Laboratory -1: Learn Your Tool

Ref: Simone Chiaretta, Front-end Development with ASP.NET Core, Angular, and Bootstrap, John Wiley & Sons, Inc

Objectives:

O[1]. To learn the execution procedure of the console and web applications.
O[2]. To learn the execution process of a program using .Net framework.

In this course, students are going to learn two different frameworks namely ConsoleApp (.Net Core/.Net Framework) and Web Application(ASP.NET Core/.Net Framework). In ConsoleApp students will practice exercises on C# programming language, and in Web Application, they will practice different interoperabilities between the web server and database server, connectivity of different web features namely form, API, and services, and authorization activities. Thus, the focus of today's laboratory is to learn your tool(Visual Studio 2019) properly to execute programs.

Task1: How to run a program as ConsoleAPP?

a) Create a simple program(SampleProgram.cs) using Console .Net Core.
b) Execute the program using Visual Stdio 2019 Editor.

The most important tool that comes with .NET Core is the dotnet host, which is used to launch .NET Core console applications, including the development tools, via the new .NET command-line interface (CLI). This CLI centralizes all the interactions with the framework and acts as the base layer that all other IDEs, like Visual Studio, use to build applications.
In order to try it out, just open the command prompt, create a new folder, move into this folder, and type dotnet new console. This command creates the skeleton of a new .NET Core console application, made of a Program.cs code file and the .csproj project definition file, named as the folder in which the command was launched. The new command can be executed using other arguments to specify the type of project to build: console (the one we used before), web, mvc, webapi, classlib, xunit (for unit testing), and etc. This is also the structure of all commands of the .NET CLI: dotnet followed by the command name, followed by its arguments.
Now that all the pieces are ready, the application can be executed by simply typing the command dotnet run. This first builds the application and then invokes it via the dotnet application host. In fact, this could be done manually as well, first by explicitly using the build command and then by launching the result of the build (which is a DLL with the same name of the folder where the application has been created) using the application host: dotnet bin\Debug\netcoreapp2.0\ consoleapplication.dll (consoleapplication is the name of the folder). In addition to building and running apps, the dotnet command can also deploy them and create packages for sharing libraries.

SampleProgram.cs

```
using System;
namespace ConsoleApplication
```

```
{
        public class Program
        {
                public static void Main(string[] args)
                {
                        Console.WriteLine("Hello World!");
                }
         }
}
```

c) Execute the program using MS Command Prompt.
   - Atfirst, go to the specific directory, where your project is located, then run the following command:
     dotnet new console --force --output ConsoleAppNetCore
     dotnet run --project ConsoleAppNetCore
     //ConsoleAppNetCore is the project name.

Web Forms provided the abstractions to deal with HTTP and web server objects and introduced the concept of server-side events to hide the stateless nature of the web, using the ViewState. The result was a very successful, feature-rich web framework with a very approachable programming model.

Task2: How to run a program with Web Form as WebApplication in .Net Framework?

a) Create a simple web form using WebApplication .Net Framework.
b) Execute the program with web form in Visual Stdio 2019 Editor.

Microsoft started to struggle to maintain frequent small components assembled and updated because of the huge monolithic framework of ASP.NET. The problem was not only a matter of release cycles. The development style also was changing. Hiding and abstracting away the complexities of HTTP and HTML markup helped a lot of WinForm developers to become web developers, but after more than five years of experience, developers wanted more control, especially over the markup rendered on pages.
In order to solve these two problems, in 2008 the ASP.NET team developed the ASP.NET MVC framework, based on the Model-View-Controller design pattern, which was also used by many of the popular frameworks at the time. This pattern allowed a cleaner and better separation of business and presentation logic, and, by removing the server-side UI components, it gave complete control of the HTML markup to developers. Furthermore, instead of being included inside the .NET framework, it was released out of band, making faster and more frequent releases possible. Although the ASP.NET MVC framework solved most of the problems of Web Forms, it still depended on IIS and the web abstracting library System.Web. This means that it was still not possible to have a web framework that was totally independent from the larger .NET framework.

Fast-forward a few years, single page applications (SPAs) was arrived. Basically, instead of interconnected, server-generated, data-driven pages, applications were becoming mostly static pages where data was displayed interacting with the server via Ajax calls to web services or Web APIs. Also, many services started releasing APIs for mobile apps or third-party apps to interact

with their data Another web framework was released to adapt better to these new scenarios: ASP.NET Web API.
The ASP.NET team also took this opportunity to build an even more modular component model, finally ditching System.Web and creating a web framework that could live its own life independently from the rest of ASP.NET and the larger .NET framework. A big role was also played by the introduction of NuGet, Microsoft's package distribution system, making it possible to deliver all these components to developers in a managed and sustainable way. One additional advantage of the break-up from System.Web was the capability to not depend on IIS anymore and to run inside custom hosts and possibly other web servers.

Task3: How to run a program in .Net Framework?

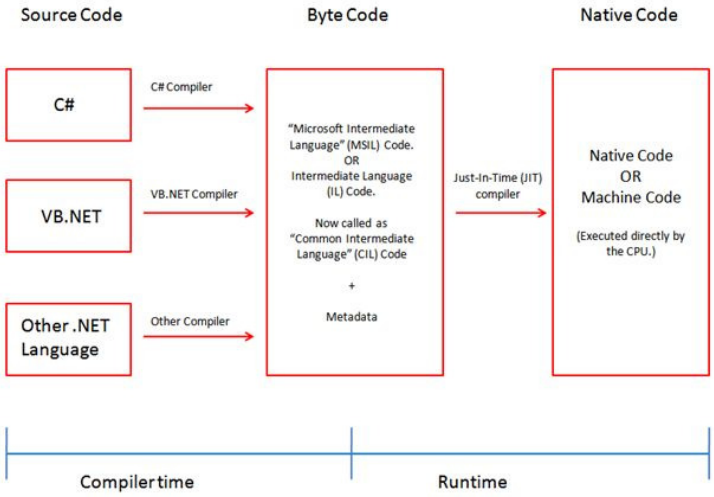


Fig 1: Compile and Runtime Environment of .Net Framework

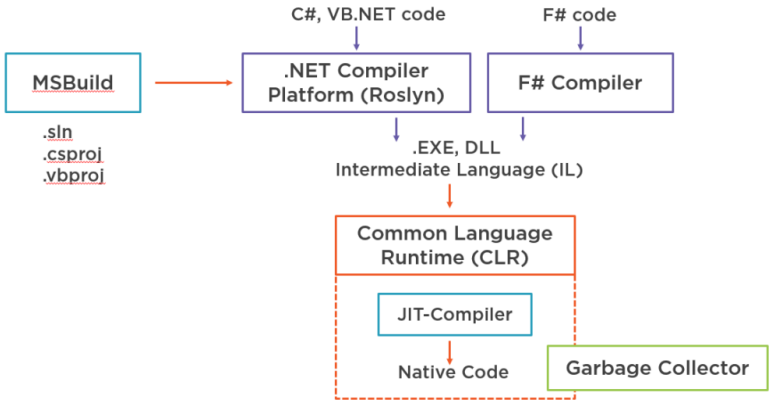


Fig 2: Detailed Runtime Environment of .Net Framework

.NET SDK is a set of libraries and tools for developing and running .NET applications. SDK download includes the following components:

- .NET CLI: Command-line tools that you can use for local development and continuous integration scripts.
- dotnet driver: A CLI command that runs framework-dependent apps.

- .NET runtime: Provides a type system, assembly loading, a garbage collector, native interop, and other basic services.
- Runtime libraries. Provides primitive data types and fundamental utilities.

Task4: Cross-Language Interoperability.Net Framework?

The .NET Framework is language-independent. We can develop many languages targeting the .NET Framework, such as C#, C++, F#, Visual Basic and Windows PowerShell. In simple words, a function, class or anything written in one language can be easily used in another language. (For example a function, class or anything written in C# can be easily used in VB.NET).

In our example we will use two(2) languages namely VB .Net and C#.

a) Create a simple WebApplication in .Net Framework using C#. Build and trace the path of the DLL in bin/debug folder.

```
namespace CrossPlatformApp
{
   public class Class1
   {
      public int Sum(int n1, int n2) {
         return n1 + n2;
      }

      public int Sum2(int n1, int n2) {
         return n1 + n2;
      }

      public int Sum3(int n1, int n2) {
         return n1 + n2;
      }
   }
}
```

b) Create a simple WebApplication in .Net Framework using VB.
Add refernce of the C# project then write the following codes

```
Imports System
Imports CrossPlatformApp.Class1

Module Program
        Dim obj As New CrossPlatformApp.Class1

         Sub Main(args As String())
                Console.WriteLine(obj.Sum(2, 3))
         End Sub
End Module
```

c) Execute the VB program with C# modules.

Laboratory -2: Basic of C# Programming

Objectives:

O[1]. To debug a C# program and trace the errors in a program.
O[2]. To learn class library(.dll) creation, reference as assembly, and import as a namespace.
O[3]. To learn how to use the console to take input and display output.
O[4]. To implement an array, and learn how to pass array in methods.

In this laboratory works, students are going to learn program writing in C# console-based platform, to understand the debugging facilities of Visual Studio 2019, to trace the errors in a program, and solve them. Students will learn how to create class libraries, increase reusability by using the .dll in another project.

Task1: Understand and run the following program(Program1) as ConsoleAPP(ASP.Net Core)

Program 1:

```
namespace Lab2Ex1
{
    class Program
    {
       static double Add(double a, double b)
        {
            return a * b; // deliberate bug!
        }
        static void Main(string[] args)
        {
            double a = 4.5; // or use var
            double b = 2.5;
            double answer = Add(a, b);
           // Person p = new Person();
            Console.WriteLine($"{a} + {b} = {answer}");
            Console.ReadLine(); // wait for user to press ENTER
        }    }
}
```

a) Debug the program using Toggle Breakpoint, Step Into and other debug options.
b) Execute the program using Visual Studio 2019 Editor.

The $ special character identifies a string literal as an **interpolated string**. An interpolated string is a string literal that might contain interpolation expressions. When an interpolated string is resolved to a result string, items with interpolation expressions are replaced by the string representations of the expression results.

```
string name = "Mark";
var date = DateTime.Now;

// Composite formatting:
Console.WriteLine("Hello, {0}! Today is {1}, it's {2:HH:mm} now.", name, date.DayOfWeek, date);
```

```
// String interpolation:
Console.WriteLine($"Hello, {name}! Today is {date.DayOfWeek}, it's {date:HH:mm} now.");

// Both calls produce the same output that is similar to:
// Hello, Mark! Today is Wednesday, it's 19:40 now.
```

An assembly is a file that is automatically generated by the compiler upon a successful compilation of every .NET application. It can be either a Dynamic Link Library or an executable file. An Assembly contains Intermediate Language (IL) code, which is similar to Java byte code. In the .NET language, it consists of metadata. Metadata enumerates the features of every "type" inside the assembly or the binary. In addition to metadata, assemblies also have a special file called Manifest. It contains information about the current version of the assembly and other related information.

You can view the IL code generated by a . NET-aware compiler with the help of a utility called ILDASM.exe, which comes with the .NET Framework. ILDASM stands for Intermediate Language Disassembler. It may be located under the BIN directory of the .NET SDK installation folder. Use ILASM.EXE command to redirect to EXE or DLL file.

Fig1: Output of IlDASM.EXE Tool

Task2: Class library(.dll) creation, reference as assembly, and import as namespace

a) Create two namespaces in two different projects and import them into the third project and use the imported namespaces methods.

b) Apply your knowledge into program1 and use Add methods from separate .dll. Explain the code reusability.

Task3: Use the console to take input to a C# program and display contents to the console.

Problem2:

Read a three digit integer input from the console, and print the reverse of that number in the console.

- You can call a reverse() method from Main() and pass the integer values.
- Reverse() method will return the reverse value to Main().

- Main() will catch and print the value.

i=Convert.ToInt32(Console.ReadLine());
or
int res;
string myStr = "200";
res = int.Parse(myStr);

Convert.ToInt32() takes an object as its argument. Convert.ToInt32() also does not throw ArgumentNullException when its argument is null the way Int32.Parse() does. That also means that Convert.ToInt32() is probably a wee bit slower than Int32.Parse(). Convert.ToInt32() calls int.Parse() internally. Except for one thing Convert.ToInt32() returns 0 when argument is null, Otherwise both work the same way.

Task4: Understand the following program(Program3) and run it as ConsoleAPP(ASP.Core)

Problem3:
Take 10 inputs from console and print their average in console.

- You should use separate function to take input and find average.
- Main() will call those functions and print the average.

type [ ] array_name= new type[size];
int [ ] sample = new int [10];
int [ ] sample; sample=new int [10];

In C++ arrays are not valid as return type. However, in C# arrays are implemented as objects, a method can also return as an array.

Considering the following code, an array is a reference type and so for this function:

public static void FirstDouble(int[] array)
the variable array is a reference because int[] is a reference type. So array is a *reference* that is *passed by value*.

Thus, modifications made to the array inside the function are applied to the int[] object to which the array refers. And so those modifications are visible to all references that refer to that same object. And that includes the reference that the caller holds.

Now, if we look at the implementation of this function:

```
public static void FirstDouble(int[] array)
{
   //double each elements value
   for (int i = 0; i < array.Length; i++)
      array[i] *= 2;
```

```
    //create new object and assign its reference to array
    array = new int[] { 11, 12, 13 };
}
```

If the function had been declared like this:

public static void FirstDouble(ref int[] array)

then the reference array would have been passed by reference and the caller would see the newly created object { 11, 12, 13 } when the function returned

Task4: Understand the following program(Program4) and run it as ConsoleAPP(ASP.Core)

Problem3:
Write a program so that if the user gives input ‘d’ or ‘D’; ‘k’ or ‘K’ and ‘s’ or ‘S’ the program will print Dhaka, Khulna, and Sylhet respectively.

- You need to use a switch statement.
- Dhaka, Khulna, and Sylhet will be stored in a string array.

Assignment -3: Properties and Indexer

Objectives:

O[1]. To learn properties and apply them in real-world problem-solving.
O[2]. To learn Indexers and apply them in real-world problem-solving.

In this works, students are going to learn program writing in the C# console-based platform with the use of properties, and indexers.

Task1: Implement a class named Account that contains the following members:

- A private int data field named Id for the account (default 0).
- A private double data field named Balance for the account (default 0.0).
- A private double data field named minBalance for the account (default 1000.0).
- A no-arg constructor that creates a default account with Id=0 and Balance=0.0
- An argument constructor that creates an account with the specified id, initial balance
- A method named setBalance() that sets a given amount to an instance variable Balance if the given amount is more than minBalance.
- A method named getBalance() that returns the value of the current balance.

a) Create a **class library (.dll)** of the class and execute the dll from another project in Visual Studio 2019 console Editor.
b) Check that the private field members are not visible from your driver project.

Property in C# is a member of a class that provides a flexible mechanism for classes to expose private fields. Internally, C# properties are special methods called accessors. The set accessor automatically receives a parameter called value and assigns it to property. Get accessor automatically returns the property value. The value keyword represents the value of a property. Property is accessed as a public field. We can also use a property instead of a method that has one parameter and a return value because we can define custom logic in the get and set accessors.

- A property with get and set accessor is a Read/Write property(Id, Balance).
- A property with get accessor is a Read-only property (minBalance).
- A property with set accessor is a Write only property.

C# also supports static properties, which belong to the class rather than to the objects of the class. All the rules applicable to a static member apply to static properties also. Remember that set/get accessor of static property can access only other static members of the class. Also , static properties are invoking by using the class name.

Task2: Change the problem in Task1 so that we can add a property for balance private variable and call set and get accessors automatically.

a) Create a class library (.dll) of the class and execute the dll from another project in Visual Studio 2019 console Editor.
b) Check that the private field members are not visible from your driver project and you do not need to share to the driver project.
c) Change the property to be static. Thus Balance also needs to be static. Explain Why?
d) Create a Read Only property for minBalance and try to change the value of minBalance property. You will get Compile error. Explain- Why?
e) Show auto-implemented properties(get; set;). Auto-implemented properties in C# make code more readable and clean if there is no additional calculation needed. Compiler creates a private anonymous field that can only be accessed through the get and set accessors.

This feature in C# allows you to index as class or struct as you would do it with an array. When we define an indexer for a class, we force it to behave like a virtual array. The array access operator, or [], can be used to access instances of a class that implements the indexer. The user can get or set the indexed value without pointing to an instance or a type member. The indexers are very similar to properties, but the main difference is that accessors to the indexers will take parameters, while properties cannot.

Task3: Implement a class named Account that contains the following members:

- A private double data field array named Balance. Which will hold the account balance of customers against their customer id.
- An argument constructor that allocates the size of the Balance and the size will be given during the Account class instantiation.
- An indexer with set and get accessors will be added in the Account class.
    - set accessor will check the valid id (0 to size) and set the value to the Balance array corresponding to customer id location, if and only if the id is valid.
    - get accessor will return the Balance array value of a customer only if the customer id is valid, otherwise, return the -2 value.

a) Create a class library (.dll) of the class and execute the dll from another project in Visual Studio 2019 console Editor.
b) Test the driver class with the creation of an Account object
c) Now use the object reference as a virtual array with [ ]operator to create 5 balance values and store them using indexer.
d) Trace the result of how the indexer is used to store and retrieve balance values.
e) Show overloaded indexer to use an index as string

```
String[] array= new string[] {“zero”,“one”, “two”, “three”, “four”, “five”}
public int this[string name]
 {
      get
      {
        for(int i=0, i<6; i++){
              if(array[i].ToLower() == name.ToLower())
                return Balance[i];
```

```
            }

            return 0;
        }
    }
```

f) Show multi-dimensional indexers.

C# includes specialized classes that store series of values or objects are called collections. There are two types of collections available in C#: non-generic collections and generic collections. The System.Collections namespace contains the non-generic collection types and System.Collections.Generic namespace includes generic collection types. In most cases, it is recommended to use generic collections because they perform faster than non-generic collections and also minimize exceptions by giving compile-time errors.

Generic List<T> contains elements of the specified type. It grows automatically as you add elements to it. The List<T> is a collection of strongly typed objects that can be accessed by index and having methods for sorting, searching, and modifying the list.

Task4: Create a student list using List collection and access the values using Indexers.

a) Implement a student class and create property for Id, Name, and Address.
b) Implement a department class and create a student list using List collection
c) Add few student objects into the List
d) Create an indexer into the department class and use the instance of the class to behave a virtual array.
e) Indexer get will return the student name whose id is given as index.
f) Indexer set will change the student name whose id is given as index.
g) Show the working strategies of the indexer.

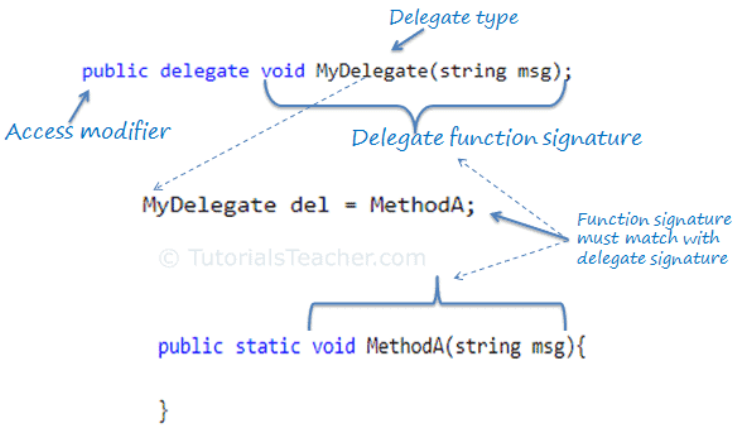
Assignment -4: Delegate, Anonymous Method, and Lambda expression

Objectives:

O[1]. To learn the delegates and apply them in real-world problem-solving.
O[2]. To learn the difference between tight coupling coding and loose coupling coding and apply them in programming.
O[3]. To apply delegate to invoke anonymous methods.
O[4]. To understand the anonymous method and lambda expression.

In this works, students are going to understand tight coupling and loose coupling coding concepts and apply them in the Asp.Net framework web form application. In addition, they will learn programming with method invocation by method name and by a delegate, anonymous method and its implementation with delegates and lambda expression will also be discussed.

A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance. [Microsoft]



https://www.tutorialsteacher.com/csharp/csharp-delegates

Figure 1: Association of delegate instance with any method with a compatible signature and return type.

There are three(3) steps involved while working with delegates: Declare a delegate, Set a target method, and Invoke a delegate.

Tight coupling is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its class.

Loose coupling is achieved using a design that promotes single-responsibility and separation of concerns. A loosely coupled class can be consumed and tested independently of other (concrete) classes.

Task1: Implement a class named Employee that contains the following members:

- A public auto-implemented property named name. Which will return string-type data.
- A public auto-implemented property named Id. Which will return int type data.
- A public auto-implemented property named salary. Which will return double type data.
- A public auto-implemented property named experience. Which will return int type data.
- A static method named IsPromotable(), which will take input a list of the employee as parameters and no return.
- This method will check the experience of all employees and print the name of the employees whose experience is greater than 4 years.

Implement a driver class with the following members:

- Create a instance of a List<T> and name it as employee list.
- Initialize the list with the following values:

```
employeeList.Add(new Employee() { name = "Merinda", Id = 1, salary = 200000.00, experience = 5 });
employeeList.Add(new Employee() { name = "Belal", Id = 2, salary = 255000.00, experience = 4 });
employeeList.Add(new Employee() { name = "Roy", Id = 3, salary = 280000.00, experience = 6 });
employeeList.Add(new Employee() { name = "Poly", Id = 4, salary = 160000.00, experience = 3 });
```

- Invoke the IsPromotable() method of the Employee class with employeeList parameter.

a) The above code is a tightly coupled code, we need to make it loosely coupled by moving the IsPromotable() method implementation in the driver class. This will help the programmer to work in framework programming and increase independency and reusability.

b) Now use delegate to invoke IsPromotable() method and create the following method into the Employee class to print the promotable employee name.

```
public static void Promotable(List<Employee> empList, IsPromotable_D promotable) {
        foreach (Employee e in empList) {
           if (promotable(e))
              Console.WriteLine(e.name + "is promotable");
        }
}
```

Task2: How to run a program with Web Form as WebApplication in .Net Framework?

a) Create a simple web form using WebApplication .Net Framework.

b) Execute the program with web form in Visual Stdio 2019 Editor.

Microsoft started to struggle to maintain frequent small components assembled and updated because of the huge monolithic framework of ASP.NET. The problem was not only a matter of release cycles. The development style also was changing. Hiding and abstracting away the complexities of HTTP and HTML markup helped a lot of WinForm developers to become web

developers, but after more than five years of experience, developers wanted more control, especially over the markup rendered on pages.
In order to solve these two problems, in 2008 the ASP.NET team developed the ASP.NET MVC framework, based on the Model-View-Controller design pattern, which was also used by many of the popular frameworks at the time. This pattern allowed a cleaner and better separation of business and presentation logic, and, by removing the server-side UI components, it gave complete control of the HTML markup to developers. Furthermore, instead of being included inside the .NET framework, it was released out of band, making faster and more frequent releases possible. Although the ASP.NET MVC framework solved most of the problems of Web Forms, it still depended on IIS and the web abstracting library System.Web. This means that it was still not possible to have a web framework that was totally independent from the larger .NET framework.

Fast-forward a few years, single page applications (SPAs) was arrived. Basically, instead of interconnected, server-generated, data-driven pages, applications were becoming mostly static pages where data was displayed interacting with the server via Ajax calls to web services or Web APIs. Also, many services started releasing APIs for mobile apps or third-party apps to interact with their data Another web framework was released to adapt better to these new scenarios: ASP.NET Web API.
The ASP.NET team also took this opportunity to build an even more modular component model, finally ditching System.Web and creating a web framework that could live its own life independently from the rest of ASP.NET and the larger .NET framework. A big role was also played by the introduction of NuGet, Microsoft's package distribution system, making it possible to deliver all these components to developers in a managed and sustainable way. One additional advantage of the break-up from System.Web was the capability to not depend on IIS anymore and to run inside custom hosts and possibly other web servers.

