C# and ASP.NET

1. What is the .NET Framework?

.NET and .NET Framework

What is .NET framework? Explain its architecture.

.NET is a developer platform made up of tools, programming languages, and libraries for building many different types of applications.

There are various implementations of .NET. Each implementation allows .NET code to execute in different places—Linux, macOS, Windows, iOS, Android, and many more.

1. **.NET Framework** is the original implementation of .NET. It supports running websites, services, desktop apps, and more on Windows.
2. **.NET** is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS. .NET is open source on GitHub. .NET was previously called .NET Core.
3. **Xamarin/Mono** is a .NET implementation for running apps on all the major mobile operating systems, including iOS and Android.

.NET Standard is a formal specification of the APIs that are common across .NET implementations. This allows the same code and libraries to run on different implementations.

The .NET Framework is a new and revolutionary platform created by Microsoft for developing applications

- It is a platform for application developers
- It is a Framework that supports Multiple Language and Cross language integration.
- IT has IDE (Integrated Development Environment).
- Framework is a set of utilities or can say building blocks of your application system.
- .NET Framework provides GUI in a GUI manner.
- .NET Framework provides interoperability between languages i.e. Common Type System (CTS) .
- .NET Framework also includes the .NET Common Language Runtime (CLR), which is responsible for maintaining the execution of all applications developed using the .NET library.
- The .NET Framework consists primarily of a gigantic library of code.

Definition: A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

Cross Language integration

You can use a utility of a language in another language (It uses Class Language Integration).

.NET Framework includes no restriction on the type of applications that are possible. The .NET Framework allows the creation of Windows applications, Web applications, Web services, and lot more.

The .NET Framework has been designed so that it can be used from any language, including C#, C++, Visual Basic, JScript, and even older languages such as COBOL

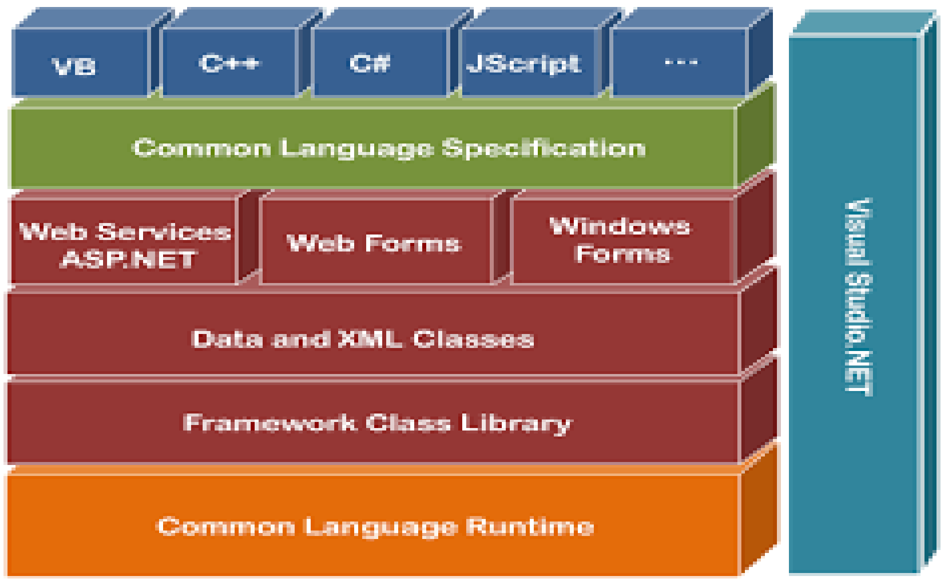
Architecture of .NET Framework

The two major components of .NET Framework are the Common Language Runtime and the .NET Framework Class Library.

- The **Common Language Runtime (CLR)** is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.
- The **Class Library** provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.

.NET applications are written in the C#, F#, or Visual Basic programming language. Code is compiled into a language-agnostic Common Intermediate Language (CIL). Compiled code is stored in assemblies—files with a .dll or .exe file extension.

When an app runs, the CLR takes the assembly and uses a just-in-time compiler (JIT) to turn it into machine code that can execute on the specific architecture of the computer it is running on.



1. **What is CTS? Explain.**

What Does Common Type System (CTS) Mean?

The Common Type System (CTS) is a standard for defining and using data types in the .NETframework. CTS defines a collection of data types, which are used and managed by the run time to facilitate cross-language integration.

CTS provides the types in the .NET Framework with which .NET applications, components and controls are built in different programming languages so information is shared easily. In contrast to low-level languages like C and C++ where classes/structs have to be used for defining types often used (like date or time), CTS provides a rich hierarchy of such types without the need for any inclusion of header files or libraries in the code.

