building web apps

Difference Between .NET and ASP.NET Framework

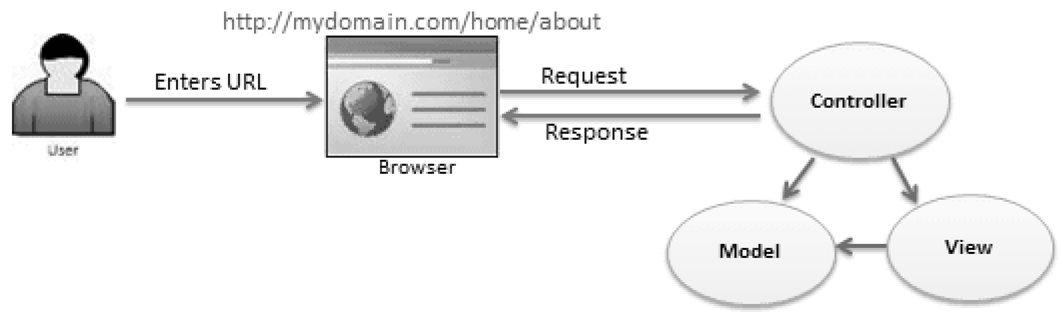
	.NET	ASP.NET
1.	.NET is a software development framework aimed to develop Windows, Web and Server based applications.	ASP.NET is a main tool that present in the .NET Framework and aimed at simplifying the creation of dynamic webpages.
2.	Server side and client side application development can be done using .NET framework.	You can only develop server side web applications using ASP.NET as it is integrated with .NET framework.
3.	Mainly used to make business applications on the Windows platform.	It is used to make dynamic web pages and websites using .NET languages.
4.	Its programming can be done using any language with CIL (Common Intermediate Language) compiler.	Its programming can be done using any .NET compliant language.

ASP.NET MVC

ASP.NET gives you a powerful, patterns-based way to build dynamic websites using the MVC pattern that enables a clean separation of concerns.

MVC is a design pattern used to decouple user-interface (view), data (model), and application logic (controller). This pattern helps to achieve separation of concerns.

Using the MVC pattern for websites, requests are routed to a Controller that is responsible for working with the Model to perform actions and/or retrieve data. The Controller chooses the View to display, and provides it with the Model. The View renders the final page, based on the data in the Model.



Model

The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it. 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.

View

Views are responsible for presenting content through the user interface. They use the Razor view engine to embed .NET code in HTML markup. There should be minimal logic within views, and any logic in them should relate to presenting content. If you find the need to perform a great deal of logic in view files in order to display data from a complex model, consider using a View Component, ViewModel, or view template to simplify the view.

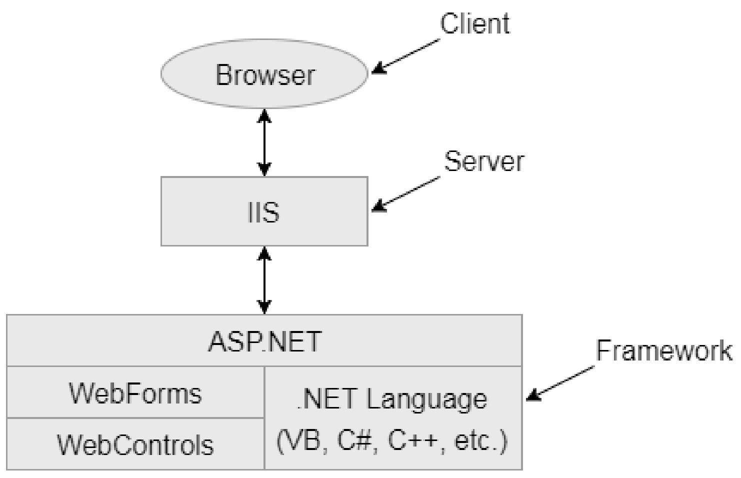
Controller

Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. In the MVC 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).

ASP.NET Web Forms

ASP.NET Web Forms is a part of the ASP.NET web application framework and is included with Visual Studio. It is one of the four programming models you can use to create ASP.NET web applications, the others are ASP.NET MVC, ASP.NET Web Pages, and ASP.NET Single Page Applications.