Task3: Take three inputs from Number1, Number2, and Option using web form of ASP.NET framework and invoke the method Add or Sub method according to the option value with Number1 and Number2 as parameters.

Implement a class named MathClass that contains the following members:

- a public static method named Add() that takes two integer arguments x and y and returns the integer value of x+y.
- a public static method named Sub() that takes two integer arguments x and y and returns the integer value of x-y.

Implement a driver class(UI) that contains the following members:

- Read a string of numbers and convert them into an integer. Assign the integer value to the integer private Number1 field.
- Read a string of numbers and convert them into an integer. Assign the integer value to the integer private Number2 field.
- Read a string of numbers and convert them into an integer. Assign the integer value to the integer private Option field.
- Now check the value of Option in the main method and call the corresponding method.

Look carefully, the classes are tightly coupled with each other. UI class needs to change for any additional methods in MathClass.

a) Make the classes loosely coupled. Shift the option checking method and place it in the MathClass. Call the method from the UI class.
b) Use delegate and return delegates to the UI class.
c) Convert the Add and Sub methods as anonymous by the delegate and check the results.
d) Convert the anonymous methods as lambda expressions and check the results.

MCSE 541: Web Computing and Mining

.Net Introduction

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

@Text Book: Chapter 6

About Me

Experience	Time	Institute	
Bachelor (BSc)	1998-2001	American International University-Bangladesh	
Masters (MSc)	2003-2005	Asian Institute of Technology Thailand	
Doctorate (PhD)	2006-2009	Tokyo Institute of Technology Japan	
Post Doctoral Researcher	2009-2011	JSPS-NII Japan	
Assistant Prof.	2005-2014	American International University-Bangladesh	
Contact Asst. Prof.	2013-2014	Thompson Rivers University (TRU), Kamloops, BC, CANADA	
Associate Prof.	2014-2019	East West University	
Professor	2019-	IUBAT	

People and Places

Prof. Dr. Shamim Akhter,

Professor, CSE, Stamford University, Bangladesh

https://www.linkedin.com/in/prof-dr-md-shamim-akhter-a3a96666/

profshamimakhter@gmail.com

Please Call me : Prof. Shamim!, Professor, or Sir

Course Link:

https://classroom.google.com/c/NDk2NzE3Mzg5MTY4?cjc=jrdkuao

Class Schedule –Summer 2022

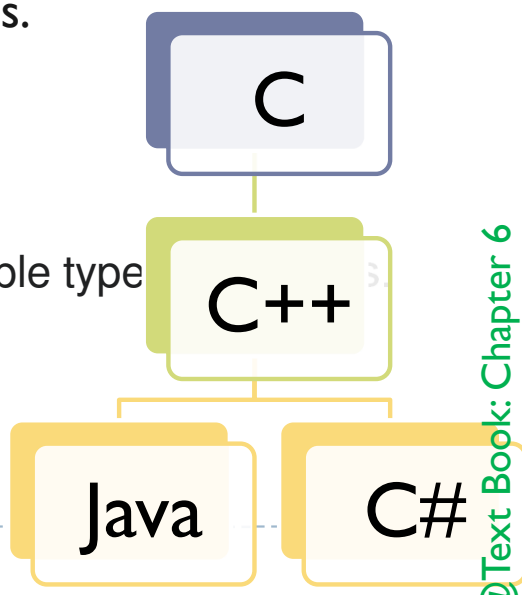
Day	9:00-10:00	10:00-11:00	11:00-12:30	12:30-14:00	14:00-15:30	15:30-17:00	17:00-18:30
Saturday	Web Data Mining		COA CSE233 Room: 503 Batch: 75A	Counsiling Hour	Research		
Sunday		Class Meterials					
Monday							
Tuesday							
Wednesday		Counsiling Hour				Class Meterials	COA CSE233 Room: 503 Batch: 75A
Thursday							

.Net and C#

- .NET is a free, open-source development platform
  - for building many kinds of apps-Web apps, web APIs etc.
  - supports integrated development environments (IDEs), and other tools and supports three programming languages C#, F#, Visual Basic.

- Java has portability but
  - lacks cross-language interoperability
    - requires for large and distributed software system.
    - Assemblies take the form of executable (*.exe*) or dynamic link library (*.dll*) files, and are the building blocks of .NET applications. They provide the common language runtime with the information it needs to be aware of type implementations.

  - Not Fully Integrates with windows platforms
    - Universal Windows Platform (UWP) API @ Windows 10
      - allows developers to create apps that will potentially run on multiple type
      - Development support by C#, JavaScript but not Core Java
    - Windows Forms written C#
      - Write rich client applications for Labtop, Desktop or Tablet PCs



@Text Book: Chapter 6

ClassLibrary
Framework

Program.cs*

CrossLanguage | CrossLanguage.Class1 | Sum(int n1, int n2)

```
1   using System;
2
3   [assembly:CLSCompliant(true)]
4
5   namespace CrossLanguage
6   {
        0 references
7       public class Class1
8       {
            0 references
9           public int Sum(int n1, int n2)
10          {
11              return n1 + n2;
12          }
            0 references
13          public int Sum2(int n1, int n2)
14          {
15              return n1 + n2;
16          }
            0 references
17          public int Sum3(int n1, int n2)
18          {
19              return n1 + n2;
20          }
21      }
22
23  }
```

Testing Cross-language interoperability
.NET Framework and Common Language Runtime(CLR)
Mix Language Environment-C#, VB

Microsoft Intermediate Language (MSIL)
Portable Assembly language

ConsoleApp

Program.vb*

ConsoleApp1 | Program | Main

```
1   Imports CrossLanguage.Class1
2
    0 references
3   Module Program
4       Dim obj As New CrossLanguage.Class1
5
        0 references
6       Sub Main(args As String())
7           obj.e
8           Cons
9       End Sub
10  End Module
11
```

Equals
GetHashCode
GetType
Sum
Sum2
Sum3
ToString

Function Object.Equals(obj As Object) As Boolean
Determines whether the specified object is equal to the current object.

Solution Explorer

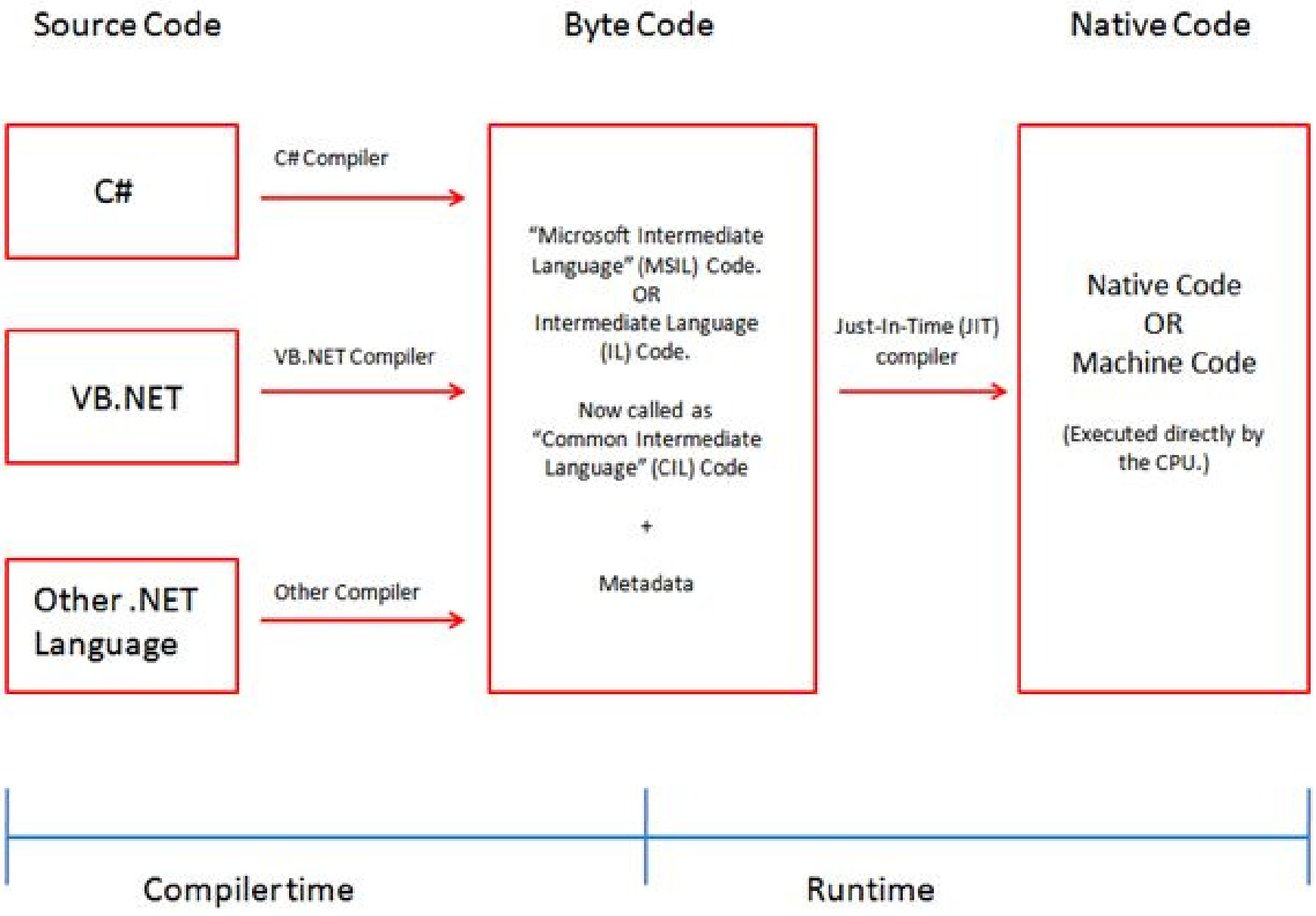
Search Solution Explorer (Ctrl+;)

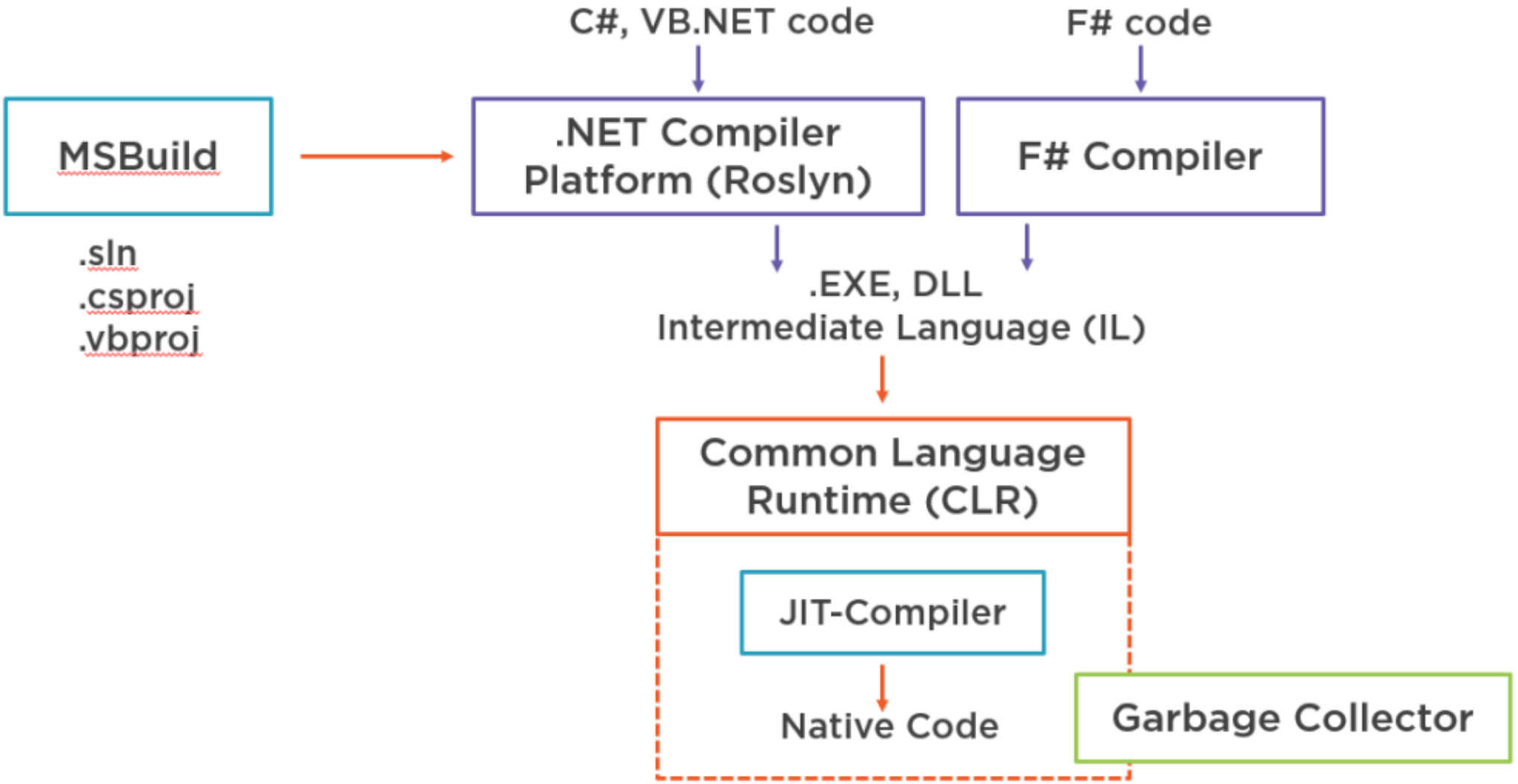
Solution 'ConsoleApp1' (1 of 1 project)
ConsoleApp1
Dependencies
Assemblies
CrossLanguage
Frameworks
Microsoft.NETCore.App
Program.vb

Add Reference
and browse the dll

@Text Book: Chapter 6

.NET Code Execution Process



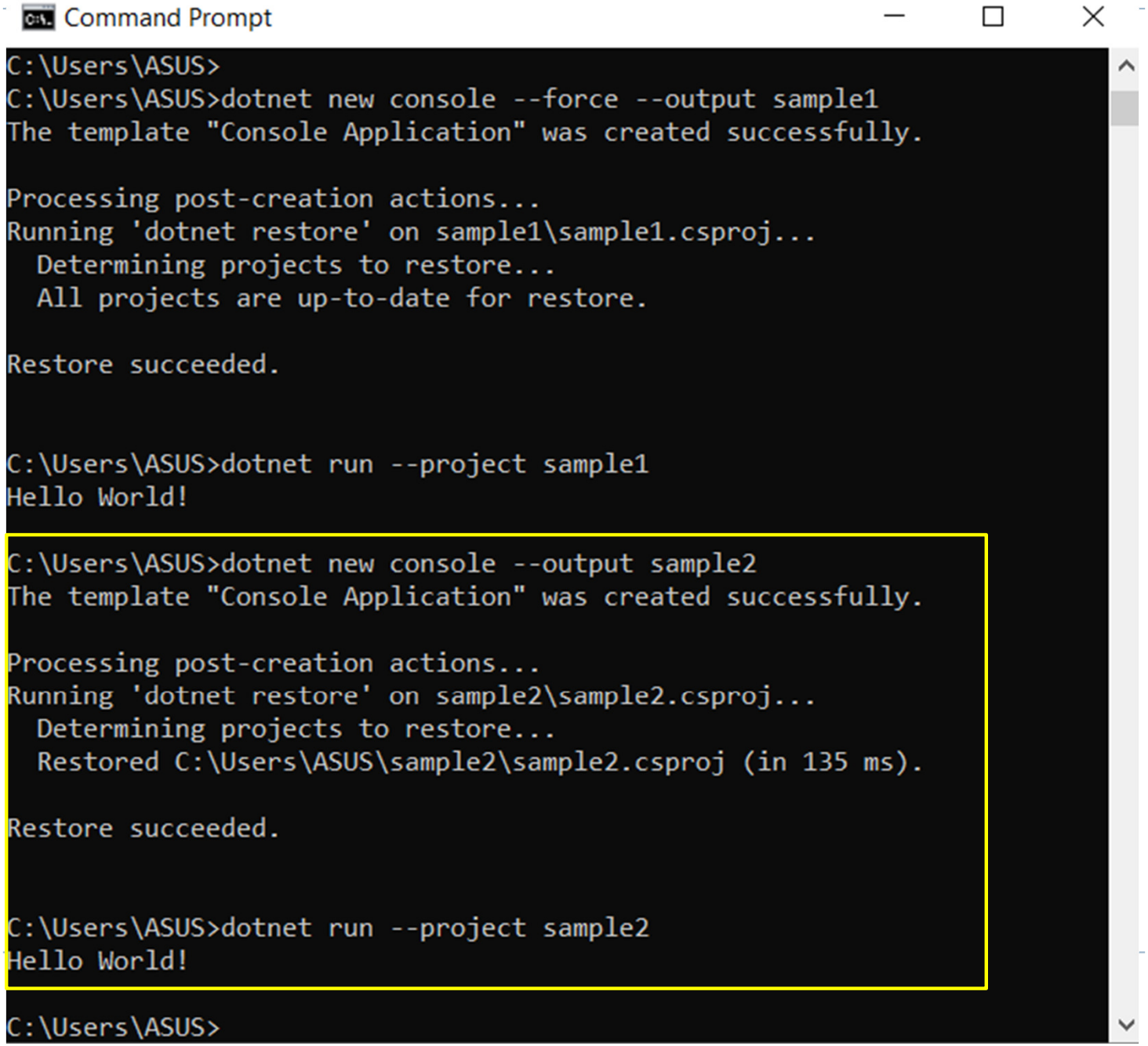

C#, VB.NET code
F# code
MSBuild
.NET Compiler Platform (Roslyn)
F# Compiler
.sln
.csproj
.vbproj
.EXE, DLL
Intermediate Language (IL)
Common Language Runtime (CLR)
JIT-Compiler
Native Code
Garbage Collector

SDK and Runtimes

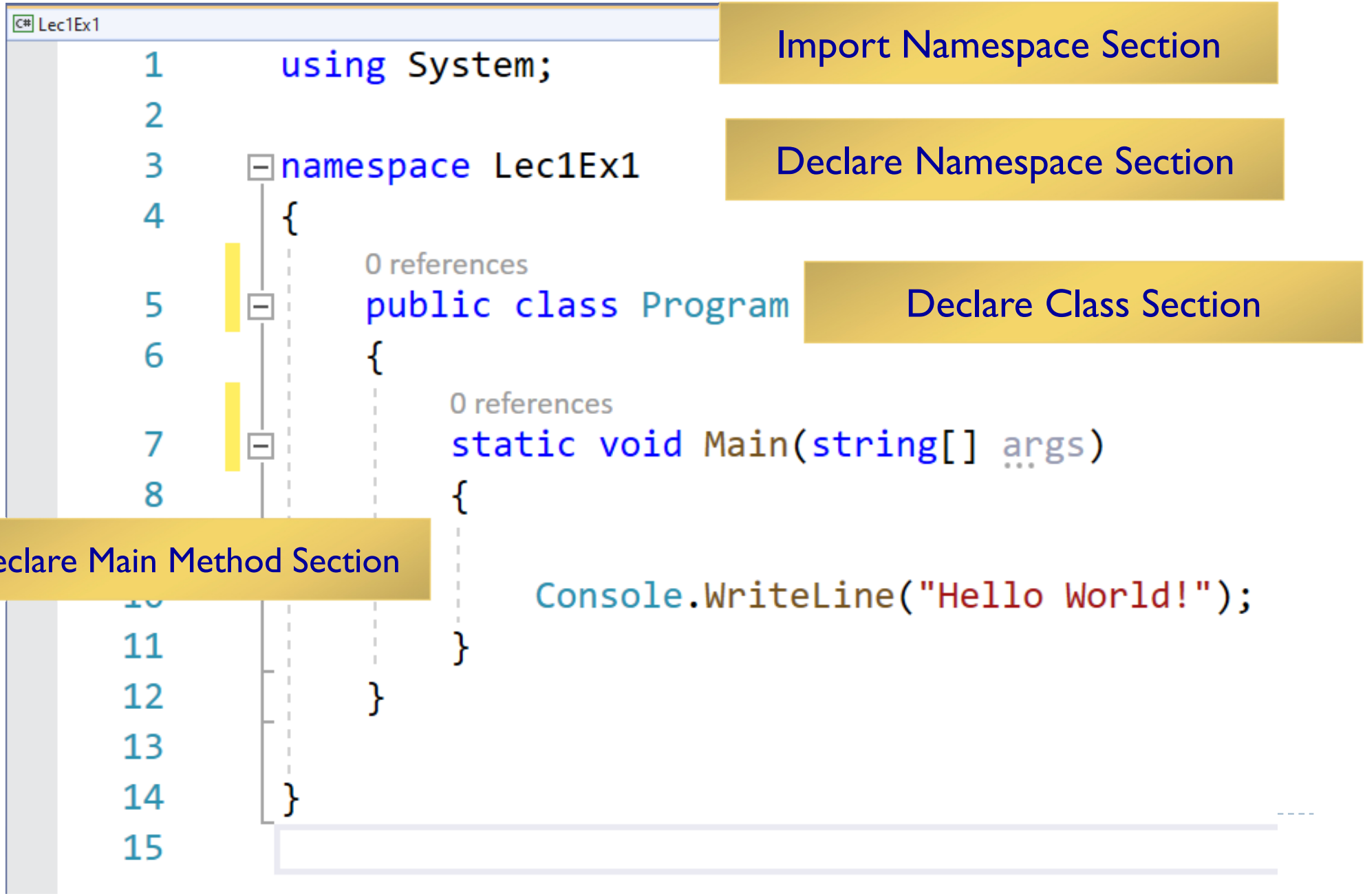
- .NET SDK is a set of libraries and tools for developing and running .NET applications.

- SDK download includes the following components:
  - .NET CLI: Command-line tools that you can use for local development and continuous integration scripts.
  - dotnet driver: A CLI command that runs framework-dependent apps.
  - .NET runtime: Provides a type system, assembly loading, a garbage collector, native interop, and other basic services.
  - Runtime libraries. Provides primitive data types and fundamental utilities.

Command Line Tool



Basic Structure of A C# Program



Namespace

- In C# namespace defines a declarative region
  - Remove the same name conflicts between user define class and .Net Library classes.
  - Within a namespace, we can declare another namespace, a class, an interface, a structure, an enumeration or a delegate.

Import Section:

Import statements are use to import the Base Class Libraries. This is similar to C #include statement.

Syntax: **using namespace;**

Example: **using System;**

```
System.Console.WriteLine("Hello World!");
```

If the required namespace is a member of another namespace we have to specify the parent and child namespaces separated by a dot.

- **using System.Data;**
- **using System.IO;**

a user-defined namespace is to be declared. In .NET applications, all classes related to the project should be declared inside one namespace.