CTS is a specification created by Microsoft and included in the European Computer Manufacturer‘s Association standard. It also forms the standard for implementing the .NET framework.

CTS is designed as a singly rooted object hierarchy with System.Object as the base type from which all other types are derived. CTS supports two different kinds of types:

1. Value Types: Contain the values that need to be stored directly on the stack or allocated inline in a structure. They can be built-in (standard primitive types), user-defined (defined in source code) or enumerations (sets of enumerated values that are represented by labels but stored as a numeric type).
2. Reference Types: Store a reference to the value‘s memory address and are allocated on the heap. Reference types can be any of the pointer types, interface types or self-describing types (arrays and class types such as user-defined classes, boxed value types and delegates).

Although operations on variables of a value type do not affect any other variable, operations on variables of a reference type can affect the same object referred to by another variable. When references are made within the scope of an assembly, two types with the same name but in different assemblies are defined as two distinct types, whereas when using namespaces, the run time recognizes the full name of each type (such as System.Object, System.String, etc.). The rich set of types in CTS has well-designed semantics such that they can be widely used as a base type in Common Language Runtime (CLR) - based languages. This is why all .NET developers must have a thorough understanding of CTS.

What is an assembly? Explain its different types.

Microsoft .Net Assembly

Microsoft **.Net Assembly** is a logical unit of code, that contains code which the Common Language Runtime (CLR) executes. It is the smallest unit of deployment of a .net application and it can be a **.dll** or an **exe** . Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality. It include both executable application files that you can run directly from Windows without the need for any other programs (.exe files), and libraries (.dll files) for use by other applications.

Assemblies are the building blocks of .NET Framework applications. During the compile time Metadata is created with Microsoft Intermediate Language (MSIL) and stored in a file called Assembly Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. Assembly Manifest contains information about itself. This information is called Assembly Manifest, it contains information about the members, types, references and all the other data that the runtime needs for execution.

Every Assembly you create contains one or more program files and a Manifest. There are two types program files : Process Assemblies (EXE) and Library Assemblies (DLL). Each Assembly can have only one entry point (that is, DllMain, WinMain, or Main).

We can create two types of Assembly:

1. Private Assembly

2. Shared Assembly

A private Assembly is used only by a single application, and usually it is stored in that application's install directory. A shared Assembly is one that can be referenced by more than one application. If multiple applications need to access an Assembly, we should add the Assembly to the Global Assembly Cache (GAC). There is also a third and least known type of an assembly: Satellite Assembly . A Satellite Assembly contains only static objects like images and other non-executable files required by the application.

Difference between NameSpace and Assembly

A namespace is a logical grouping of types. An assembly can contain types in multiple namespaces and a single namespace can be spread across assemblies. More about... NameSpace and Assembly

How to add and remove Assembly from GAC ?

Each computer that .Net Framework is installed has a Global Assembly Cache, which is located in the Assembly folder in the Windows directory, normally C:\WinNT\Assembly. The Global Assembly Cache tool(Gacutil.exe) allows you to view and manipulate the contents of the global assembly cache. More about.. Add, Remove Assembly

What is .Net Assembly Qualified Name ?

An Assembly qualified name is the internal name of the assembly, combined with the Assembly Version, Culture, and public Key: these combination make it unique.

How to Assembly versioning ?

The AssemblyVersion attribute assigns the version number of the assembly, and this is embedded in the manifest. Version information for an assembly consists of the following four values : a major and minor version number, and two further optional build and revision numbers.

Write a program in C# which demonstrates Command line argument

Command Line Outputs

```
using System;
public class hello2{
public static void Main(string [] args)
{
Console.Write("Hello, IICC!");
Console.Write("You entered following {0} command line arguments :",
args.length);
For(int i=0;i<args.length;i++) { Console.WriteLine("{0},args[i]);
}
}}
```

output

```
D:\> hello2 arg1 arg2 arg3
```

Write General Structure of a C# program
A skeleton of a C# program

```
using System;
namespace MyNamespace1
{
Class MyClass1{}
struct MyStruct
{}
Interface IMyInterface{}
Delegate int MyDelegate();
Enum MyEnum {}
namespace MyNamesoace2{ }
class MyClass2 {
        public static void Main(string [] args)
        {
        ……
        }
}
}
```

General Structure of a C# program

- Documentation section
- Using directive section optional
- Interface section
- Classes section
- Main method section essential

Explain the C# main method and the compilation of C# program.