Web Forms are pages that your users request using their browser. These pages can be written using a combination of HTML, client-script, server controls, and server code. When users request a page, it is compiled and executed on the server by the framework, and then the framework generates the HTML markup that the browser can render. An ASP.NET Web Forms page presents information to the user in any browser or client device.



The main purpose of Web Forms is to overcome the limitations of ASP and separate view from the application logic.

Web Forms are web pages built on the ASP.NET Technology. It executes on the server and generates output to the browser. It is compatible to any browser to any language supported by .NET common language runtime. It is flexible and allows us to create and add custom controls.

We can use Visual Studio to create ASP.NET Web Forms. It is an IDE (Integrated Development Environment) that allows us to drag and drop server controls to the web forms. It also allows us to set properties, events and methods for the controls. To write business logic, we can choose any .NET language like: Visual Basic or Visual C#.

Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.

ASP.NET Web Forms Features

ASP.NET is full of features and provides an awesome platform to create and develop web application. Here, we are discussing these features of Web Forms.

Server Controls

Master Pages

Working with data

Membership

Client Script and Client Frameworks

Routing

State Management

Security

Performance

Error Handling

Server Controls

Web Forms provides rich set of server controls. These controls are objects that run when the page is requested and render markup to the browser. Some Web server controls are similar to familiar HTML elements, such as buttons and text boxes. It also provides controls that we can use to connect to data sources and display data.

Master Pages

It allows us to create a consistent layout for the pages in our application. This page defines the look and feel and standard behavior that we want for all of the pages in our application. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

Working with Data

In an ASP.NET Web Forms application, we use data-bound controls to automate the presentation or input of data in web page UI elements such as tables and text boxes and drop-down lists.

Membership

Project's Account folder contains the files that implement the various parts of membership: registering, logging in, changing a password, and authorizing access. Additionally, ASP.NET Web Forms supports OAuth and OpenID. These authentication enhancements allow users to log into your site using existing credentials, from such accounts as Facebook, Twitter and Google.

Client Script and Client Frameworks

We can enhance the server-based features of ASP.NET by including client-script functionality in ASP.NET Web Form pages. We can use client script to provide a richer, more responsive user interface to the users. We can also use client script to make asynchronous calls to the Web server while a page is running in the browser.

Routing

We can configure URL routing of our application. A request URL is simply the URL a user enters into their browser to find a page on our web site. We use routing to define URLs that are semantically meaningful to users and that can help with search-engine optimization (SEO).

State Management

ASP.NET Web Forms includes several options that help you preserve data on both a per-page basis and an application-wide basis.

Security

Developing a secure application is most important aspect of software development process. ASP.NET Web Forms allow us to add extensibility points and configuration options that enable us to customize various security behaviors in the application.

Performance

Web Forms provides good performance and allows us to modify performance related to page and server control processing, state management, data access, application configuration and loading, and efficient coding practices.

Debugging and Error Handling

We can diagnose problems that occur in our Web Forms application. Debugging and error handling are well supported within ASP.NET Web Forms so that our applications compile and run effectively.

Inline Code and Code Behind in ASP.NET

A brief overview of the in-line code and Code-behind model in ASP.NET.

Inline Code

Inline Code refers to the code that is written inside an ASP.NET Web Page that has an extension of .aspx. It allows the code to be written along with the HTML source code using a <Script> tag. It's major point is that since it's physically in the .aspx file it's deployed with the Web Form page whenever the Web Page is deployed.

```
<%@ Language=C# %>
<HTML>
    <script runat="server" language="C#">
        void MyButton_OnClick(Object sender, EventArgs e)
        {
            MyLabel.Text = MyTextbox.Text.ToString();
        }
    </script>
    <body>
        <form id="MyForm" runat="server">
            <asp:textbox id="MyTextbox" text="Hello World"
runat="server"></asp:textbox>
            <asp:button id="MyButton" text="Echo Input"
OnClick="MyButton_OnClick" runat="server"></asp:button>
            <asp:label id="MyLabel" runat="server"></asp:label>
        </form>
    </body>
</HTML>
```

Code Behind

Code Behind refers to the code for an ASP.NET Web page that is written in a separate class file that can have the extension of .aspx.cs or .aspx.vb depending on the language used. Here the code is compiled into a separate class from which the .aspx file derives. You can write the code in a separate .cs or .vb code file for each .aspx page. One major point of Code Behind is that the code for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any Inline Server Code.