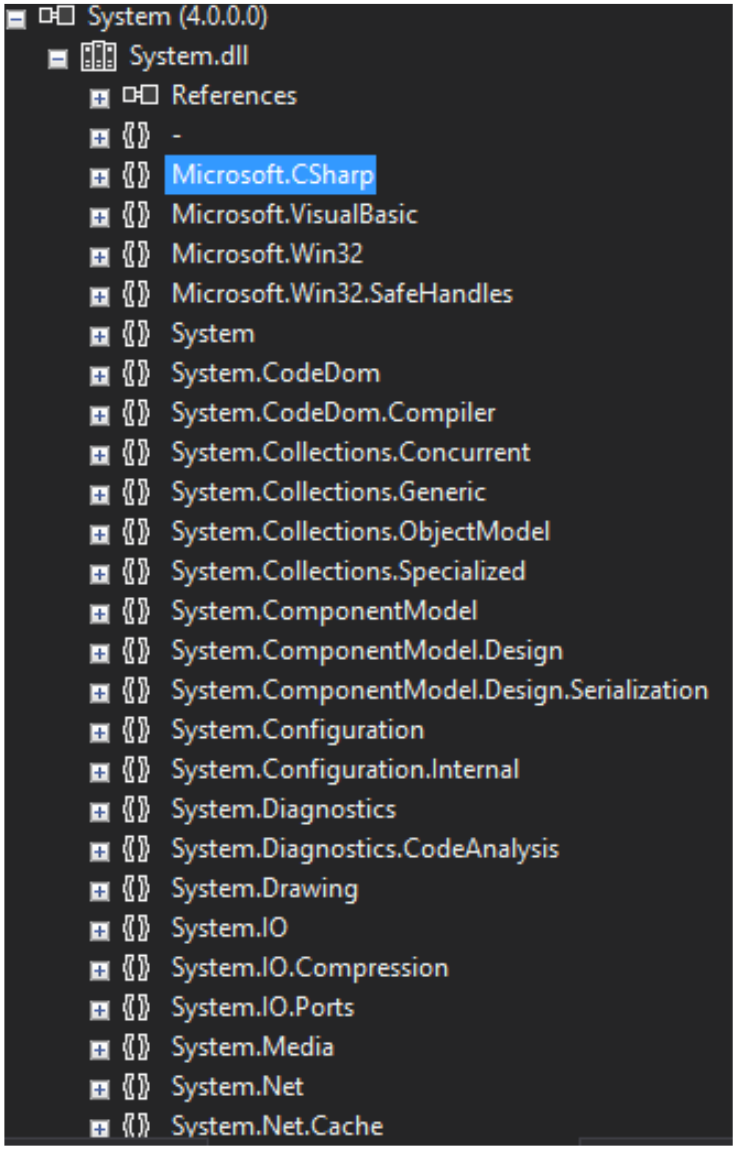
Syntax:

```
namespace NamespaceName
{
}
```

Generally, the namespace name will be the same as the project name.

Namespace and Assembly

- Namespaces organize objects in an assembly.
- An assembly is a reusable, versionable and self-describing building block of a CLR application.
- Assemblies can contain multiple namespaces.
- Namespaces can contain other namespaces.
- An assembly provides a fundamental unit of physical code grouping.
- A namespace provides a fundamental unit of logical code grouping.



Using Namespace

```
1   using System;
2   //using Lec1Ex2;
3
4   namespace Lec1Ex0
5   {
        0 references
6       public class Program
7       {
            0 references
8           static void Main() {
9               Console.WriteLine("@Main Lec1Ex0");
10              Program1.Main2();
11          }
12      }
13
14  }
15  namespace Lec1Ex2
16  {
        0 references
17      public class Program1
18      {
            0 references
19          public static void Main2()
20          {
21              Console.WriteLine("@Main Lec1Ex2");
22
23          }
24      }
25
26  }
27
```

- Course Objectives and Outcomes.
  - Outline Pdf file.

Classes and Objects

- A class is a template that define the forms of an object

Blueprint/Logical State of a Bike

Construction

Physical State of Bikes

Field Members

- Instance Variable
- Static/Class Variable

Function Members

- Constructor, Destructor
- Indexers, Events, Operators and Properties

A Simple Class

- Class is created by use of keyword class.

```
modifier class Classname {

   modifier data-type field1;
   modifier data-type field2;
   ...
   modifier data-type fieldN;

   modifier Return-Type methodName1(parameters) {
      //statements
   }

   ...

   modifier Return-Type methodName2(parameters) {
      //statements
   }
}
```

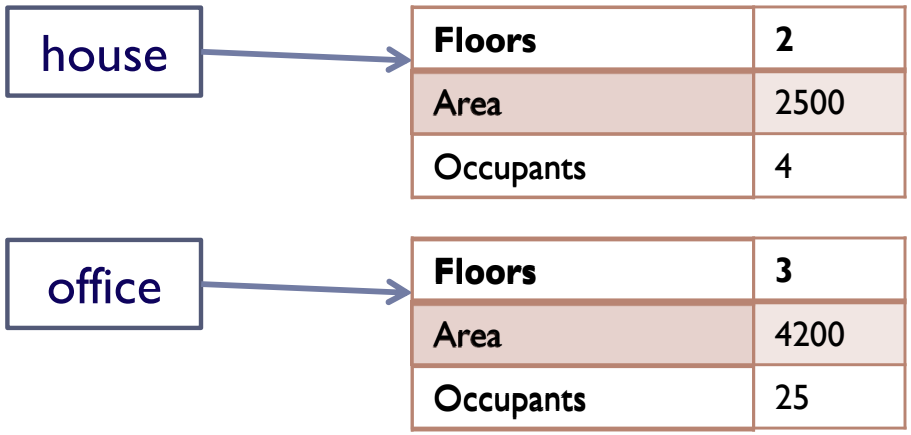
Instance Variable

Non-static variable cannot be referenced without creating class instance

```
public class VariableExample
{
    int myVariable;
    static int data = 30;
    public static void main(String args[])
    {
            VariableExample obj = new VariableExample();
            System.out.println("Value of instance variable: "+obj.myVariable);
            System.out.println("Value of static variable: "+VariableExample.data);
    }
}
```

JAVA Program
Same works for C# also

Reference Variable and Assignment




```
using System;

class Building{
        public int Floors;
        public int Area;
        public int Occupants;

        public void areaPerPerson(){
                Console.WriteLine(" "+ Area/Occupants + " area per person");
        }
}
public class Demo{
    static void Main(){
        Building house  = new Building();
        Building office = new Building();
        house.Occupants=4;   house.Area=2500;  house.Floors=2;
        office.Occupants=25;  office.Area=4200;   office.Floors=3;
        house.areaPerPerson();
        office.areaPerPerson();
    }
}
```

Building house1= new Building();
Building house2= house1

Constructor

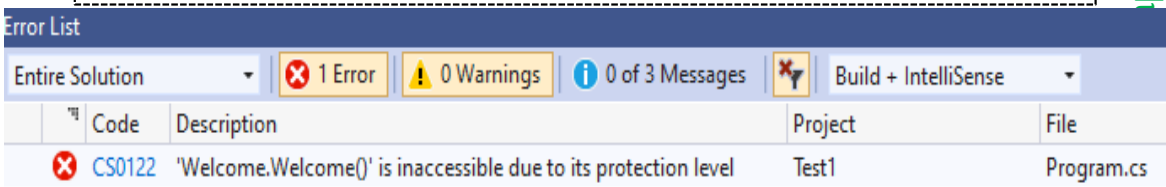
- Constructors
  - are special methods called when a class is instantiated.
  - will not return anything.
  - name is same as class name.
  - By default C# will create default constructor internally.
  - with no arguments and no body is called default constructor.
  - with arguments is called parameterized constructor.
  - by default public.
  - We can create private constructors.
  - Constructor allocates memory for all instance variables of its class.

Private Constructor

```
using System;
namespace ConstructorSample
{
    public class Welcome
    {
        private Welcome()   // // Default private constructor
        {
          Console.WriteLine("Default Private Constructor...");
        }

        static void Main(string[] args)
        {
            Welcome obj = new Welcome();
            Console.Read();
        }
    }
}
```

```
public class demo {
        static void Main(string[] args)
        {
            Welcome obj = new Welcome();
            Console.Read();
        }
    }
```



Parameterized Constructor

```
using System;
 public class MyClass
 {
     public int x;

     public MyClass (int i)
     {
         x=i;
     }
 }
  public class Demo{
     static void Main( )
     {
        MyClass t1=new MyClass (10);
        MyClass t2=new MyClass (88);
        Console.WriteLine(t1.x + " " + t2.x);
     }
 }
```

Garbage Collection and Destructors

- Recovery of free memory from unused object
  - C++ delete operator is used to free allocated memory
  - C# and Java: Garbage Collection- automatically
    - Can't know or make assumptions about the timing of garbage collection
    - Non-deterministic

- Destructor
  - Another method to clean object allocated memory.
  - It ensures that a system resource owned by an objet is released.
  - Is declared like constructor except proceed with ~(tilde) -tilda
  - No return type and takes no arguments.

@Text Book: Chapter 6

Destructor Example

```
using System;

  public class Destructor
  {
     public int x;

     public Destructor (int i)
     {
        x=i;
     }
     ~Destructor(){
        Console.WriteLine("Destructing " + x);
     }

     public void Generator(int i)
     {
        Destructor o = new Destructor(i);
     }
  }
```

```
public class Demo{
        static void Main( )
        {
           int count;
           Destructor ob = new Destructor(0);
           for(count=1; count<1000; count++)
                ob.Generator(count);

            Console.WriteLine("Done");
        }
    }
```

```
Destructing 755
Destructing 940
Destructing 14
Destructing 199
Destructing 384
Destructing 569
Destructing 754
Destructing 939
Destructing 13
Destructing 198
Destructing 383
Destructing 568
Destructing 753
Destructing 938
Destructing 12
Destructing 197
Destructing 382
Destructing 567
Destructing 752
Destructing 937
Destructing 11
Destructing 196
Destructing 381
Destructing 566
Destructing 751
Destructing 936
Destructing 10
Destructing 195
Destructing 380
Destructing 565
Destructing 750
Destructing 935
```

- No serialization
- Non deterministic

this Keyword

```
using System;
class Rect{
    public int Width;
    public int Height;
   //public Rect(){ this(3, 2); }
   public Rect(int Width, int Height){
            this.Width=Width;
            this.Height=Height;
    }
    public int Area(){
            return this.Width * this.Height;
    }
}
            class UseRect{
                static void Main(){
                    Rect r1= new Rect(4,5);
                    Rect r2= new Rect(7, 9);
                    Console.WriteLine("Area of r1:" +r1.Area());
                    Console.WriteLine("Area of r1:" +r2.Area());
                }
            }
```

MCSE 541: Web Computing and Mining

C# Class, Object & Method
C# Operator, Indexer & Properties

Prof. Dr. Shamim Akhter

@Text Book: Chapter 9,10

Classes and Objects

- A class is a template that define the forms of an object

Blueprint/Logical State of a Bike

Construction

Physical State of Bikes

Field Members

- Instance Variable
- Static/Class Variable

Function Members

- Constructor, Destructor
- Indexers, Events, Operators and Properties

A Simple Class

- Class is created by use of keyword class.

```
modifier class Classname {

   modifier data-type field1;
   modifier data-type field2;
   ...
   modifier data-type fieldN;

   modifier Return-Type methodName1(parameters) {
      //statements
   }

   ...

   modifier Return-Type methodName2(parameters) {
      //statements
   }
}
```

Instance Variable

Handling Static and Non-static variable

1) Non-static variable cannot be referenced without creating class instance, WHY?
2) Static variable is referenced without class instance but error when referred with instance. WHY?

1)

Program.cs*

Winter_Prog1 | Winter_Prog1.Program | Main(string[] args)

```
1   using System;
2
3   namespace Winter_Prog1
4   {
        2 references
5       class Program
6       {
7           int myVariable;
8           static int data = 30;
9
            0 references
10          static void Main(string[] args)
11          {
12              Program p = new Program();
13
14              Console.WriteLine(myVariable);
15              Console.WriteLine(p.data
16          }
17      }
```

(field) int Program.myVariable

CS0120: An object reference is required for the non-static field, method, or property 'Program.myVariable'

2)

```
1   using System;
2
3   namespace Winter_Prog1
4   {
        2 references
5       class Program
6       {
7           int myVariable;
8           static int data = 30;
9
            0 references
10          static void Main(string[] args)
11          {
12              Program p = new Program();
13
14              Console.WriteLine(p.myVariable);
15              Console.WriteLine(p.data);
16          }
17      }
```

(field) static int Program.data

CS0176: Member 'Program.data' cannot be accessed with an instance reference; qualify it with a type name instead

Show potential fixes (Alt+Enter or Ctrl+.)

Output

Show output from: Package Manager

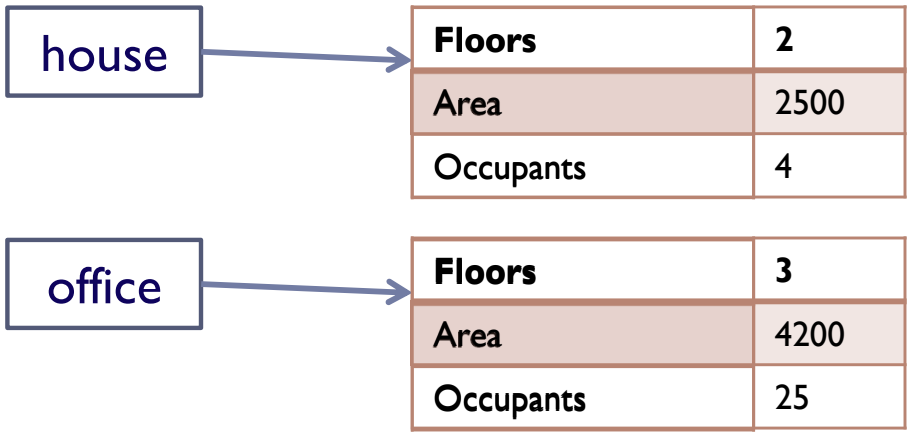
@Text Book: Chapter 6

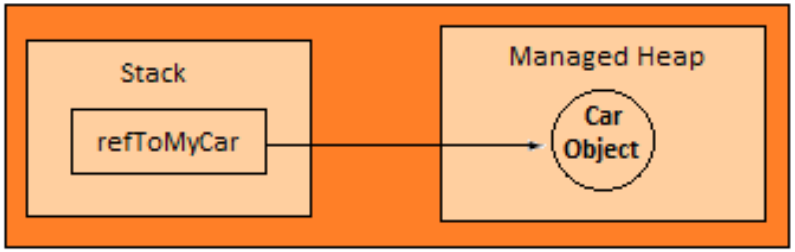
Reference Variable and Assignment

```
using System;

class Building{
        public int Floors;
        public int Area;
        public int Occupants;

        public void areaPerPerson(){
                Console.WriteLine(" "+ Area/Occupants + " area per person");
        }
}
public class Demo{
    static void Main(){
        Building house  = new Building();
        Building office = new Building();
        house.Occupants=4;   house.Area=2500;  house.Floors=2;
        office.Occupants=25;  office.Area=4200;   office.Floors=3;
        house.areaPerPerson();
        office.areaPerPerson();
    }
}
```





Building house1= new Building();
Building house2= house1

Constructor

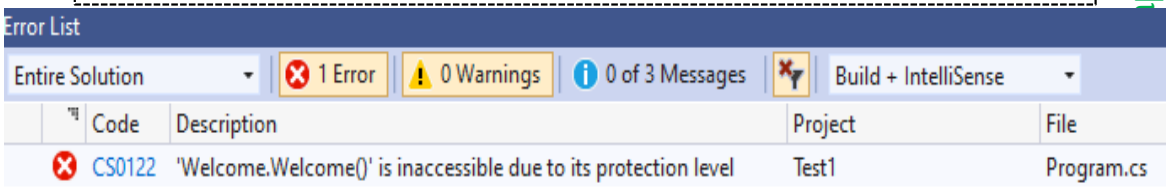
- Constructors
  - are special methods called when a class is instantiated.
  - will not return anything.
  - name is same as class name.
  - By default C# will create default constructor internally.
  - with no arguments and no body is called default constructor.
  - with arguments is called parameterized constructor.
  - by default public.
  - We can create private constructors.
  - Constructor allocates memory for all instance variables of its class.

Private Constructor

```
using System;
namespace ConstructorSample
{
    public class Welcome
    {
        private Welcome()   // // Default private constructor
        {
          Console.WriteLine("Default Private Constructor...");
        }

        static void Main(string[] args)
        {
            Welcome obj = new Welcome();
            Console.Read();
        }
    }
}
```

```
public class demo {
        static void Main(string[] args)
        {
            Welcome obj = new Welcome();
            Console.Read();
        }
    }
```



Parameterized Constructor

```
using System;
 public class MyClass
 {
     public int x;

     public MyClass (int i)
     {
         x=i;
     }
 }
  public class Demo{
     static void Main( )
     {
        MyClass t1=new MyClass (10);
        MyClass t2=new MyClass (88);
        Console.WriteLine(t1.x + " " + t2.x);
     }
 }
```

Garbage Collection and Destructors

Memory allocation/deallocation rule:

IF YOU ALLOCATE IT (use a new),
YOU MUST DEALLOCATE IT (delete it)!

- Recovery of free memory from unused object
  - C++ delete operator is used to free allocated memory
  - C# and Java: Garbage Collection- automatically
    - Can't know or make assumptions about the timing of garbage collection
    - Non-deterministic
- Garbage Collector cleans the memory by three ways,
  1. Destructor
  2. Dispose()
  3. Finalize

- Destructor
  - is called/invoked automatically by .NET Garbage Collector (GC). We can not manually invoke or control it.
  - It ensures that a system resource owned by an objet is released.
  - Is declared like constructor except proceed with ~(tilde) -tilda
  - No return type and takes no arguments.

Destructor Example

```
using System;

  public class Destructor
  {
     public int x;

     public Destructor (int i)
     {
        x=i;
     }
     ~Destructor(){
        Console.WriteLine("Destructing " + x);
     }

     public void Generator(int i)
     {
        Destructor o = new Destructor(i);
     }
  }
```

```
public class Demo{
        static void Main( )
        {
           int count;
           Destructor ob = new Destructor(0);
           for(count=1; count<1000; count++)
                ob.Generator(count);

            Console.WriteLine("Done");
        }
    }
```

```
Destructing 755
Destructing 940
Destructing 14
Destructing 199
Destructing 384
Destructing 569
Destructing 754
Destructing 939
Destructing 13
Destructing 198
Destructing 383
Destructing 568
Destructing 753
Destructing 938
Destructing 12
Destructing 197
Destructing 382
Destructing 567
Destructing 752
Destructing 937
Destructing 11
Destructing 196
Destructing 381
Destructing 566
Destructing 751
Destructing 936
Destructing 10
Destructing 195
Destructing 380
Destructing 565
Destructing 750
Destructing 935
```

- No serialization
- Non deterministic

this Keyword

```
using System;
class Rect{
    public int Width;
    public int Height;
   //public Rect(){ this(3, 2); }
   public Rect(int Width, int Height){
            this.Width=Width;
            this.Height=Height;
    }
    public int Area(){
            return this.Width * this.Height;
    }
}
            class UseRect{
                static void Main(){
                    Rect r1= new Rect(4,5);
                    Rect r2= new Rect(7, 9);
                    Console.WriteLine("Area of r1:" +r1.Area());
                    Console.WriteLine("Area of r1:" +r2.Area());
                }
            }
```

Methods-get( ) and set()

```
class Rectangle
  {
     private int height;
     private int width;

     public Rectangle( ) { this.height = 0; this.width = 0; }
     ~Rectangle( ) { }

     public void set(int h, int w) {
        this.height= h; this.width = w;
     }
```

Methods-get( ) and set()

```
public int getHeight( ) {
        return this.height;
    }
    public int getWidth() {
        return this.width;
    }

    public Rectangle get() {
        Rectangle rec = new Rectangle();
        rec.height = this.height;
        rec.width = this.width;
        return rec;
    }
}
```

Methods-get( ) and set()

```
class Demo {
        static void Main(string[] args)
        {
            Rectangle R1 = new Rectangle();
            R1.set(10, 20);
            Console.WriteLine("Area of R1="+ R1.getHeight()*R1.getWidth());

            Rectangle R2 = R1.get();
            Console.WriteLine("Area of R2=" + R2.getHeight() * R2.getWidth());
        }
    }
```

Properties

- Suppose, you want to accomplish limit the range of values that can be assigned to a field.
  - But the filed has private access. Now?
  - We need to use get() / set() method to access the field.
  - Lets make it a better organize and more user comfortable.

- Property, a class member
  - Combines a specific field with the method that access it.

```
element-type {
    get{     //get access code
    }
    set{    //set access code
    }
}
```

A property applies specific rule to a field's value.

```
class Program
 {
        private int prop;
        public Program() { prop = 0; }

        public int prop_Property {
            get { return prop; }
            set { if (value >= 0) prop = value; }
        }
 }
```

```
class Demo {
        static void Main(string[ ] args)
        {
            Program p = new Program( );
            Console.WriteLine(p.prop_Property);
            p.prop_Property = 100;
            Console.WriteLine(p.prop_Property);
            p.prop_Property = -10;
            Console.WriteLine(p.prop_Property);
        }
    }
```

Operator Method

- Defines the action of the operator relative to its class.
  - this process is called operator overloading and closely related to method overloading.
- The method should be a public and static method.
- The function is marked by keyword **operator** followed by the operator symbol which we are overloading.
- The return type of an operator function represents the result of an expression.

Two forms of operator methods

General form for overloading a **Unary operator**

```
public static ret-type operator op(param-type operand)
{
        //operations
}
```

General form for overloading a **Binary operator**

```
public static ret-type operator op(param-type1 operand1, param-type2 operand2, )
{
        //operations
}
```

```
public class Rectangle
 {
        private int length;
        private int breadth;
        public Rectangle ( ){length=0; breadth=0;}
        public Rectangle(int length, int breadth)
        {
            this.length = length;
            this.breadth = breadth;
        }
        public int Area()
        {
            return length * breadth;
        }

        public void DisplayArea()
        {
            Console.WriteLine(this.Area());
        }
 }
```

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        Rectangle rect3 = rect1 + rect2;
        rect3.DisplayArea();
    }
}
```

Binary

Solution:

Operator Overloading

Binary Operator Overloading

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        Rectangle rect3 = rect1 + rect2;
        rect3.DisplayArea();
    }
}
```

```
public static Rectangle operator +(Rectangle rect, Rectangle rect1)
{
        Rectangle result= new Rectangle( );
        result.length    = rect.length + rect1.length;
        result.breadth  = rect.breadth + rect1.breadth;

        return result;
}
```

- N.B: The above operator method will be put after the constructor of Rectangle Class.

Unary Operator Overloading

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect3 = -rect1;
        rect3.DisplayArea();

    }
}
```

```
int a = 10; int b = -a; // b = -10
```

If p is true, then !p will be false.
If p is false, then !p will be true.