- Must contain Main method in which control starts and ends
- Main method is a static method
- Resides inside a class
- public :- access modifier
    - It tells the compiler that the Main method is accessible by anyone.
- static :-
    - Declares that the Main method is a global and can be called without an creating instance of a class.
- void :-
    - states that the Main method does not return any value.
- Three ways to declare Main
- i) can be void

```
static void Main()
{
.......
}
```

- Three ways to declare Main
- ii) it can return int

```
static int Main()
{
.......
return 0;
}
```

- iii) it can also take arguments

```
static void Main(string[] args)
{
.......
}
```

    - OR

```
static int Main(string[] args)
{
.......
return 0;
}
```

What are different ASP.Net files ?

Web site applications can contain different file types. By default, some are supported and managed by ASP.NET, and others are supported and managed by the IIS server. Optionally, you can specify that all types should be handled by ASP.NET.

Most of the ASP.NET file types can be automatically generated using the Add New Item menu item in Visual Studio.

File types are associated with applications by using mappings. For example, if you double-click a .txt file in Windows Explorer, typically Notepad opens, because in Windows, .txt file types are associated by default to Notepad.exe. In Web applications, file types are mapped to application extensions in IIS.

File Types Managed by ASP.NET

File types that are managed by ASP.NET are mapped to
the Aspnet_isapi.dll in IIS.

File type	Location	Description
.asax	Application root.	Typically a Global.asax file that represents the application class and contains optional methods (event handlers) that run at various points in the application life cycle.
.ascx	Application root or a subdirectory.	A Web user control file that defines a custom functionality that you can add to any ASP.NET Web Forms page.
.ashx	Application root or a subdirectory.	A handler file that is invoked in response to a Web request in order to generate dynamic content.
.asmx	Application root or a subdirectory.	An XML Web services file that contains classes and methods that can be invoked by other Web applications.
.aspx	Application root or a subdirectory.	An ASP.NET Web Forms page that can contain Web controls and presentation and business logic.
.axd	Application root.	A handler file that is used to manage Web site

		administration requests, such as Trace.axd.
.browser	App_Browsers subdirectory.	A browser definition file that identifies the features of an individual browser.
.cd	Application root or a subdirectory.	A class diagram file.
.compile	Bin subdirectory.	A precompiled stub files that point to an assembly that represents a compiled Web site file. When you precompile a Web site project, executable file types (.aspx, ascx, .master, and theme files) are compiled and put in the Bin subdirectory.
.config	Application root or a subdirectory.	A configuration file contains XML elements that represent settings for ASP.NET features.
.cs, .vb	App_Code subdirectory, or in the case of a code-behind file for an ASP.NET page, in the same directory as the Web page.	Source code files (.cs or .vb files) that define code that can be shared between pages, such as code for custom classes, business logic, HTTP modules, and HTTP handlers.
.csproj, .vbproj	Visual Studio project directory.	A project file for a Visual Studio Web-application project.
.disco, .vsdisco	App_WebReferences subdirectory.	An XML Web services discovery file that is used to help locate Web services.
.dsdgm, .dsprototype	Application root or a subdirectory.	A distributed service diagram (DSD) file that can be added to any Visual Studio solution that provides or consumes Web services to reverse-engineer an architectural view of the Web service interactions.
.dll	Bin subdirectory.	A compiled class library file (assembly). In a Web site project, instead of placing compiled assemblies in the Bin subdirectory, you can put source code for classes in the App_Code subdirectory.
.licx, .webinfo	Application root or a subdirectory.	A license file. Licensing allows control authors to help protect intellectual property by

		checking that a user is authorized to use the control.
.master	Application root or subdirectory.	A master page that defines the layout for other Web pages in the application.
.mdb, .ldb	App_Data subdirectory.	An Access database file.
.mdf	App_Data subdirectory.	A SQL Server Express database file.
.msgx, .svc	Application root or a subdirectory.	An Indigo Messaging Framework (MFx) service file.
.resources, .resx	App_GlobalResources or App_LocalResources subdirectory.	A resource file that contains resource strings that refer to images, localizable text, or other data.
.sdm, .sdmDocument	Application root or a subdirectory.	A system definition model (SDM) file.
.sitemap	Application root.	A sitemap file that defines the logical structure of the Web application. ASP.NET includes a default sitemap provider that uses sitemap files to display a navigational control in a Web page.
.skin	App_Themes subdirectory.	A skin file that contains property settings to apply to Web controls for consistent formatting.
.sln	Visual Studio project directory.	A solution file for a Visual Studio project..
.soap	Application root or a subdirectory.	A SOAP extension file.