MyCodebehind.aspx

```
<%@ Language="C#" Inherits="MyStuff.MyClass" %>
<HTML>
    <body>
```

```
        <form id="MyForm" runat="server">
            <asp:textbox id="MyTextBox" text="Hello World"
runat="server"></asp:textbox>
            <asp:button id="MyButton" text="Echo Input" Onclick="MyButton_Click"
runat="server"></asp:button>
            <asp:label id="MyLabel" runat="server" />
        </form>
    </body>
</HTML>
```

Mycodebehind.cs

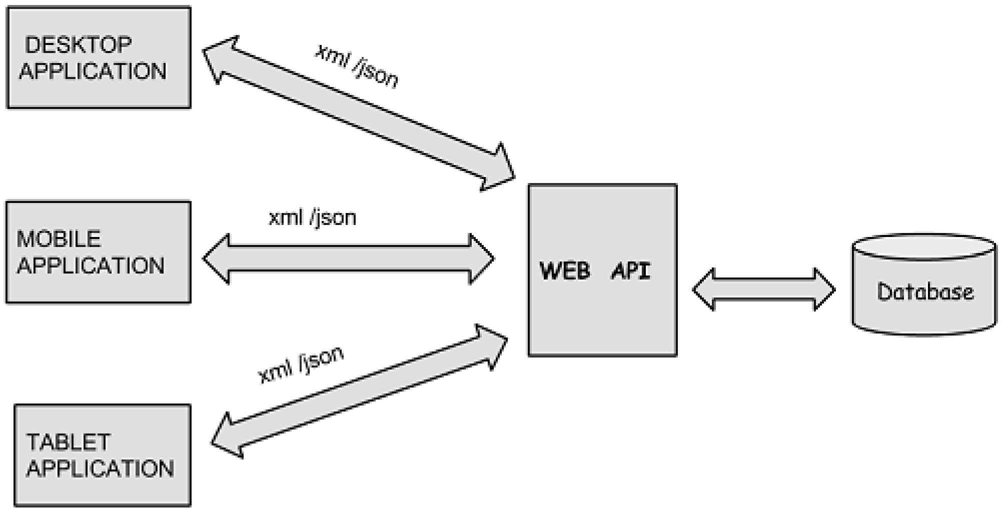
```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace MyStuff
{
    public class MyClass : Page
    {
        protected System.Web.UI.WebControls.Label MyLabel;
        protected System.Web.UI.WebControls.Button MyButton;
        protected System.Web.UI.WebControls.TextBox MyTextBox;

        public void MyButton_Click(Object sender, EventArgs e)
        {
            MyLabel.Text = MyTextBox.Text.ToString();
        }
    }
}
```

ASP.NET Web API

ASP.NET Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Framework. Web clients are not restricted to just web browsers ,lot of different clients are being used to access the data over the internet.Clients such as mobile devices and tablets commonly access the data over the internet.For such broader category of clients ,which needs to access the information over HTTP, some technology specific to the HTTP is required which is what WebAPI provides.



It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view. It is like a webservice or WCF service but the exception is that it only supports HTTP protocol.

ASP.NET Core

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps. With ASP.NET Core, you can:

- Build web apps and services, Internet of Things (IoT) apps, and mobile backends.
- Use your favorite development tools on Windows, macOS, and Linux.
- Deploy to the cloud or on-premises.
- Run on .NET Core.

.NET Architecture and Design Principles

Net Framework Design Principle

The following design principles of the .Net framework is what makes it very relevant to create .Net based applications.

Interoperability - The .Net framework provides a lot of backward support. Suppose if you had an application built on an older version of the .Net framework, say 2.0. And if you tried to run the same application on a machine which had the higher version of the .Net framework, say 3.5. The application would still work. This is because with every release, Microsoft ensures that older framework versions gel well with the latest version.

Portability- Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and Linux.

Security - The .NET Framework has a good security mechanism. The inbuilt security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.

Memory management - The Common Language runtime does all the work or memory management. The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the "Garbage Collector" which runs as part of the .Net framework.

The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.

Simplified deployment - The .Net framework also have tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.