```
public static Rectangle operator -(Rectangle rect)
{
        Rectangle result= new Rectangle( );
        result.length    = - rect.length ;
        result.breadth  = - rect.breadth ;

        return result;
}
```

- N.B: The above operator method will be put after the constructor of Rectangle Class

++ / -- Postfix and Prefix form

Basically, you've misunderstood how this line works:

```
Test obj2 = ++obj;
```

If you think of using your operator as a method, that's like saying:

```
obj = Test.operator++(obj);
obj2 = obj;
```

So yes, you end up with `obj` and `obj2` being the same reference. The result of `++obj` is the value of `obj` *after* applying the `++` operator, but that `++` operator affects the value of `obj` too.

If you use

```
Test obj2 = obj++;
```

then that's equivalent to:

```
Test tmp = obj;
obj = Test.operator++(obj);
obj2 = tmp;
```

At that point, the value of `obj2` will refer to the original object, and the vlaue of `obj` will refer to the newly-created object with a higher `x` value.

++ / -- Postfix and Prefix form

```
class Program
{
    public  int x;
     public Program() {
        this.x = 0;
     }

     public Program(int x){
        this.x = x;
     }
     public static Program operator ++(Program P) {
        Program result = new Program();
        result.x = P.x + 1;
        return result;
     }
}
```

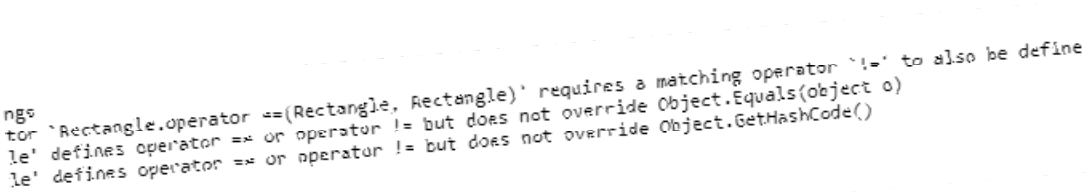
```
class Demo
{
    static void Main(string[] args)
    {
     Program P1 = new Program(2);
     Program P2 = new Program();
     Program P3 = new Program(2);
     P2 = P1++;
     Console.WriteLine("P2.x="+ P2.x+", P1.x=" + P1.x);
     P2 = ++P3;
     Console.WriteLine("P2.x=" + P2.x+", P3.x=" + P3.x);


     Console.WriteLine("Hello World!");
    }
}
```

Conditional Operator Overloading

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        if (rect1 == rect2)
            Console.WriteLine("Both Rectangle have same dimensions");
    }
}
```

Both(==) and (!=) needs to define together.
Only one will keep compile error.




```
public static bool operator ==(Rectangle rect1, Rectangle rect2)
{
    return (rect1.Area() == rect2.Area()) ? true : false;
}

public static bool operator !=(Rectangle rect1, Rectangle rect2)
{
    return (rect1.Area() != rect2.Area()) ? true : false;
}
```

N.B: The above operator method will be put after the constructor of Rectangle Class

Overloading the operator methods

```
public class Demo{
    public static void Main(){

        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        Rectangle rect3 = rect1 + rect2;
        rect3.DisplayArea();
        int area= rec3+ 2000;

    }
}
```

```
        public static Rectangle operator +(Rectangle rect, Rectangle rect1)
        {
            return new Rectangle(rect.length + rect1.length, rect.breadth + rect1.breadth);
        }

        /// The function add the area of two rectangles provided.

        public static int operator +(Rectangle rect, int area2)
        {
            return rect.Area() + area2;
        }
```

Indexer

- Allows an object to be indexed like an array.
- [ ] operator may define to classes without operator overloading.

General form of a Indexer with get and set accessor

```
element-type this[int index] {
    get{
        //return the value specified by index
    }
    set{
        // set the value specified by index
    }
}
```

An indexer allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([ ]).

A simple Indexer

```
using System;
namespace IndexerApplication {
   class IndexedNames {
      private string[ ] namelist = new string[size];
      static public int size = 10;
      public IndexedNames()
      { for (int i = 0; i < size; i++) namelist[i] = "N.A."; }
       public string this[int index] {
            get {
                  string tmp;
                  if( index >= 0 && index <= size-1 )
                  { tmp = namelist[index]; }
                  else { tmp = ""; }
                  return ( tmp );
            }
            set {
                  if( index >= 0 && index <= size-1 )
                        { namelist[index] = value; }
            }
      }
```

```
    static void Main(string[] args)
    {
            IndexedNames names = new IndexedNames();
            names[0] = "Zara"; names[1] = "Riz";
            names[2] = "Nuha"; names[3] = "Asif";
            names[4] = "Davinder"; names[5] = "Sunil";
            names[6] = "Rubic";

            for ( int i = 0; i < IndexedNames.size; i++ ) {
                    Console.WriteLine(names[i]);
            }
            Console.ReadKey();
    }
  }
}
```

```
Zara
Riz
Nuha
Asif
Davinder
Sunil
Rubic
N. A.
N. A.
N. A.
```

Indexers May Not Require an Underline Array

```
class Program
{
      public int this[int index] {
          get {
              if (index >= 0 && index < 16) return pwr(index);
              else return -1;
          }
          set { //no need set just read array//}
      } // index
     public int pwr(int N) {
          int result = 1;
          for (int i = 0; i < N; i++)
              result *= 2;
          return result;
      }
}
```

```
public class demo {
      static void Main(string[] args)
      {
          Program p = new Program();
          for(int i=0; i<18; i++)
              Console.WriteLine(p[i] + " ");
      }
  }
```

```
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
-1
-1
```

- Run the following link programs:
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/with-visual-studio?pivots=dotnet-6-0
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/debugging-with-visual-studio?pivots=dotnet-6-0
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/publishing-with-visual-studio?pivots=dotnet-6-0
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/library-with-visual-studio?pivots=dotnet-6-0

MCSE 541: Web Computing and Mining

C# Delegates, Anonymous Function & Lambda Expressions

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

@Text Book: Chapter 9,10

Function Pointer

```
#include<stdio.h>

void fun(int a){
        printf("Value of a is%d\n",a);
}

int main(){
        void (*fun_ptr)(int);
       fun_ptr=&fun;

      (*fun_ptr) (10);
       return;
}
```

- Output: Value of a is 10

- Void is considered a data type (for organizational purposes), but it is basically a keyword to use as a placeholder where you would put a data type, to represent "no data".
- void is an incomplete type that cannot be completed, This means you cannot apply the sizeof operator to void, but you can have a pointer to an incomplete type.
- Hence, you can declare a routine which does not return a value as: **void MyRoutine();**
- But, you cannot declare a variable like this: **void bad_variable;**
- However, when used as a pointer, then it has a different meaning: void* vague_pointer;
- This declares a pointer, but without specifying which data type it is pointing to.

Steps:

1. We define a function pointer(fp).
2. Assign the value to fp.
3. Call function pointer(fp).

Delegates

- Delegates is an object that refer to a method.
- Same delegates can refer to call different methods.
  - Which method a delegate will refer decided in runtime

General form for delegate

delegate *ret-type* name(*parameter-list*);

- Delegate ret-type is the reference method ret-type
- Parameter-list depends on the method argument list
- Delegate can not be static but delegate ref can be static

- Why delegates:
  - Delegates support events.
  - Delegates give your program a way to execute methods at runtime without having to know precisely what those methods are at compile time.

@Text Book: Chapter 15

Delegates Example

```
using System;
public delegate int Mydelegate(int x,int y);

class sample
{
    public static int rectangle(int a, int b)
    {
            return a * b;
    }
}
```

There are three(3) steps involved while working with delegates:
- Declare a delegate,
- Set a target method, and
- Invoke a delegate.

```
class Program
{
    static void Main()
    {
       Console.WriteLine("My simple Delegate Program");
       Mydelegate mdl = new Mydelegate(sample.rectangle);
       Console.WriteLine("The Area of rectangle is {0}", mdl(4, 5));
       Console.ReadKey();
    }
}
```

Example

```
using System;
public delegate string StrMod(string str);
class DelegateTest{
        public static string ReplaceSpaces(string s){
                Console.WriteLine("Replacing spaces with hyphens.");
                return s.Replace(' ', '-');
        }
        public static string RemoveSpaces(string s){
                string temp=" ";    int i;
                Console.WriteLine("Removing spaces.");
                for(i=0; i<s.Length; i++)
                        if(s[i]!=' ') temp+=s[i];
                return temp;
        }
        public static string Reverse(string s){
                string temp="";        int j;
                Console.WriteLine("Reversing string.");
                for(j=s.Length-1; j>=0; j--)
                        temp+=s[j];
                return temp;
        }
}
```

```
class Test{
   public static void Main(){
            string str1, str2, str3;
            StrMod strOp = new StrMod(DelegateTest.ReplaceSpaces);
            str1=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str1);

            strOp = new StrMod(DelegateTest.RemoveSpaces);
            str2=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str2);


            strOp = new StrMod(DelegateTest.Reverse);
            str3=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str3);


    }
}
```

Method Group Conversation 2.0 C#

```
class Test{
    public static void Main(){
            string str1, str2, str3;
            StrMod strOp = DelegateTest.ReplaceSpaces;
            str1=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str1);

            strOp = DelegateTest.RemoveSpaces;
            str2=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str2);

            strOp = DelegateTest.Reverse;
            str3=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str3);
    }
}
```

Multicasting-invocation list/chain

```
class Test{
    public static void Main(){
            string str, str1;
            str1="This is the Test.“;
            StrMod strOp;
            StrMod strOp1 = DelegateTest.ReplaceSpaces;
            StrMod strOp2 = DelegateTest.RemoveSpaces;
            StrMod strOp3 = DelegateTest.Reverse;
            strOp = strOp1;
            strOp += strOp2;
            strOp +=strOp3;
            str=strop(str1);
            Console.WriteLine("Resulting string: "+str);
    }
}
```

Multicasting-invocation list/chain With ref variable

```
delegate void StrMod(ref string s);

class sample {
        public static void ReplaceSpaces(ref string s){
            string temp;
            Console.WriteLine("Replacing spaces with hyphens.");
            temp= s.Replace(' ', '-');
            s = temp;
        }
        public static void RemoveSpaces(ref string s){
            string temp = " "; int i;
            Console.WriteLine("Removing spaces.");
            for (i = 0; i < s.Length; i++)
                if (s[i] != ' ') temp += s[i];
            s=temp;
        }
    }
```

```
public static void Reverse(ref string s)
{
        string temp = ""; int j;
        Console.WriteLine("Reversing string.");
        for (j = s.Length - 1; j >= 0; j--)
            temp += s[j];
        s= temp;
}
```

```
class Program
    {
        static void Main(string[] args)
        {
            String str = "This is a pot.";
            StrMod sd1, sd2, sd3;

            sd1= sample.ReplaceSpaces;
            sd2 = sample.RemoveSpaces;
            sd3 = sample.Reverse;

            sd1 += sd2;
            sd1 += sd3;

            sd1(ref str);
            Console.WriteLine(str);
        }
    }
```

Anonymous Function

Nameless methods.

They prevent creation of separate methods, especially when the functionality can be done without a new method creation.

Anonymous methods provide a cleaner and convenient approach while coding.

- A method can be used only by its delegate
  - Invoke via a delegate and never call on its own
- That time anonymous function is useful
  - Unnamed block of code that is passed to a delegate constructor
- Two types (C# 3.0) of Anonymous Function:
  - anonymous method and lambda expression

Delegates and Anonymous Method

```
class Program
{
    delegate void Countit();
     static void Hello()
     {   Console.WriteLine("Hello World!"); }

     static void Main(string[] args)
     {
         //Countit count = new Countit(Program.Hello);
         //count();
         Countit count = delegate { Console.WriteLine("Hello World!");
                                    }; //Anonymous Method
         count();
     }
}
```

Anonymous method with argument and return values

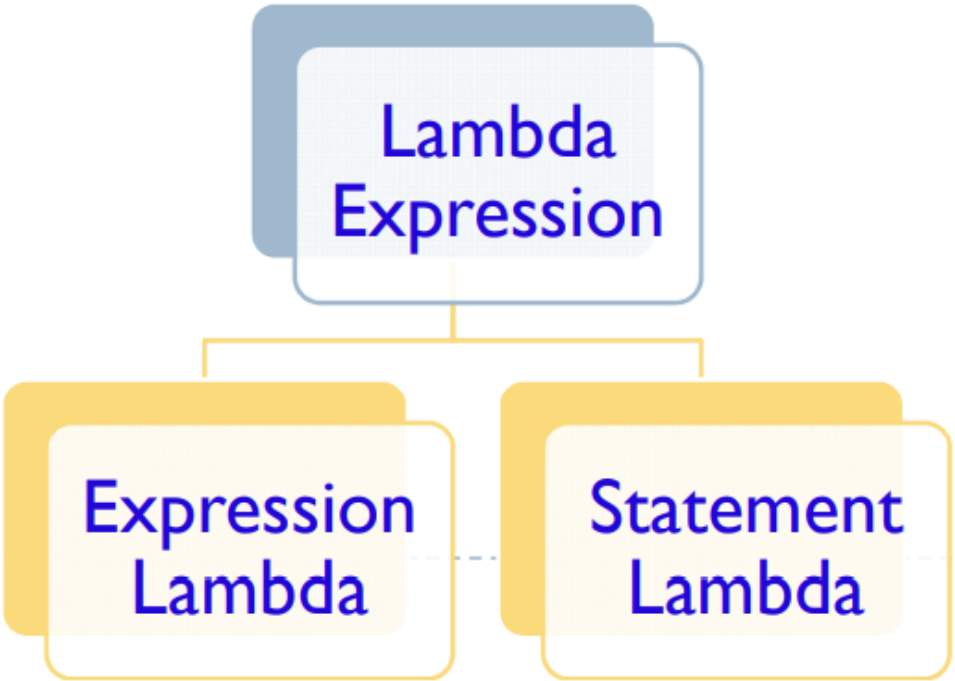
```
class Program
    {
        delegate int Countit(int end);
        static void Main(string[] args)
        {
            Countit count = delegate(int end) {
                                int i, sum = 0;
                                for (i = 1; i <= end; i++)
                                    sum += i;
                                return sum;
                            }; //Anonymous Method

            Console.WriteLine(count(5));
        }

    }
```

Lambda Expression

- An alternative to anonymous method.
- Mostly uses in LINQ (Language Integrated Query)
  - applicable to delegates and events
- Lambda Expressions use lambda operator => (goes, becomes)
  - it divides the expression into two parts
  - left is specified with input parameter, right is the lambda body



Expression Lambda

Parameter(s)	=>	Expression

- Expression on the right side of Lambda, acts on parameter(s) on the left side of Lambda.

- count=> count+2; return the value of count increased by two
- n => n%2 ==0; return true if n is even false if n is odd

3. Declare a delegate and show how a delegate replaces the following method.

```
public static double FunctionName (int b){
        return (1.0* b+2);
}
```

```
public delegate double MyD (int b)
MyD D = new MyD (FunctionName);
D(100);
```

4. Convert the following Anonymous function represented with Lambda expression to Anonymous function representation with delegate representation.

```
Delegate-D D = (int n) => {
                int c=500;
                Console.WriteLine(c);
                return (n+c);
              };
```

```
Delegate-D D = delegate (int n)
               { int c=500;
                 Console.WriteLine(c);
                 return(n+c);
               };
```

5. Write the code for defini...

4. Coverts the following anonymous class to Lambda Expression.

```
p.SomeEvent += delegate (int n){
        Console.WriteLine("Event Occured for "+n);
};
```

```
p.SomeEvent +=
        (int n) => {
                Console.WriteLine("
                Event Occured for"+
        };
```

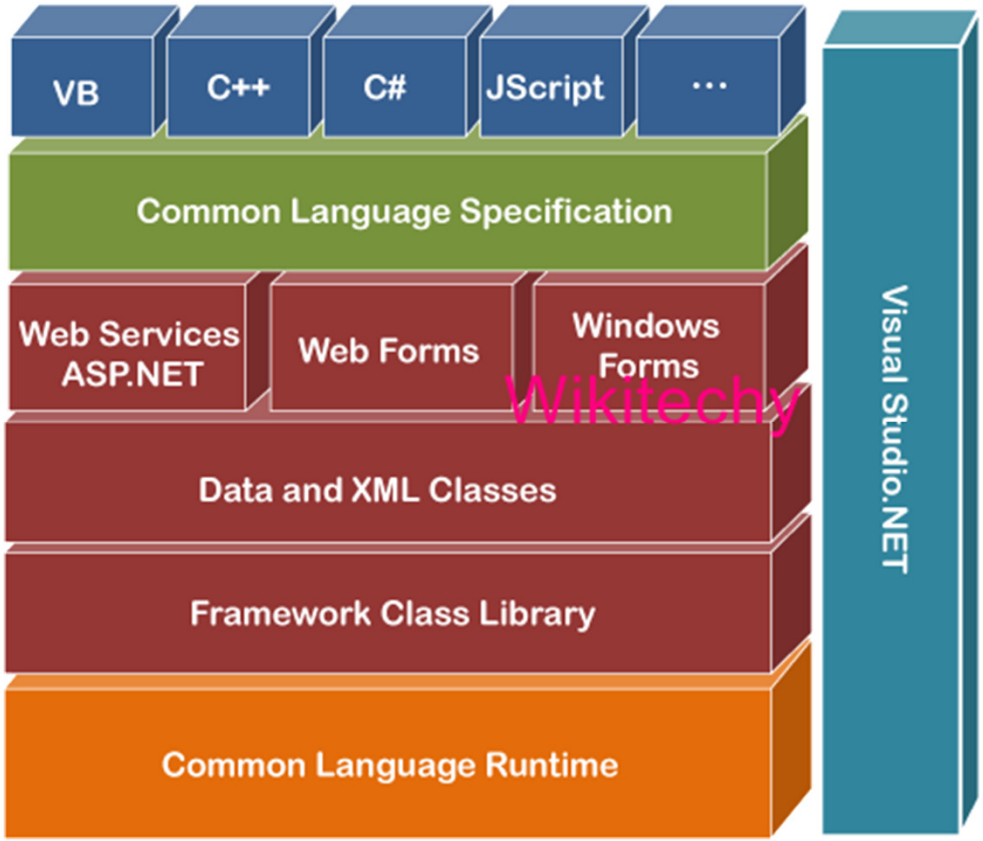
MCSE 541: Web Computing and Mining

.NET and ASP.NET Framework

Prof. Dr. Shamim Akhter
Professor, CSE, Stamford University Bangladesh

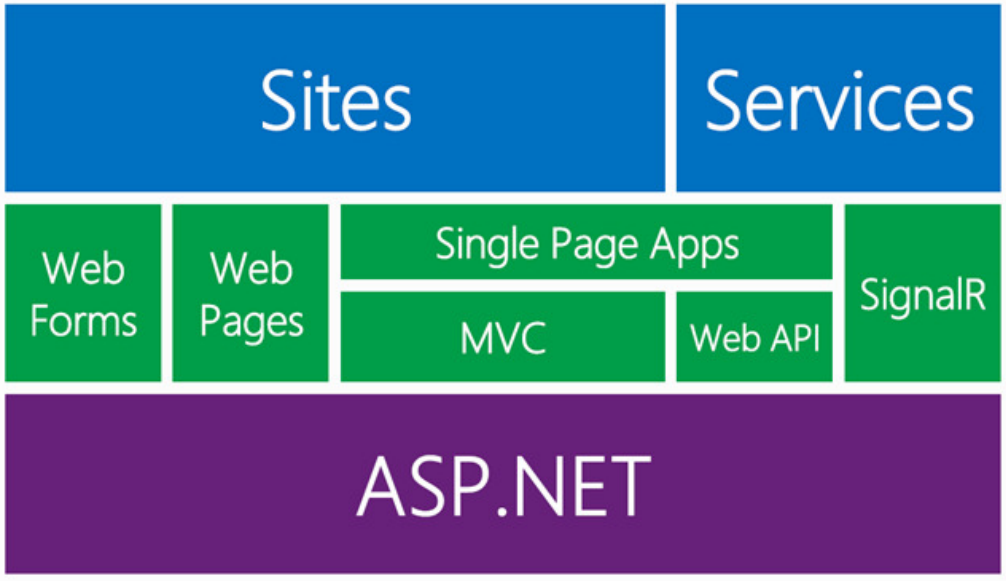
.NET?

- The Dot Net (.NET) is a open-source and cross-platform software development framework
  - integrates a number of technologies emerged from Microsoft to develop, run, and deploy web services and web or Windows applications.
- It is used to simplify the development process of multi-tier and client/server applications.
- Developers can develop big business or enterprise applications without writing code to manage security, pooling or transactions.



ASP.NET?

- The ASP.NET is a part of the .NET Framework which is used to create dynamic web pages.
- It is the latest evolution of server-side technologies and the successor to the classic Active Server Pages (ASP).
- It also simplifies the tasks of development, debugging, and deployment of web applications.



ASP.NET MVC – MVC stands for Model View Controller and allows to build web pages according to Model, View and Controller design pattern

ASP.NET Web Pages – Allows adding dynamic code and data access directly inside HTML

ASP.NET Web Forms – Allows building modular pages out of components with UI events

ASP.NET Web API – Allows developing web APIs on top of .NET framework

MICROSOFT .NET WEB STACK

- ASP.NET Web Forms was developed for two types of users:
  - **Developers who had experience** with classic Active Server Page (ASP) and were already building dynamic web sites mixing HTML and server-side code in Jscript. They were also used to interacting with the underlying HTTP connection and web server via abstractions provided by the core objects.
  - **Developers who were coming from the traditional WinForm** application development. They didn't know anything about HTML or the web and were used to building applications by dragging UI components on a design surface.

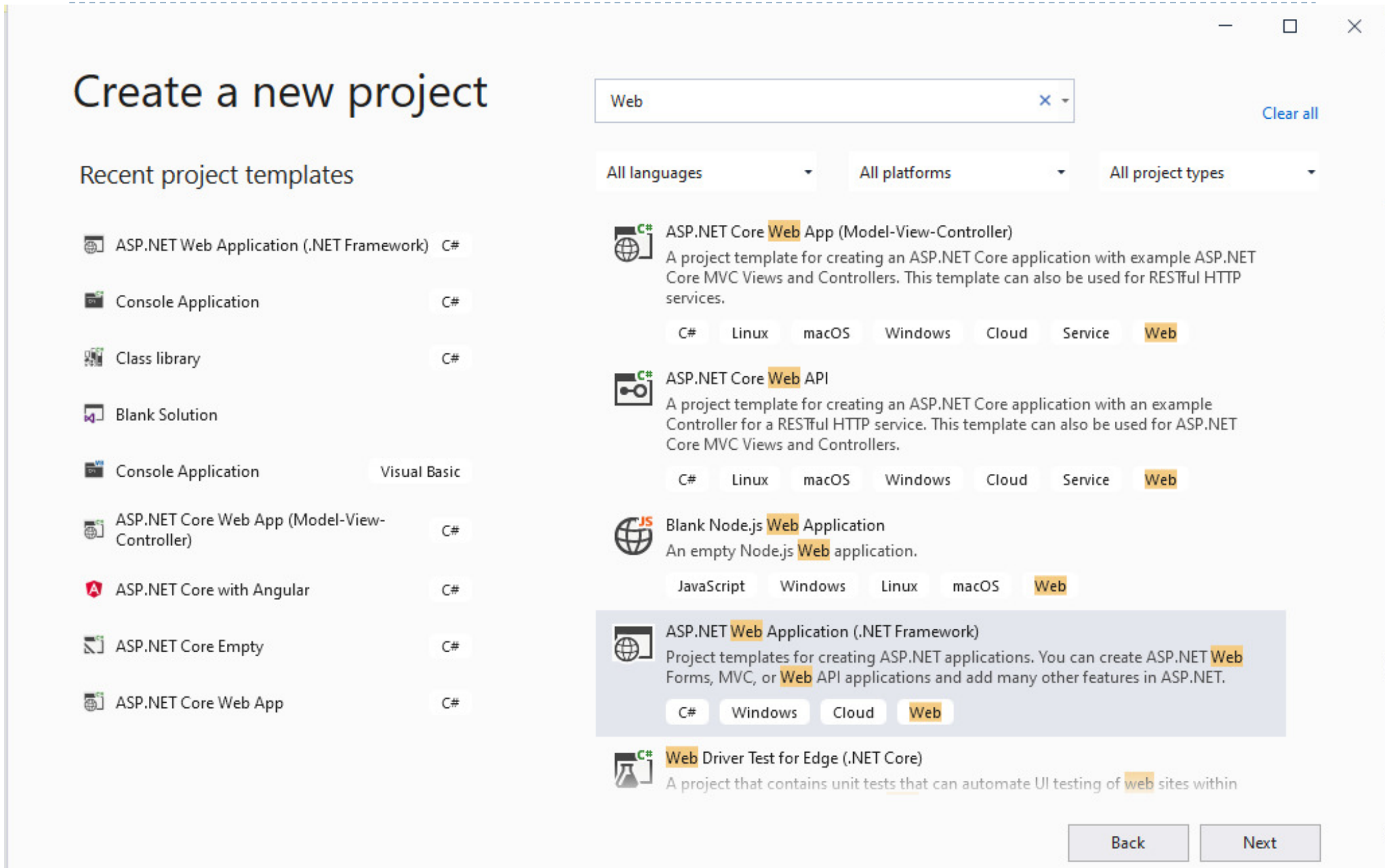
Limitations of Web Forms

All the core web abstractions were delivered within the **System.Web** library, and all the other web features depended on it.

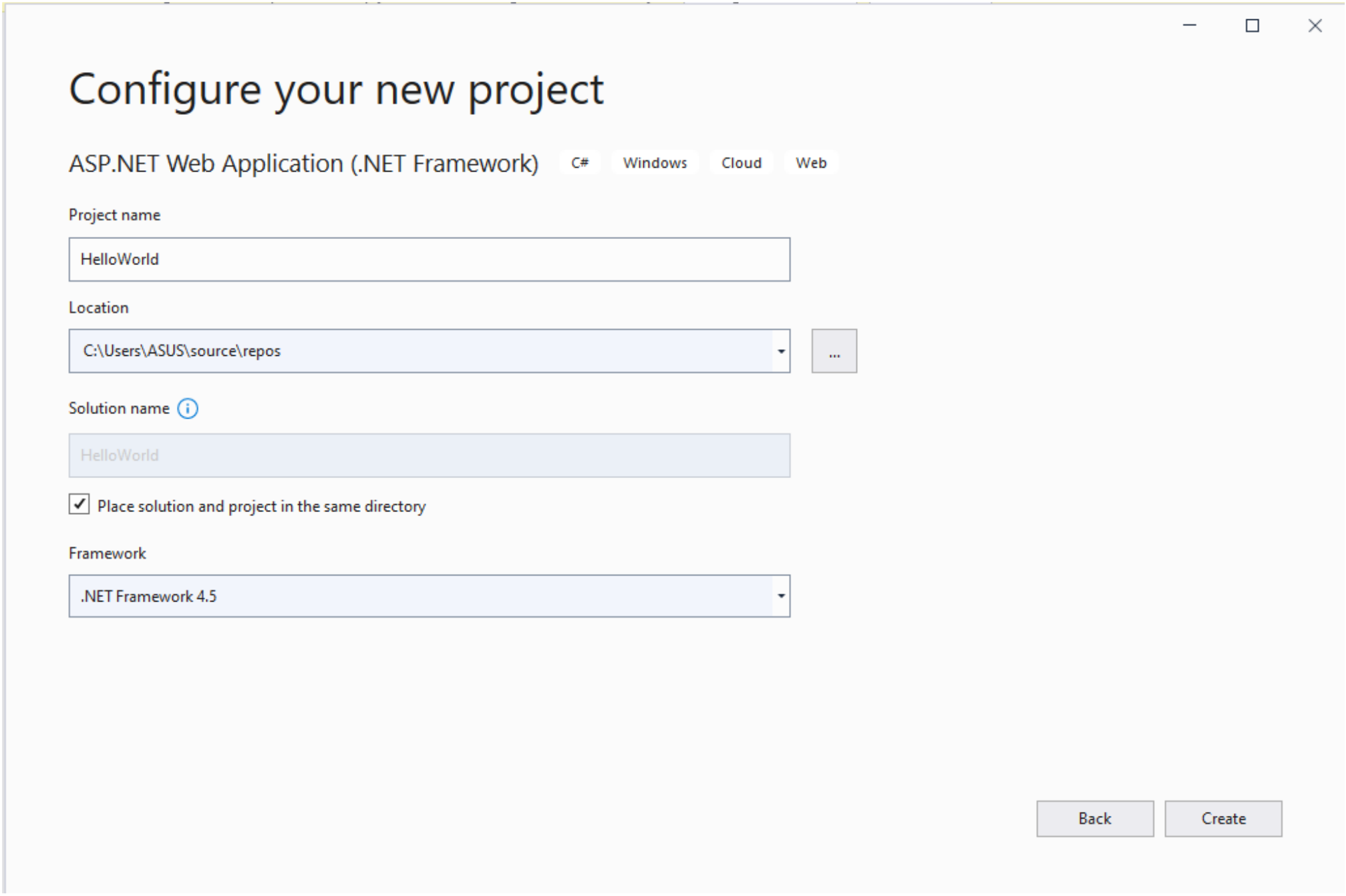
- The .NET framework and Visual Studio were intimately tied.
  - For this reason, ASP.NET had to follow the release cycle of the other products, meaning that years passed between major releases.
- ASP.NET only worked with Microsoft's web server, Internet Information Services (IIS).
- Unit testing was almost impossible and only achievable using libraries that changed the way Web Forms worked.

- The problem of release cycles:
  - Other frameworks and languages pushing the web development evolution
  - Microsoft struggle to follow them. All are very small components
    - However, ASP.NET was huge and difficult to update
- The development style was needed to change:
  - Hiding and abstracting away the complexities of HTTP and HTML markup
    - helped a lot of WinForm developers to become web developers
  - Developer (after 5 yrs.) demands more control, specially the markup rendered on pages.

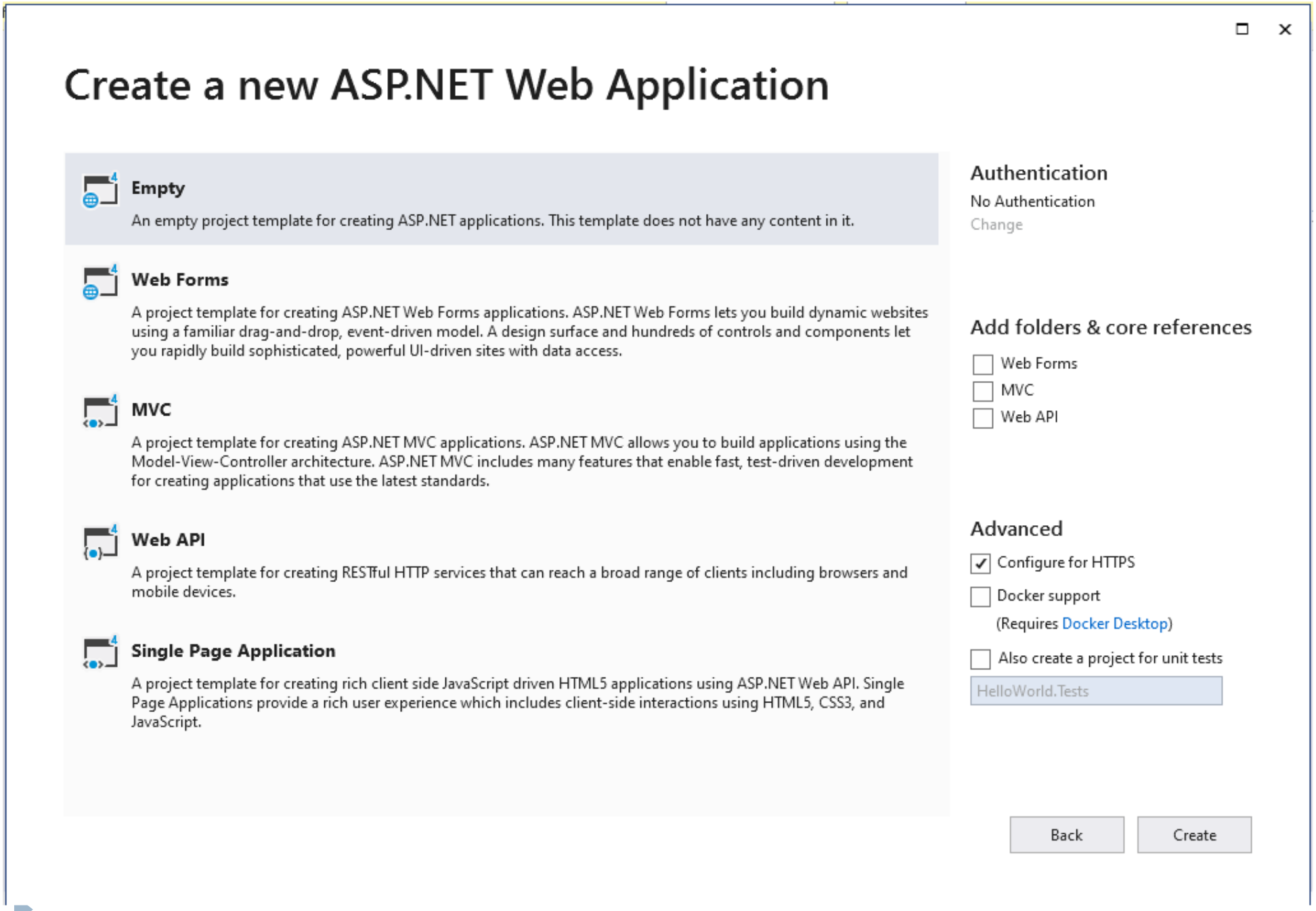
Example1: Hello World

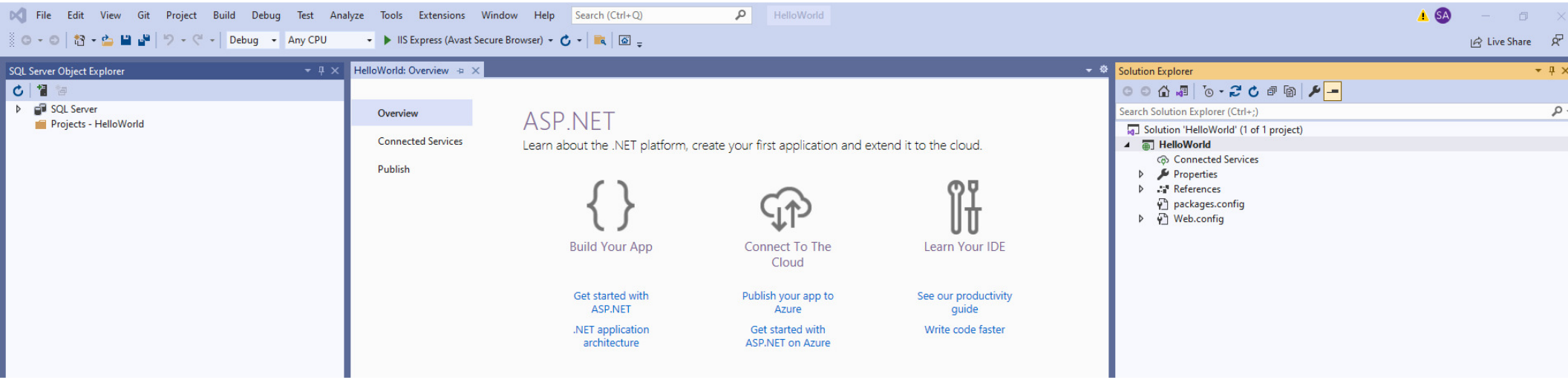


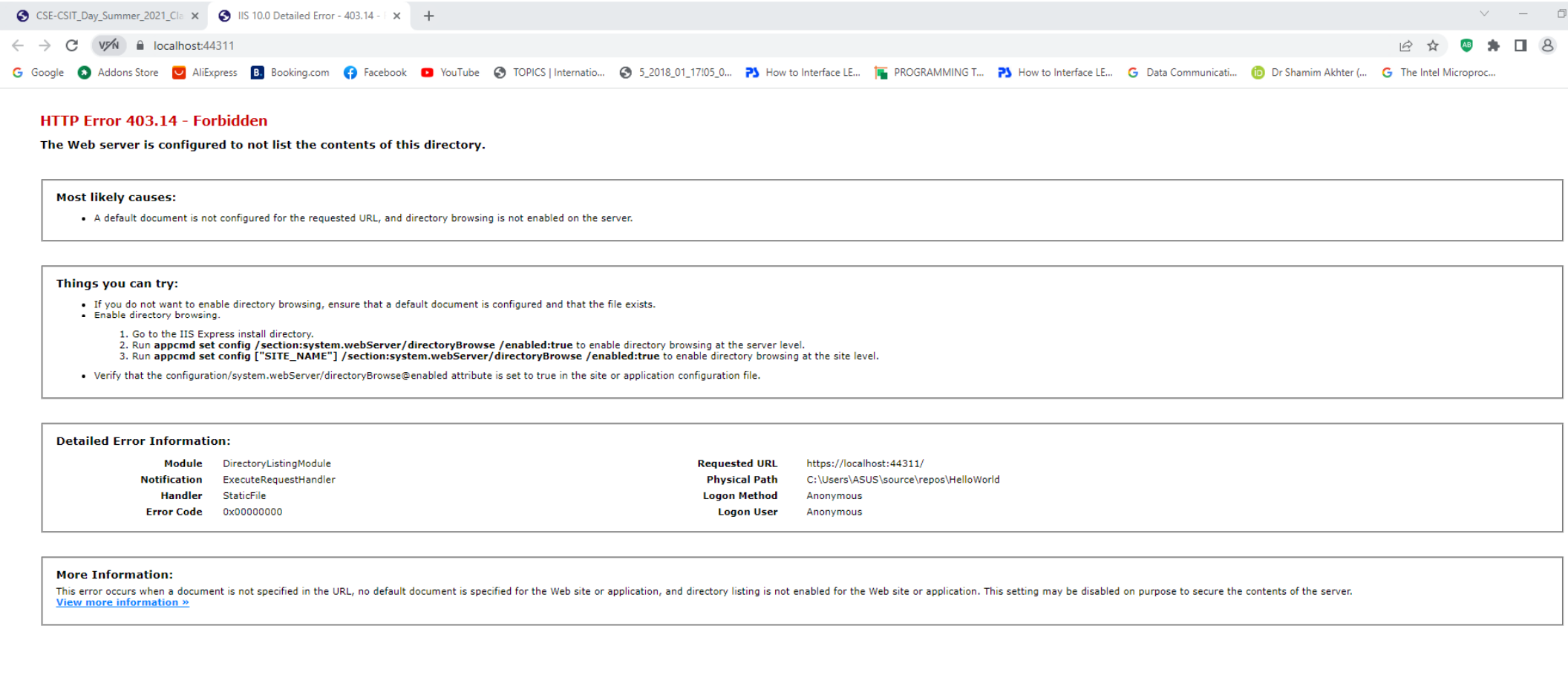
Example1: Hello World



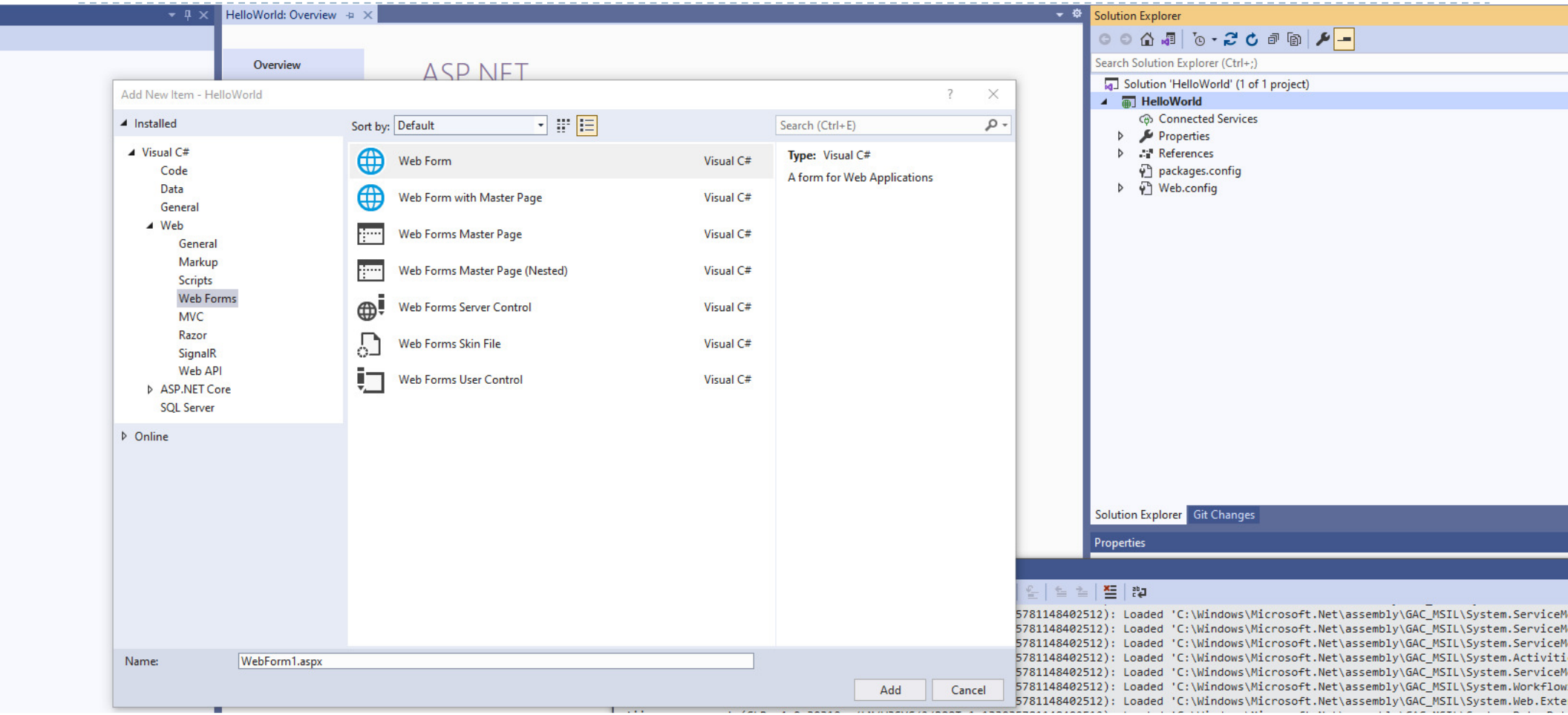
Example1: Hello World




File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
HelloWorld
Debug
Any CPU
IIS Express (Avast Secure Browser)
Live Share
SQL Server Object Explorer
SQL Server
Projects - HelloWorld
HelloWorld: Overview
Overview
Connected Services
Publish
ASP.NET
Learn about the .NET platform, create your first application and extend it to the cloud.
Build Your App
Get started with ASP.NET
.NET application architecture
Connect To The Cloud
Publish your app to Azure
Get started with ASP.NET on Azure
Learn Your IDE
See our productivity guide
Write code faster
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'HelloWorld' (1 of 1 project)
HelloWorld
Connected Services
Properties
References
packages.config
Web.config

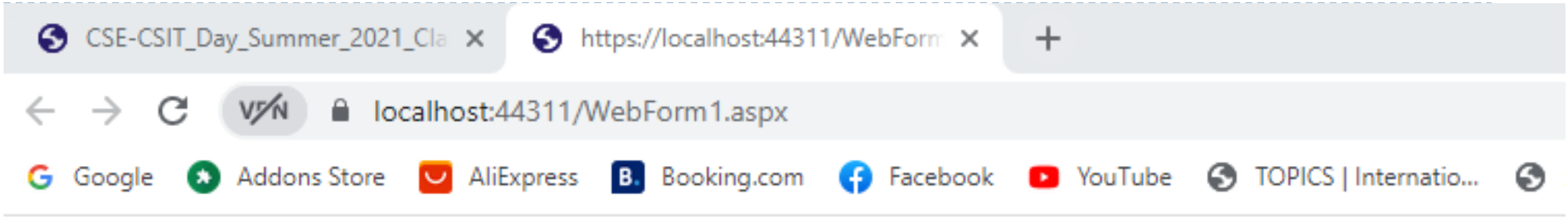

CSE-CSIT_Day_Summer_2021_Cla
IIS 10.0 Detailed Error - 403.14 -
localhost:44311
Google
Addons Store
AliExpress
Booking.com
Facebook
YouTube
TOPICS | Internatio...
5_2018_01_17!05_0...
How to Interface LE...
PROGRAMMING T...
How to Interface LE...
Data Communicati...
Dr Shamim Akhter (...
The Intel Microproc...
HTTP Error 403.14 - Forbidden
The Web server is configured to not list the contents of this directory.
Most likely causes:
A default document is not configured for the requested URL, and directory browsing is not enabled on the server.
Things you can try:
If you do not want to enable directory browsing, ensure that a default document is configured and that the file exists.
Enable directory browsing.
1. Go to the IIS Express install directory.
2. Run appcmd set config /section:system.webServer/directoryBrowse /enabled:true to enable directory browsing at the server level.
3. Run appcmd set config ["SITE_NAME"] /section:system.webServer/directoryBrowse /enabled:true to enable directory browsing at the site level.
Verify that the configuration/system.webServer/directoryBrowse@enabled attribute is set to true in the site or application configuration file.
Detailed Error Information:
Module DirectoryListingModule
Notification ExecuteRequestHandler
Handler StaticFile
Error Code 0x00000000
Requested URL https://localhost:44311/
Physical Path C:\Users\ASUS\source\repos\HelloWorld
Logon Method Anonymous
Logon User Anonymous
More Information:
This error occurs when a document is not specified in the URL, no default document is specified for the Web site or application, and directory listing is not enabled for the Web site or application. This setting may be disabled on purpose to secure the contents of the server.
View more information »

Add->New Item->Web Form




```
<body> <form id="form1" runat="server">
        <div>
            <h1> Hello World </h1>
        </div>
    </form>
</body>
```

Output



Hello World

ASP.NET MVC

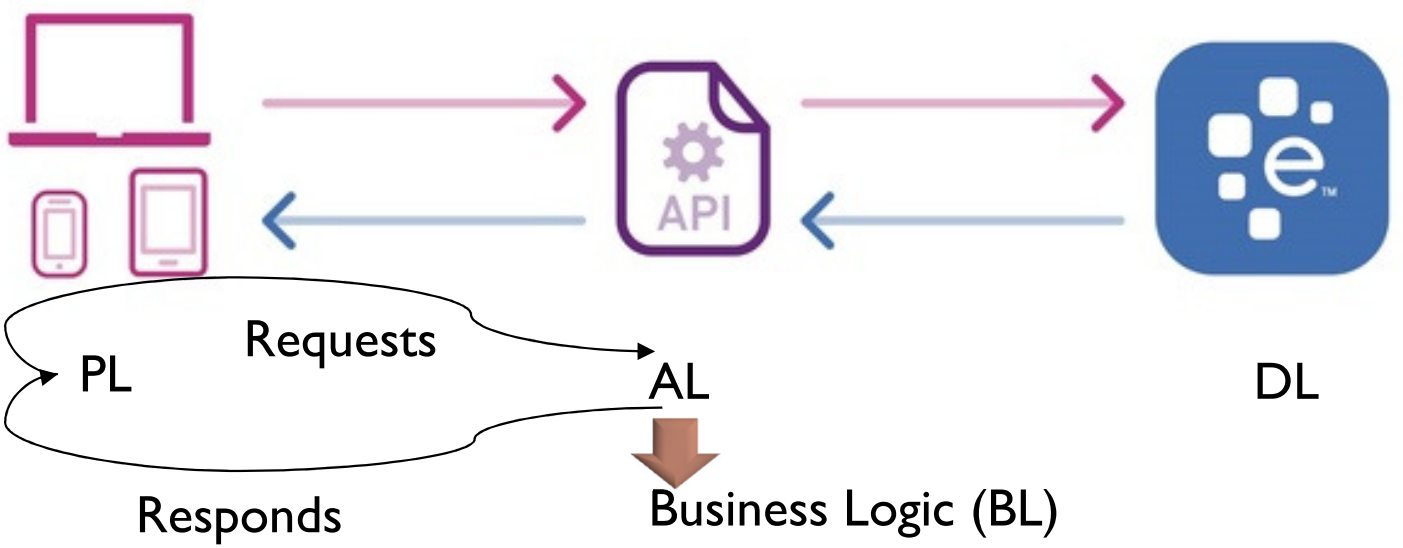
- Model-View-Controller design pattern
- A cleaner and better separation of business and presentation logic,
- Remove the server-side UI components, it gave complete control of the HTML markup to the developers.
- Furthermore, instead of being included inside the .NET framework, it was released out of band, making faster and more frequent releases possible.
- Limitation:
  - Although the ASP.NET MVC framework solved most of the problems of Web Forms, it still depended on IIS and the web abstracting library System.Web. This means that it was still not possible to have a web framework that was totally independent from the larger .NET framework.

ASP.NET Web API

- Build an even more modular component model,
  - finally ditching **System.Web** and creating a web framework that could live its own life independently from the rest of ASP.NET and the larger .NET framework.
  - A big role was also played by the introduction of NuGet, Microsoft's package distribution system, making it possible to deliver all these components to developers in a managed and sustainable way.
  - One additional advantage of the break-up from System.Web was the capability to not depend on IIS anymore and to run inside custom hosts and possibly other web servers.

Web API

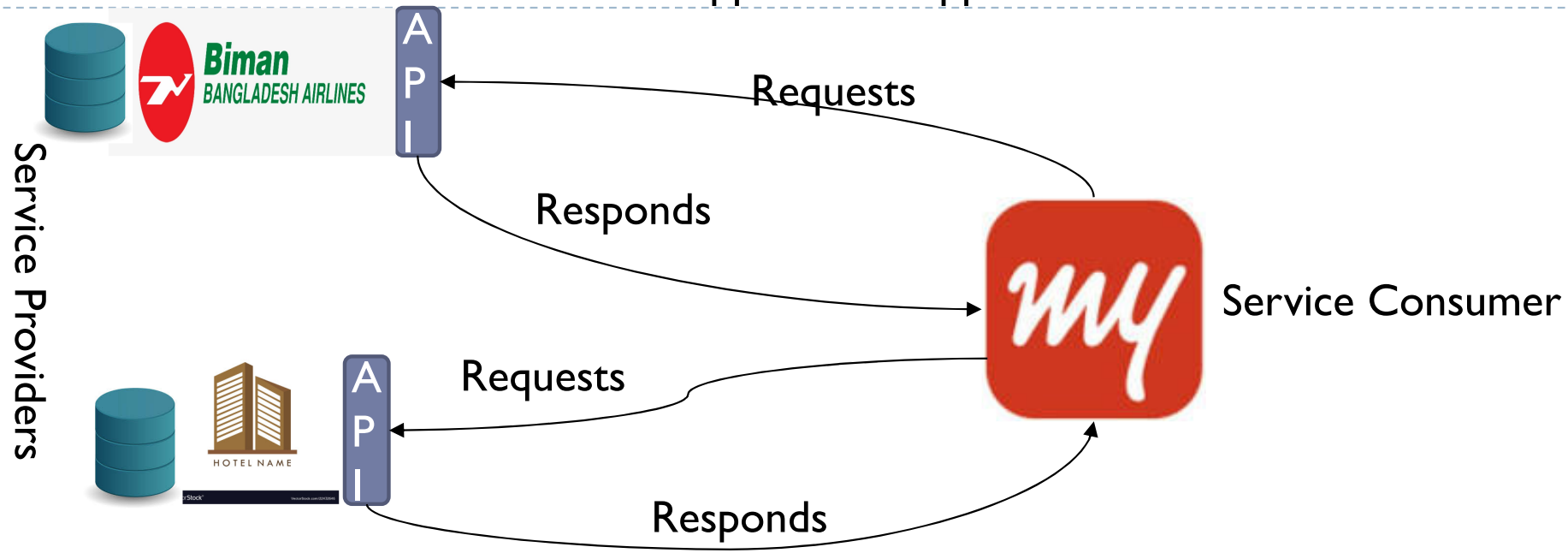
- stands for **Application Programming Interface**



Business Logic (BL)
Present them as functions or modules
-API, part of application layer and consist program
-Enable communication or exchange data between two software systems

How API Works

Communication
Application- Application



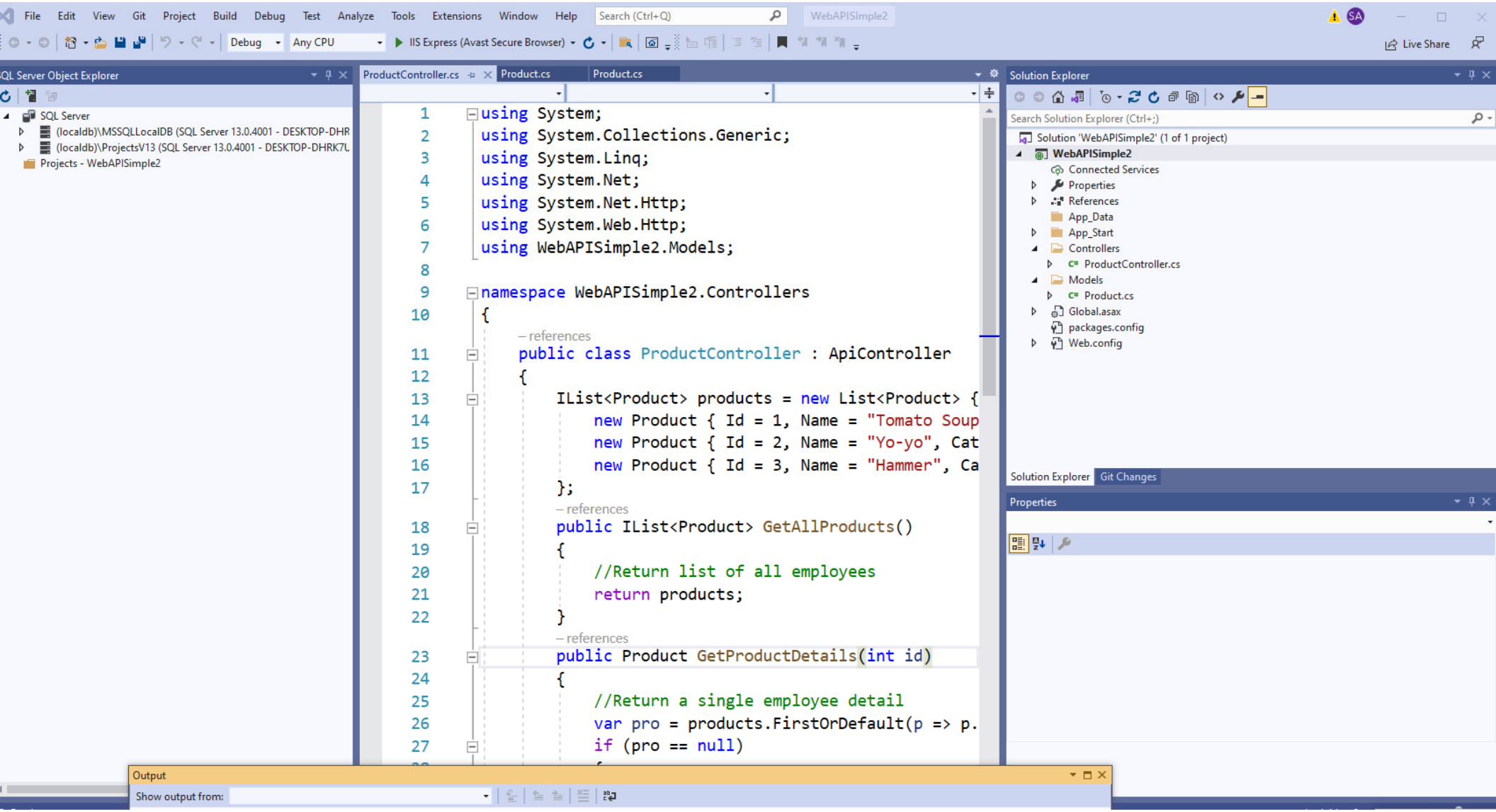
APIs are available in the Internet is called Web Services

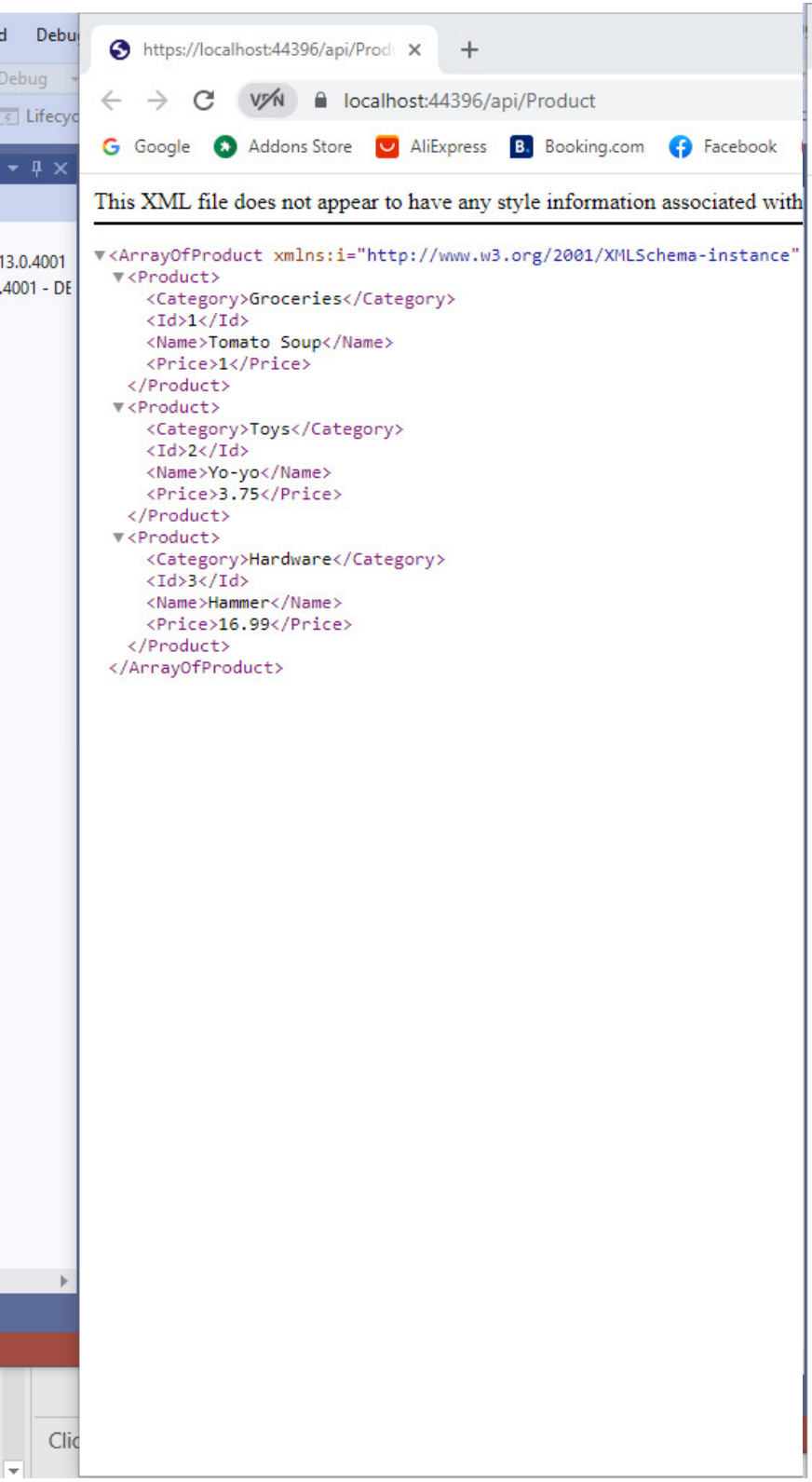
Auto-implemented properties

- Auto-implemented properties in C# make code more readable and clean if there is no additional calculation needed. Compiler creates a private anonymous field that can only be accessed through the get and set accessors.

```
public int Id
        {
            get;
            set;
        }
public string Name
        {
            get;
            set;
        }
```

Example: Web API




https://localhost:44396/api/Prod
localhost:44396/api/Product
Google
Addons Store
AliExpress
Booking.com
Facebook
This XML file does not appear to have any style information associated with
<ArrayOfProduct xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
<Product>
<Category>Groceries</Category>
<Id>1</Id>
<Name>Tomato Soup</Name>
<Price>1</Price>
</Product>
<Product>
<Category>Toys</Category>
<Id>2</Id>
<Name>Yo-yo</Name>
<Price>3.75</Price>
</Product>
<Product>
<Category>Hardware</Category>
<Id>3</Id>
<Name>Hammer</Name>
<Price>16.99</Price>
</Product>
</ArrayOfProduct>


https://localhost:44396/api/Prod
localhost:44396/api/Product/1
Google
Addons Store
AliExpress
Booking.com
Facebook
YouTube
TOPICS | Internatio...
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<Product xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/20
<Category>Groceries</Category>
<Id>1</Id>
<Name>Tomato Soup</Name>
<Price>1</Price>
</Product>

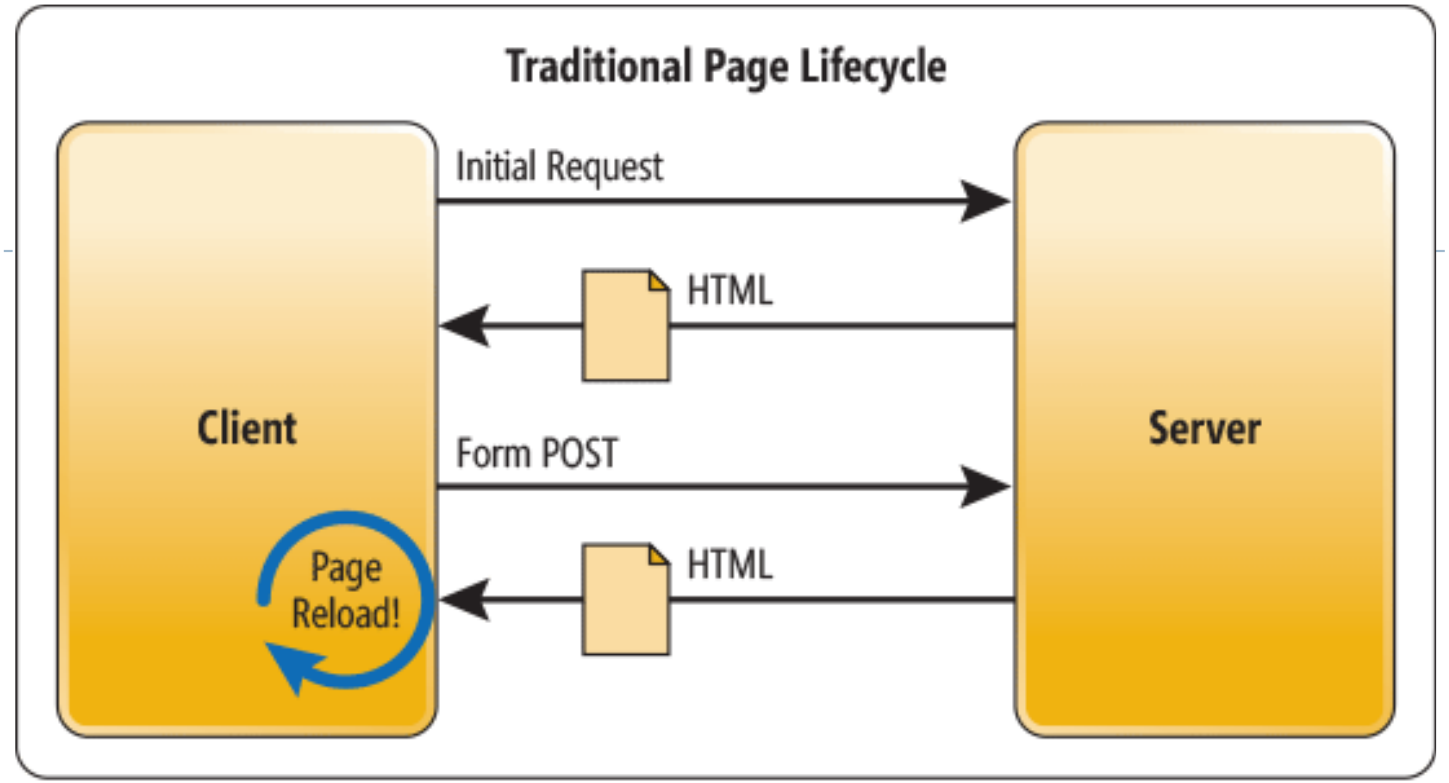
Example

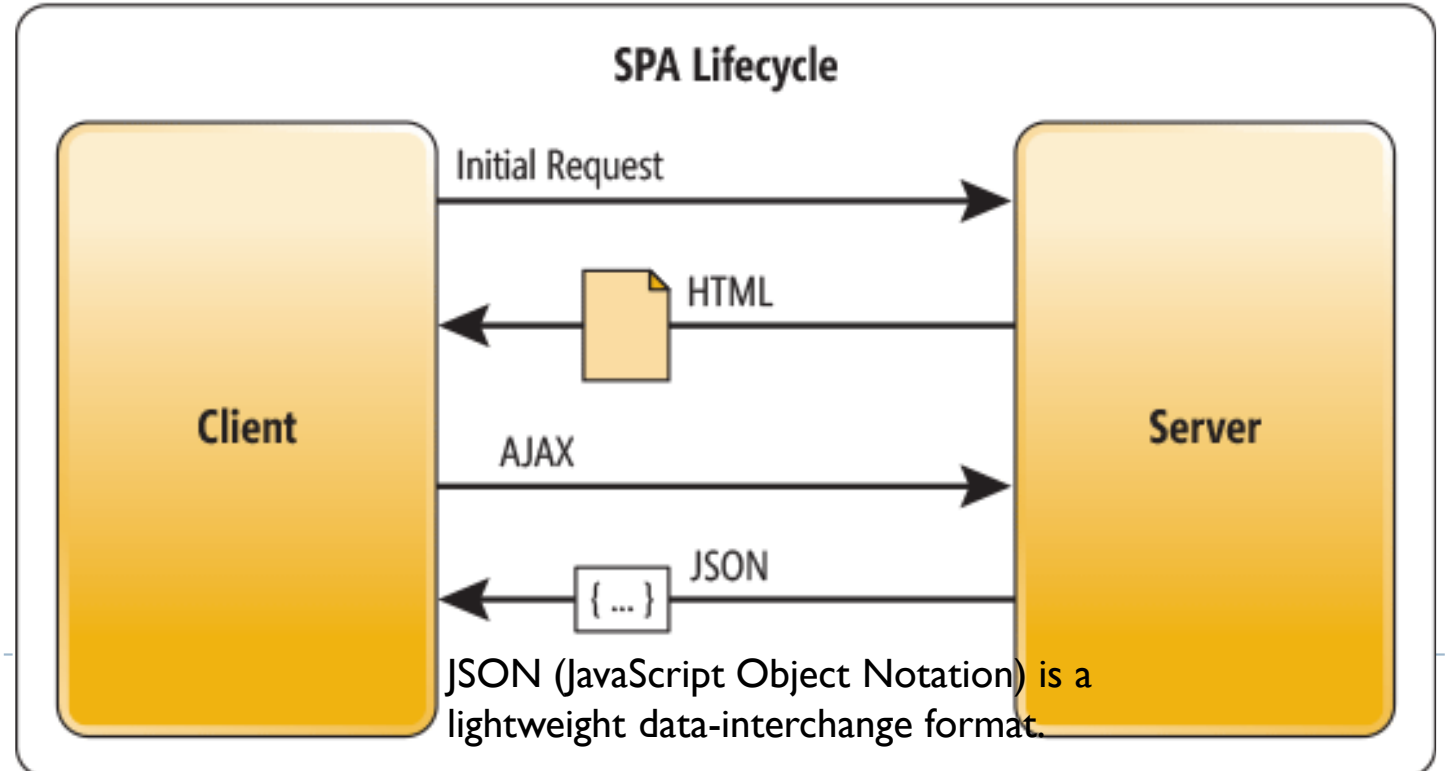
- https://www.c-sharpcorner.com/UploadFile/8a67c0/getting-started-with-web-api-step-by-step-with-sample-applic/

Single Page Applications (SPAs).

- Instead of interconnected, server-generated, data-driven pages, applications were becoming mostly static pages where data was displayed interacting with the server via AJAX(Asynchronous JavaScript and XML) calls to web services or Web APIs. Also, many services started releasing APIs for mobile apps or third-party apps to interact with their data.

As the names suggests, it is a single page where all information is available on the same page.





JSON (JavaScript Object Notation) is a lightweight data-interchange format.

Traditional

Every request for new information gives you a new version of the whole page.

Single Page Application

You request just the pieces you need.

A single-page application (SPA) is an app that works inside the browser and reduces reloading requirements. It is one page that is visited and then with the aid of JavaScript, it is loaded all other content.

- These types of applications are accessible and avoid the waiting time for the user. Examples are Gmail, Google Maps and Facebook.

OWIN and Katana @.Net Framework

- System.Web is something that has existed ever since ASP (non .NET version) and internally contains many things that you might not even need (such as Web Forms or URL Authorization), which by default all run on every request, thus consuming resources and making ASP.NET applications in general lot slower than its counterparts such as Node.js for example.
- Open Web Interface for .NET (OWIN) is a specification on how web servers and web applications should be built in order to decouple them and allow movement of ASP.NET applications to environments which were not supported before.
- Katana on the other hand, is a fully developed framework made to make a bridge between current ASP.NET frameworks and OWIN specification. At the moment, Katana has successfully adapted the following ASP.NET frameworks to OWIN:
  - Web API
  - Signal R
- ASP.NET MVC and Web Forms are still running exclusively via System.Web, and in the long run there is a plan to decouple those as well.

- Katana is slowly getting retired. Version 3.0 will most likely be last major release of Katana as a standalone framework.

- ASP.NET 5 is the successor to Katana. Katana was the beginning of the break away from System.Web and to more modular components for the web stack. You can see vNext/5 as a continuation of that work but going much further (new CLR, new Project System, new http abstractions). ASP.NET 5 was built on top of .NET Core 5. NET Core 5 was lightweight factored version of .NET Framework.

- After ASP.NET 5 ASP.NET Core 1.0 and .NET Core 1.0 were introduced.

MCSE 541:Web Computing and Mining

ASP.NET Framework to ASP.NET Core

Prof. Dr. Shamim Akhter
Professor, CSE, Stamford University Bangladesh

OWIN and Katana @.Net Framework

- System.Web is something that has existed ever since ASP (non .NET version) and internally contains many things that you might not even need (such as Web Forms or URL Authorization), which by default all run on every request, thus consuming resources and making ASP.NET applications in general lot slower than its counterparts such as Node.js for example.
- Open Web Interface for .NET (OWIN) is a specification on how web servers and web applications should be built in order to decouple them and allow movement of ASP.NET applications to environments which were not supported before.
- Katana on the other hand, is a fully developed framework made to make a bridge between current ASP.NET frameworks and OWIN specification. At the moment, Katana has successfully adapted the following ASP.NET frameworks to OWIN:
  - Web API
  - Signal R
- ASP.NET MVC and Web Forms are still running exclusively via System.Web, and in the long run there is a plan to decouple those as well.

Katana and ASP.NET 5

- Katana is slowly getting retired. Version 3.0 was the last major release of Katana as a standalone framework.

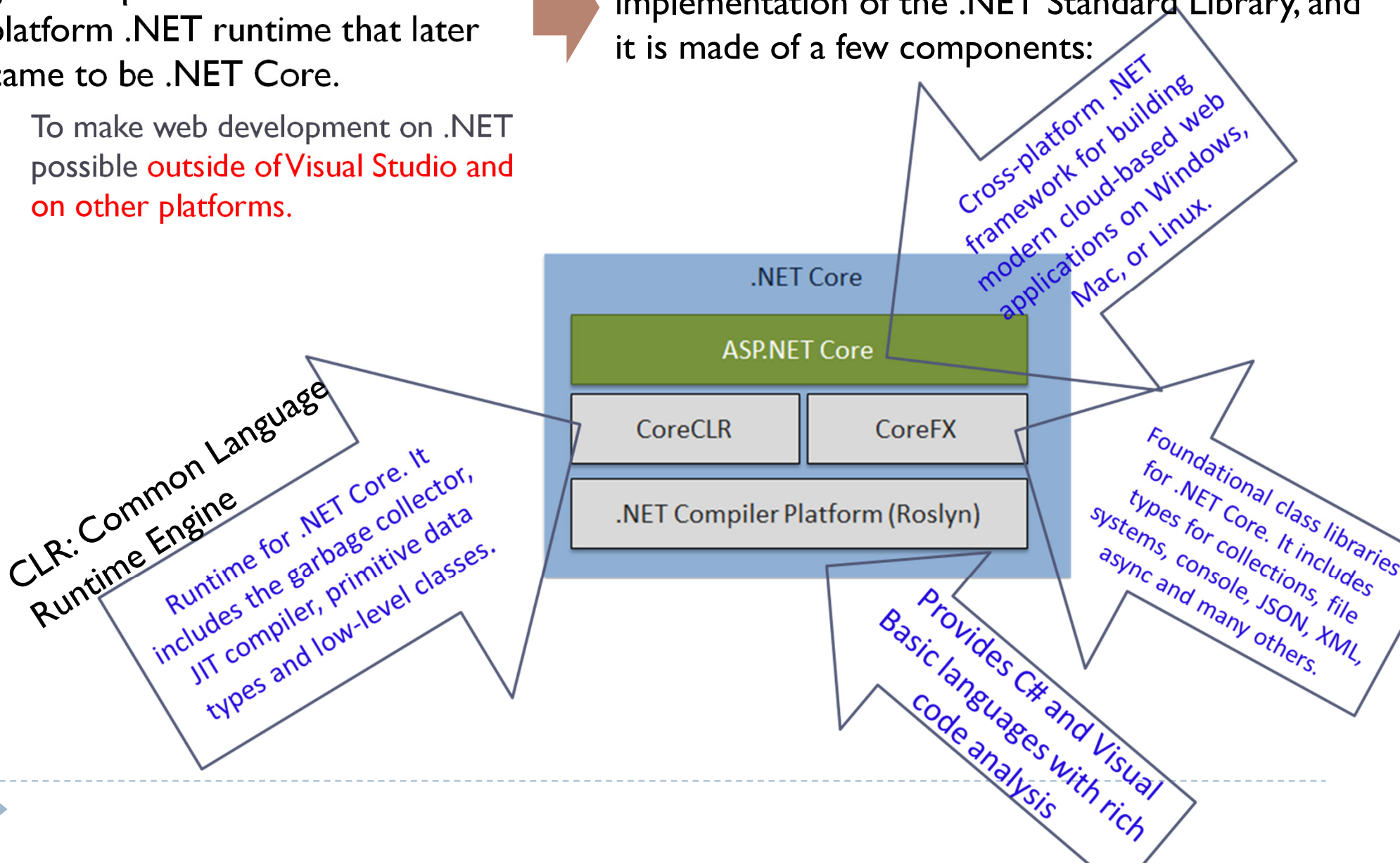
- ASP.NET 5 is the successor to Katana.
  - Katana was the beginning of the break away from System.Web and to more modular components for the web stack.
  - vNext/5 as a continuation of that work but going much further (new CLR, new Project System, new http abstractions).
  - ASP.NET 5 was built on top of .NET Core. This .NET Core was lightweight factored version of .NET Framework.

- After ASP.NET 5 ASP.NET Core 1.0 and .NET Core 1.0 were introduced.

Emergence of ASP.NET Core and .NET Core

- They re-wrote ASP.NET from the ground up and created a new cross-platform .NET runtime that later came to be .NET Core.
  - To make web development on .NET possible outside of Visual Studio and on other platforms.

.NET Core is a cross-platform and opensource implementation of the .NET Standard Library, and it is made of a few components:



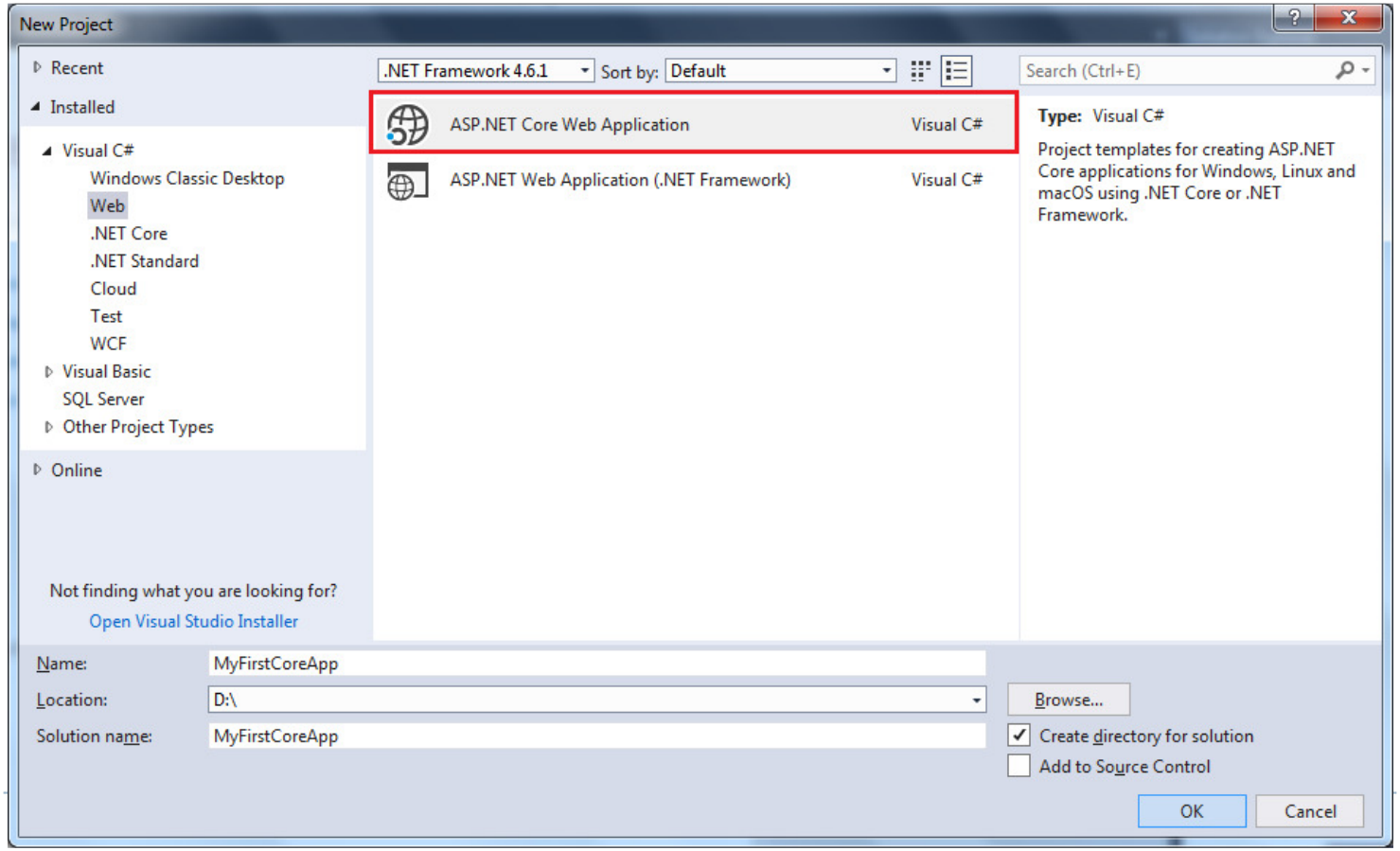
ASP.NET Core

- ASP.NET Core is a newer version of ASP.NET 4.x Framework
- High Performance, Cross Platform and open-source web framework
  - run able on Windows, Mac, Linux, Android, or iOS.
  - hosted on Internet Information Services (IIS), Apache, Docker –PaaS
  - Unified programming model for MVC controller and Web API
    - Inherits from same controller classes
    - Returns IActionResult(View Result, JSON Result)
  - Service available via dependency injection
    - configure and register a service, any controller can use it.
  - Modular framework distributed as NuGet packages
    - allows us to include packages that are required in our application.

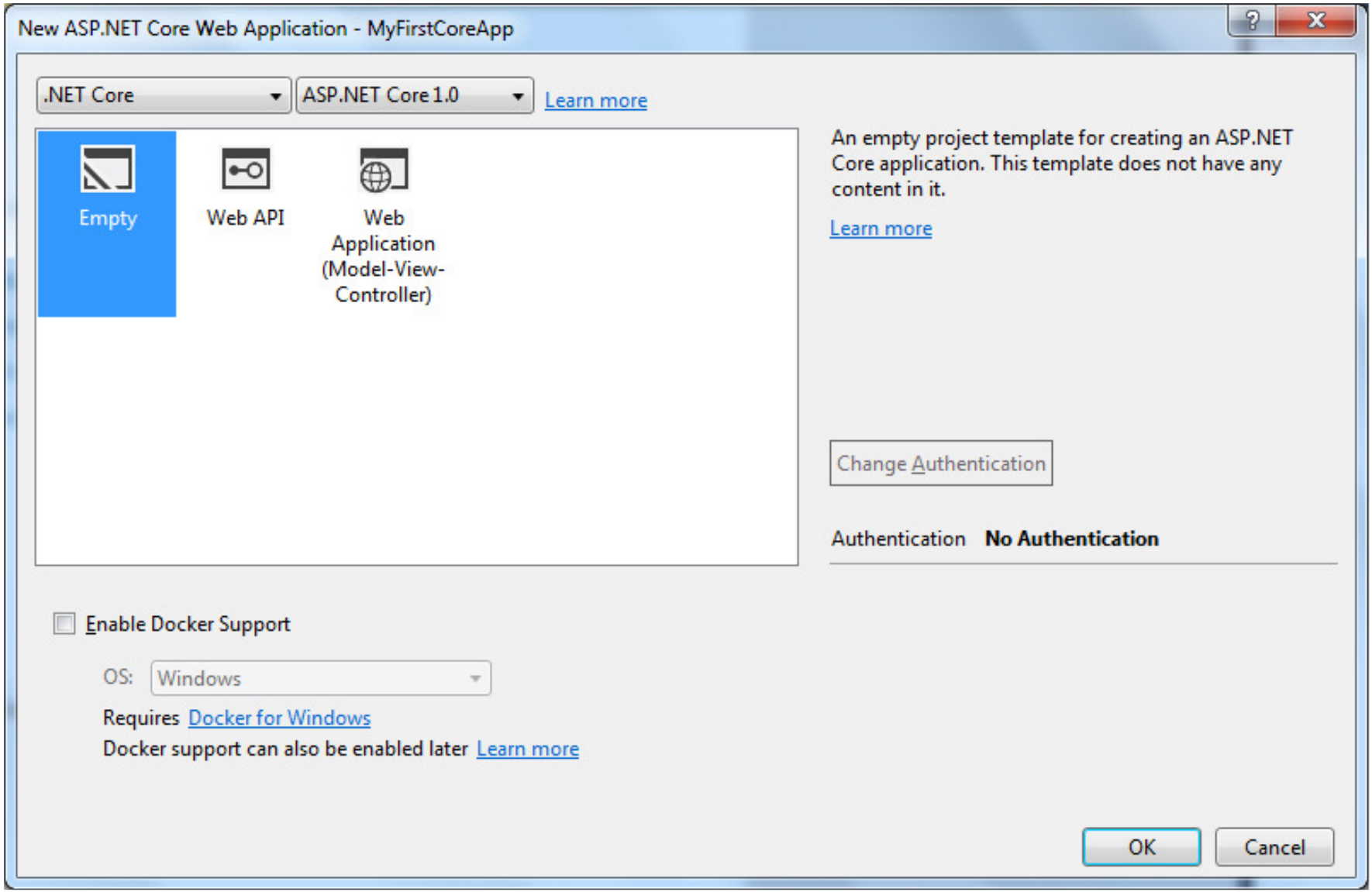
https://www.infoworld.com/article/3232636/how-to-use-dependency-injection-in-aspnet-core.html

First ASP.NET Core 2.0 Application

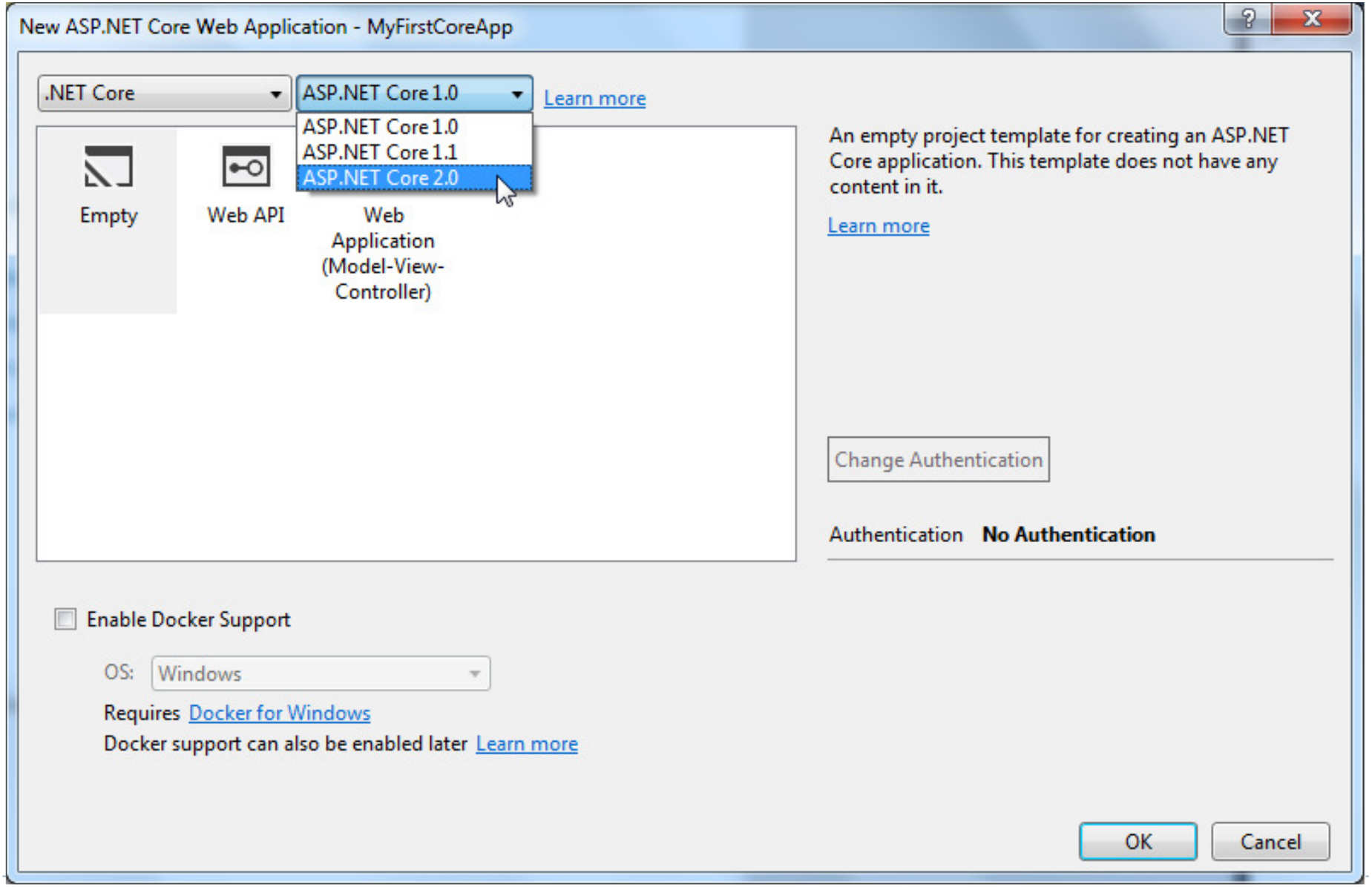
- The first step is to open Visual Studio. Click on File->New, and click on Projects.
- In the New Project dialog box, click on the Templates node. Expand the Templates node, then expand Visual C#, and click on the Web template.



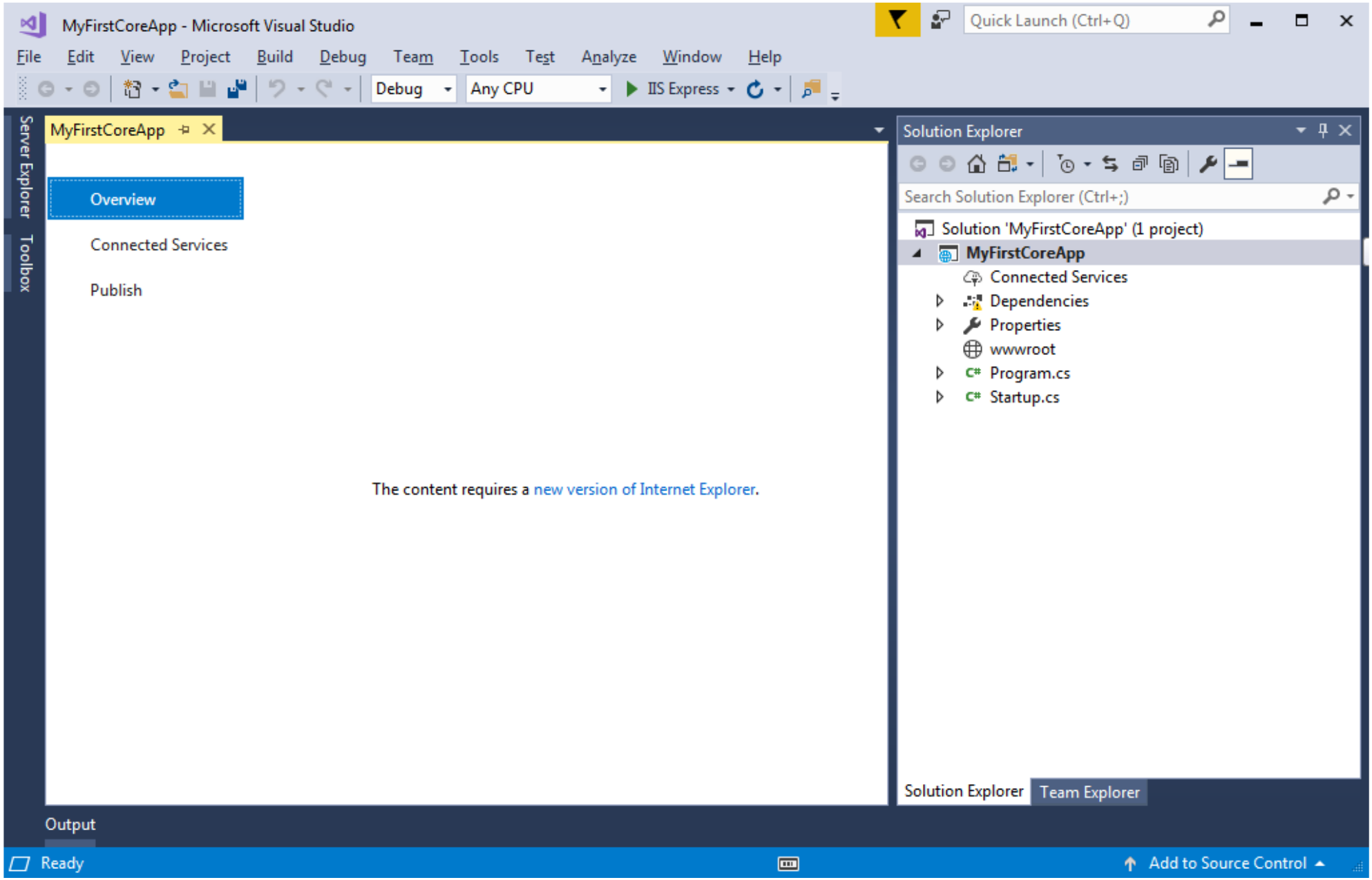
First ASP.NET Core 2.0 Application



First ASP.NET Core 2.0 Application

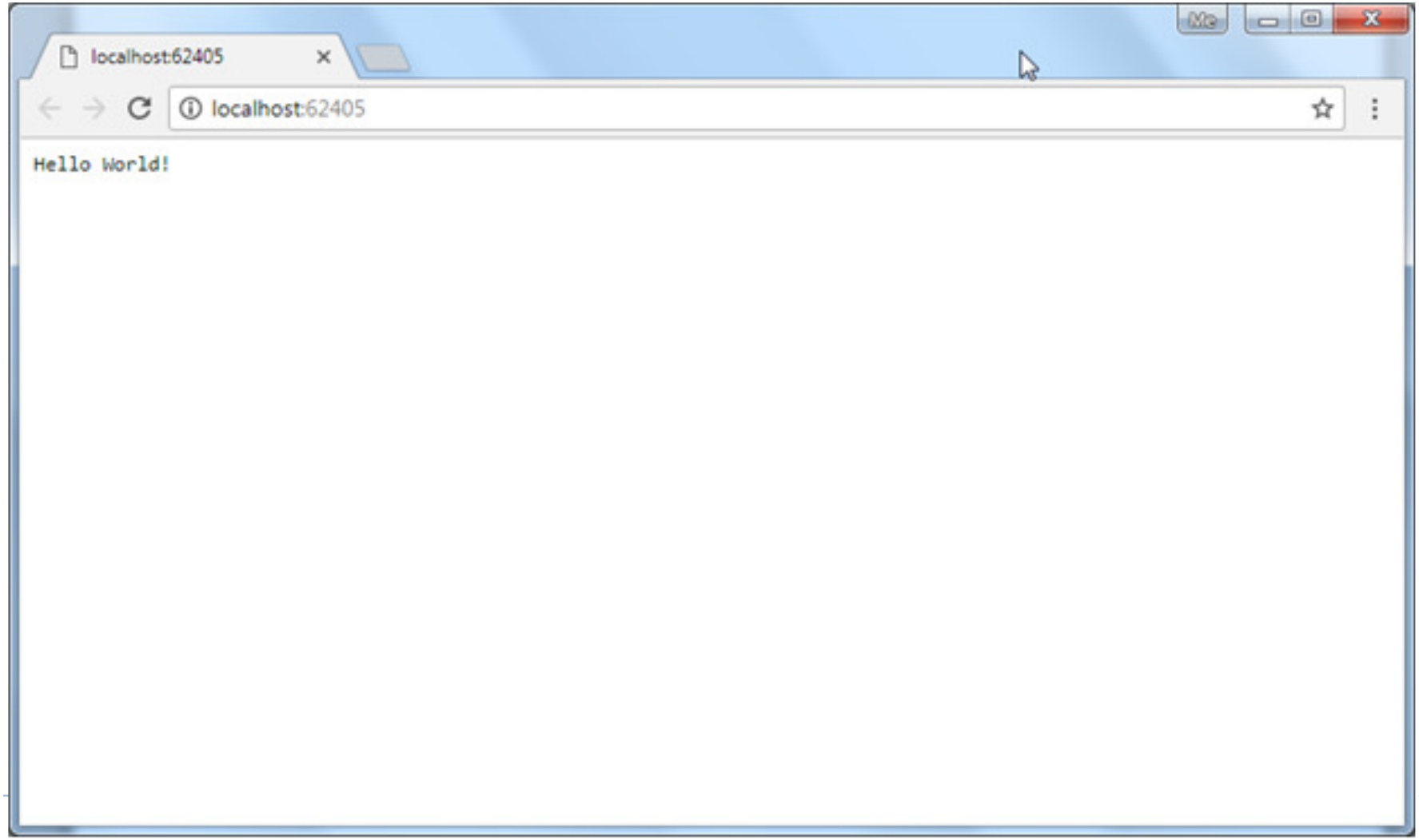


First ASP.NET Core 2.0 Application

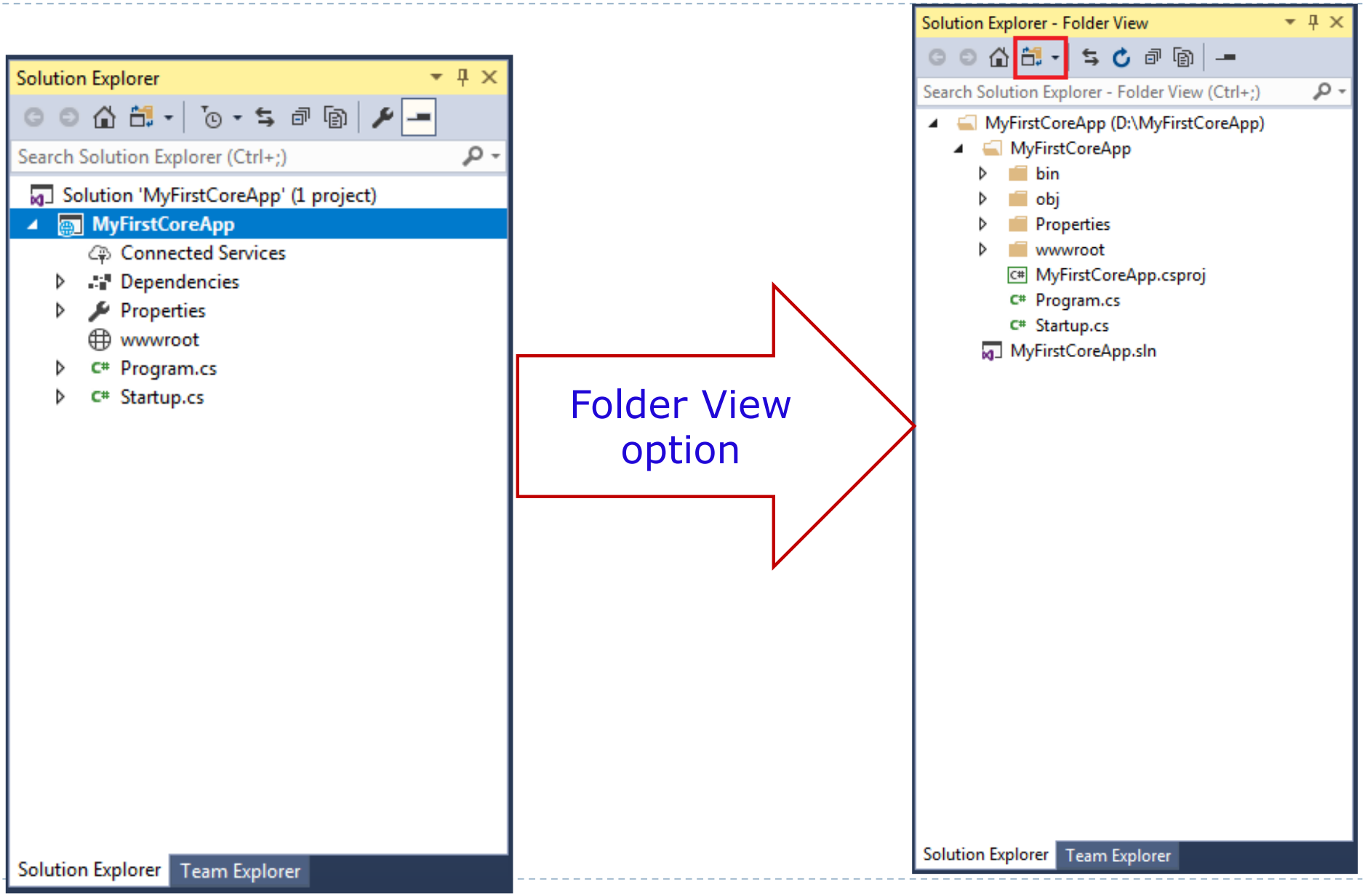


First ASP.NET Core 2.0 Application

To run this web application, go to Debug menu and click on Start without Debugging, or press Ctrl + F5. This will open the browser and display the following result.



A Default Project Structure



Solution View

Starting from the top, the first new element is the Connected Services node, which contains the list of extensions that connect to a third party remote service.

The next element is a node called Dependencies. This contains all the dependencies the application has, which can be .NET packages (via NuGet), Bower, as shown in Figure 1-4, or NPM if you application needs it.

Bower is a "package manager for the web." Bower lets you install and restore client-side packages, including JavaScript and CSS libraries.

A reference to Bower appears also later in the tree with the file `bower.json`, which contains the actual configuration of all the dependencies. These dependencies, once downloaded, will be stored in the `lib` folder inside the new `wwwroot` folder.

The next element is the `wwwroot` folder, which is even represented with a different "globe" icon. This is where all the static files of the application, CSS styles, images and JavaScript files, will be.

These files in the root of the project are also new additions:

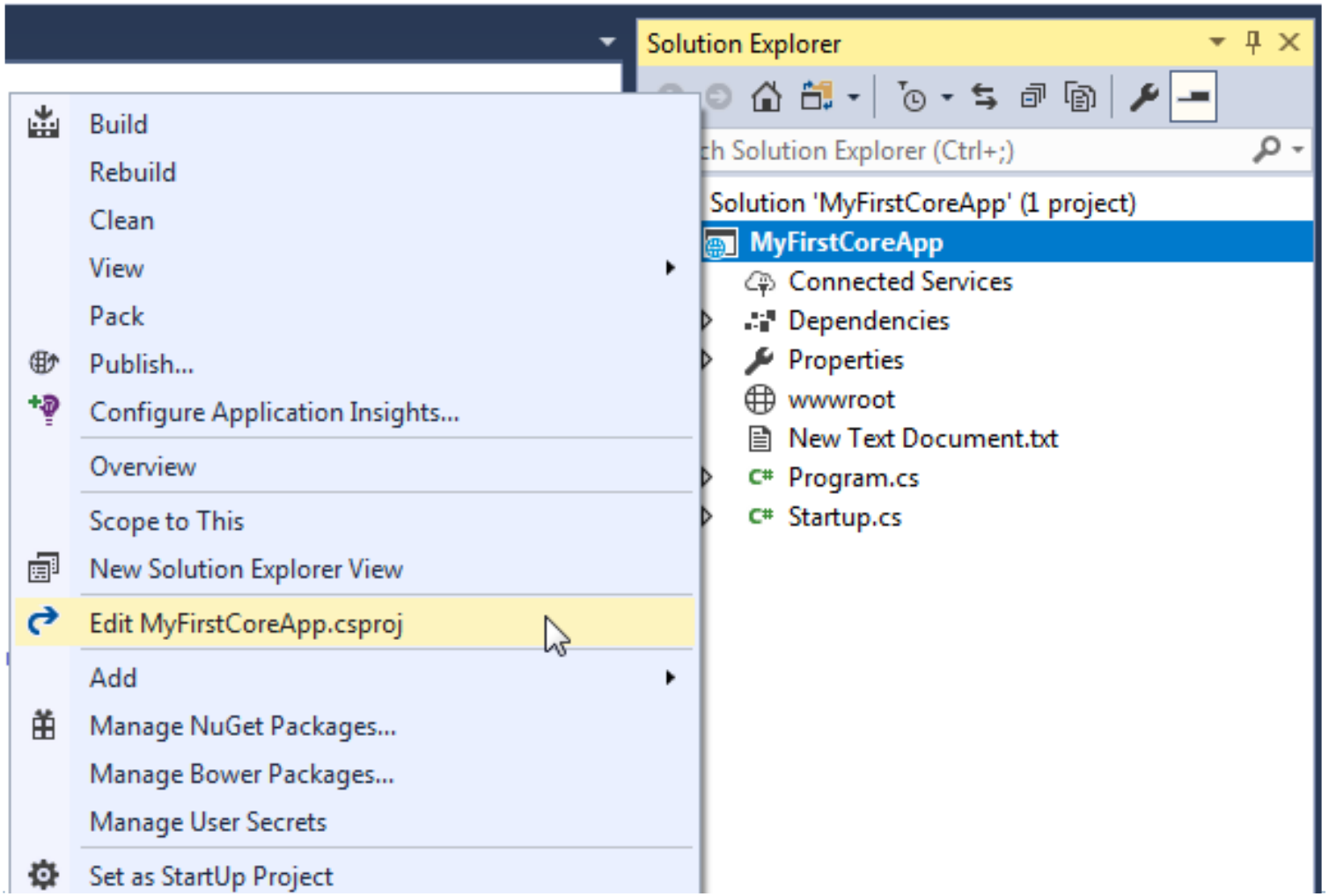
- `appsettings.json` is the new location for storing application settings instead of storing them in the `appsetting` element in the `web.config`.
- `bower.json` is the configuration file for Bower dependencies.
- `bundleconfig.json` defines the configuration for bundling and minifying JavaScript and CSS files.

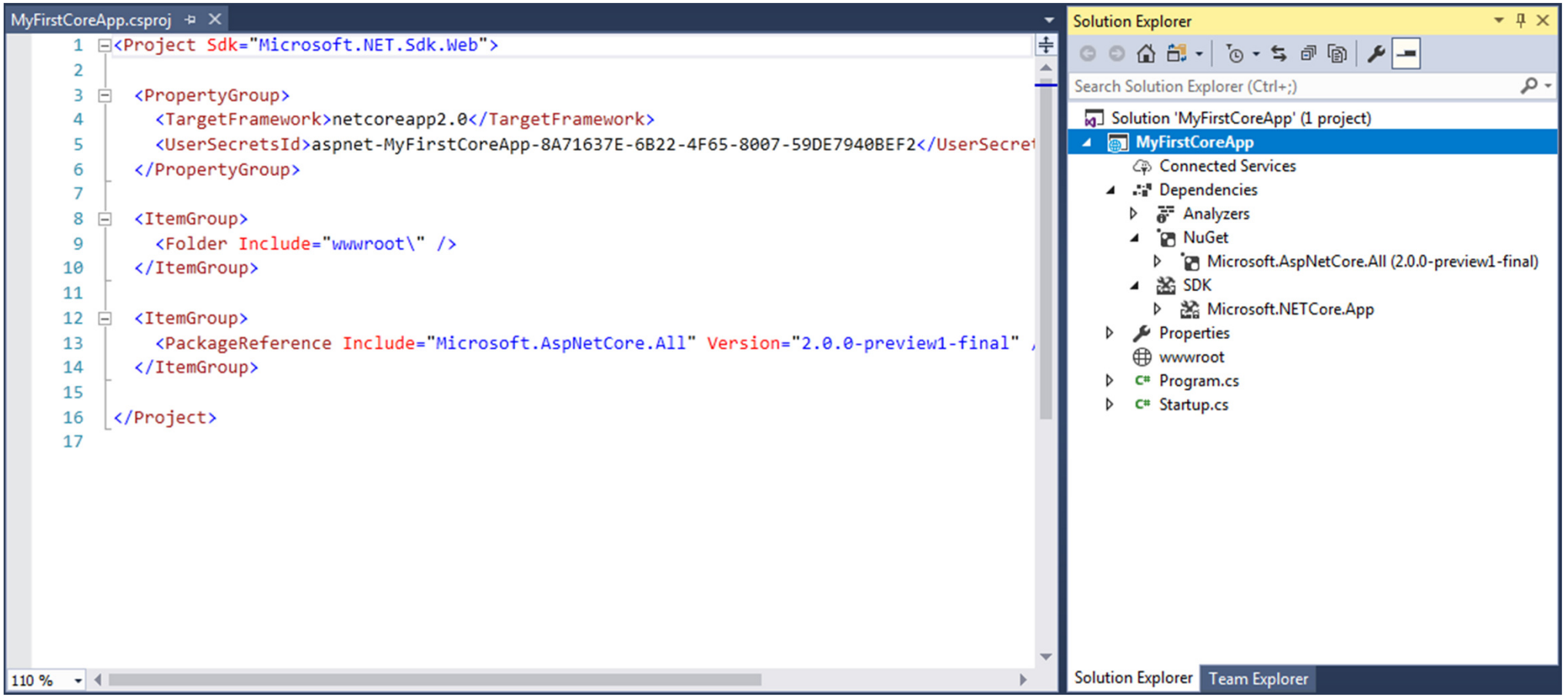
- `Program.cs` is where the web application starts. As mentioned earlier, the .NET Core app host can only start console applications, so web projects also need an instance of `Program.cs`.
- `Startup.cs` is the main entry point for ASP.NET Core web applications. It is used to configure how the application behaves. Thus the `Global.asax` file, which was used for this purpose before, has disappeared.
- `web.config` disappeared as it's not needed any more.

Folder View

.csproj

- ASP.NET Core 1.0 uses .xproj and project.json files to manage the project.
- ASP.NET Core 2.0.& Core 3.1 use .csproj



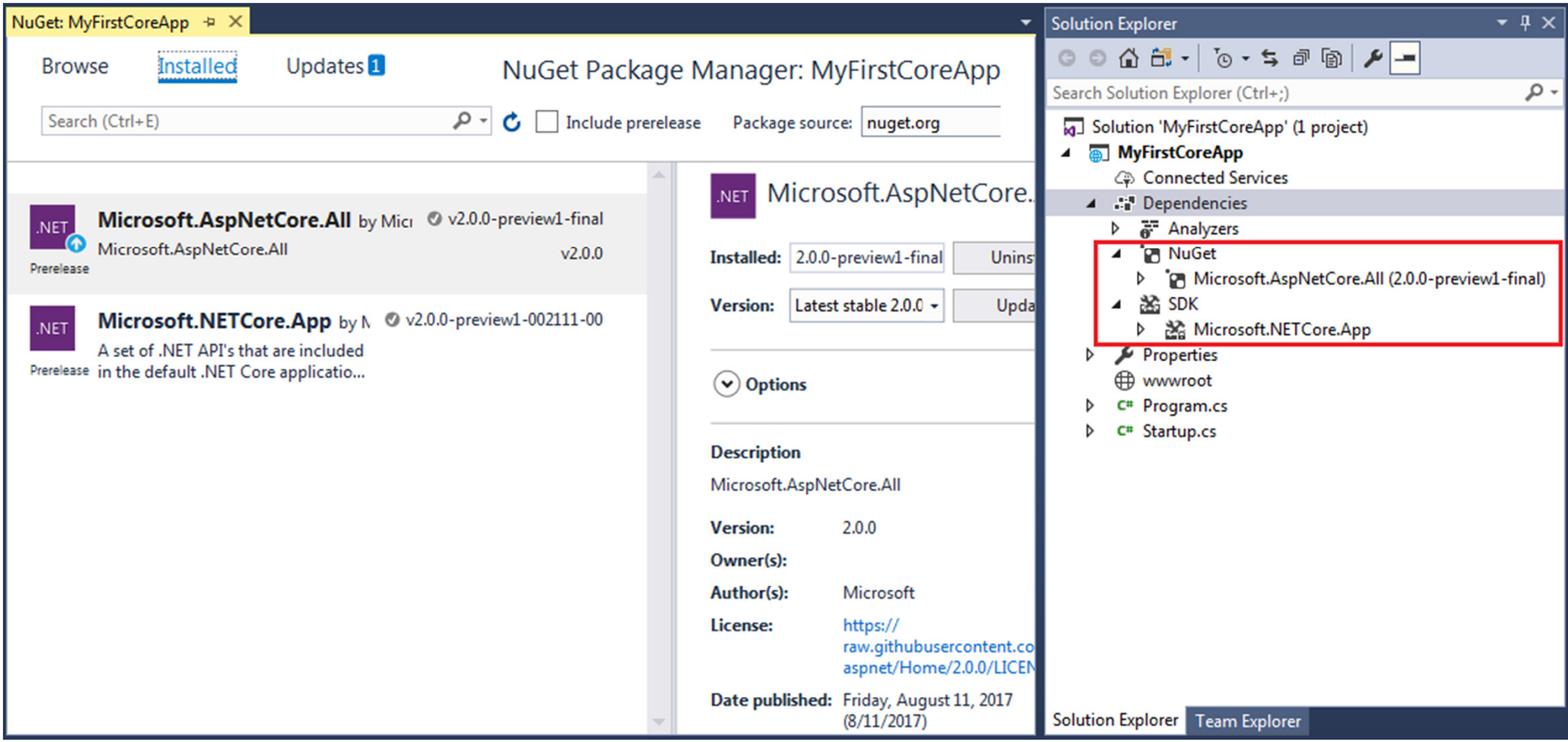


There is a SecretManager tool, provided by the .NET Core SDK (Microsoft.Extensions.SecretManager.Tools), which you can access with the dotnet CLI.

- %APPDATA%\Microsoft\UserSecrets\<user_secrets_id>\secrets.json

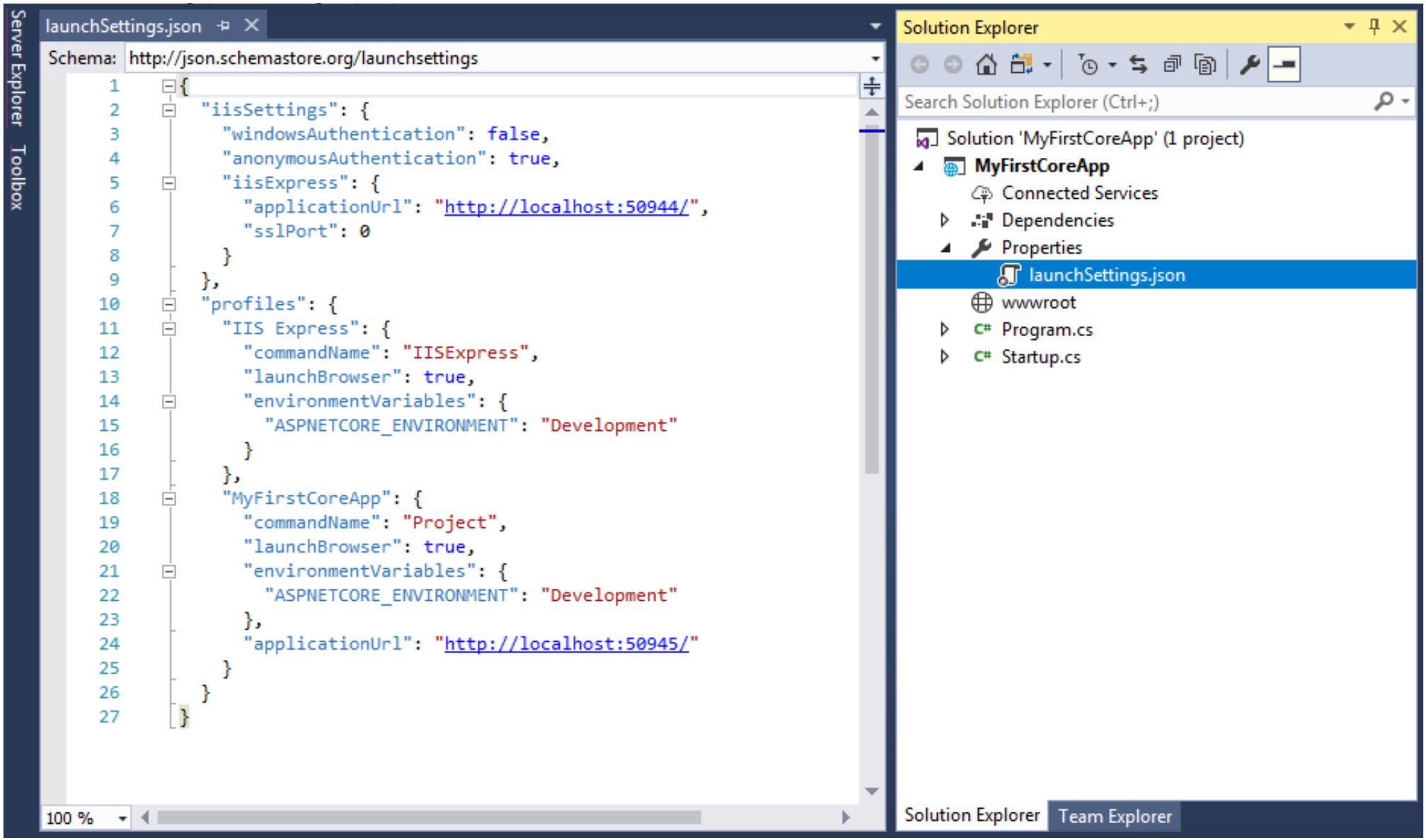
Dependencies

- Dependencies in the ASP.NET Core 2.0 project contains all the installed server-side NuGet packages as well as client-side frameworks such as jQuery, AngularJS, Bootstrap etc. These client-side dependencies are managed using Bower in Visual Studio.



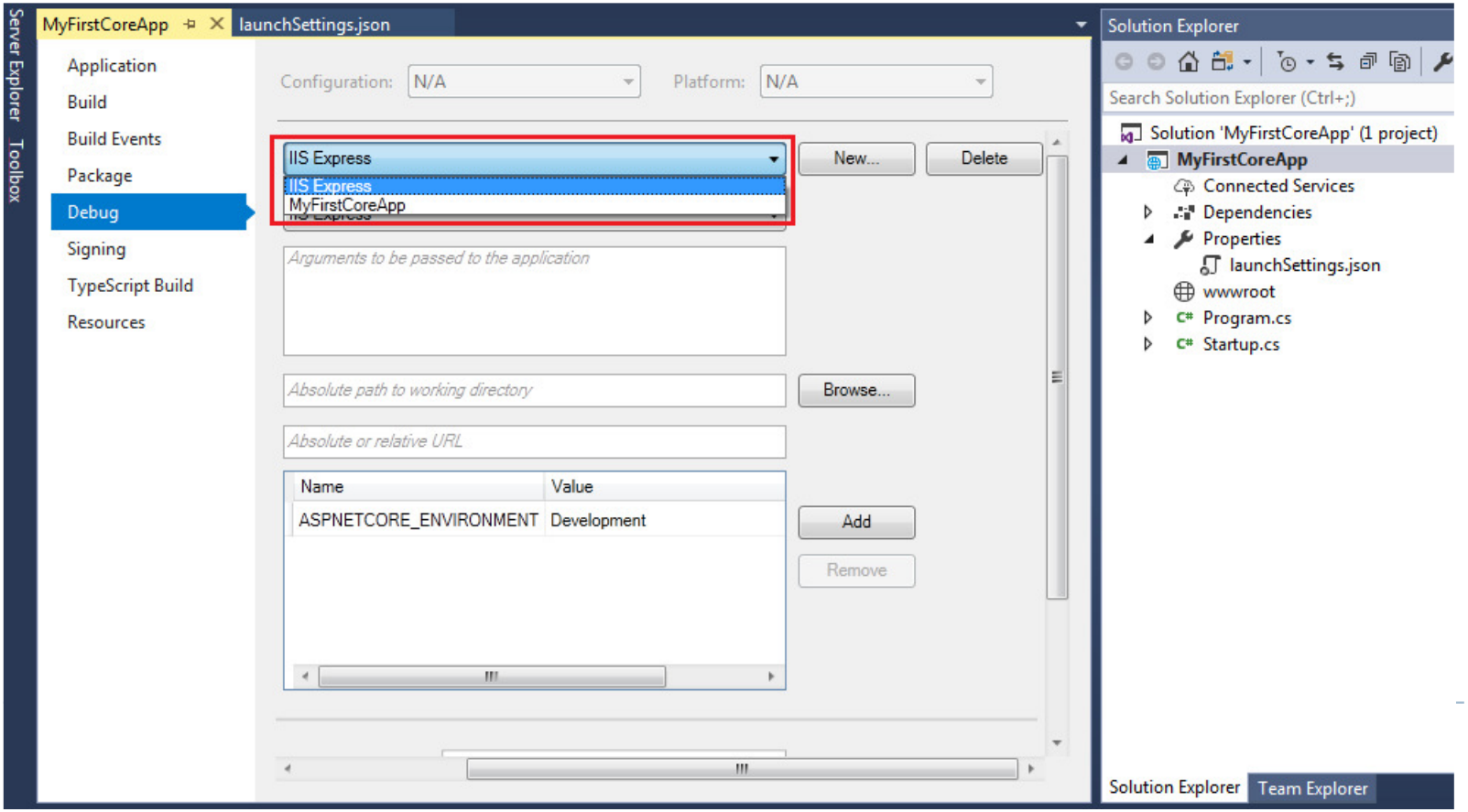
Properties

- launchSettings.json file which includes Visual Studio



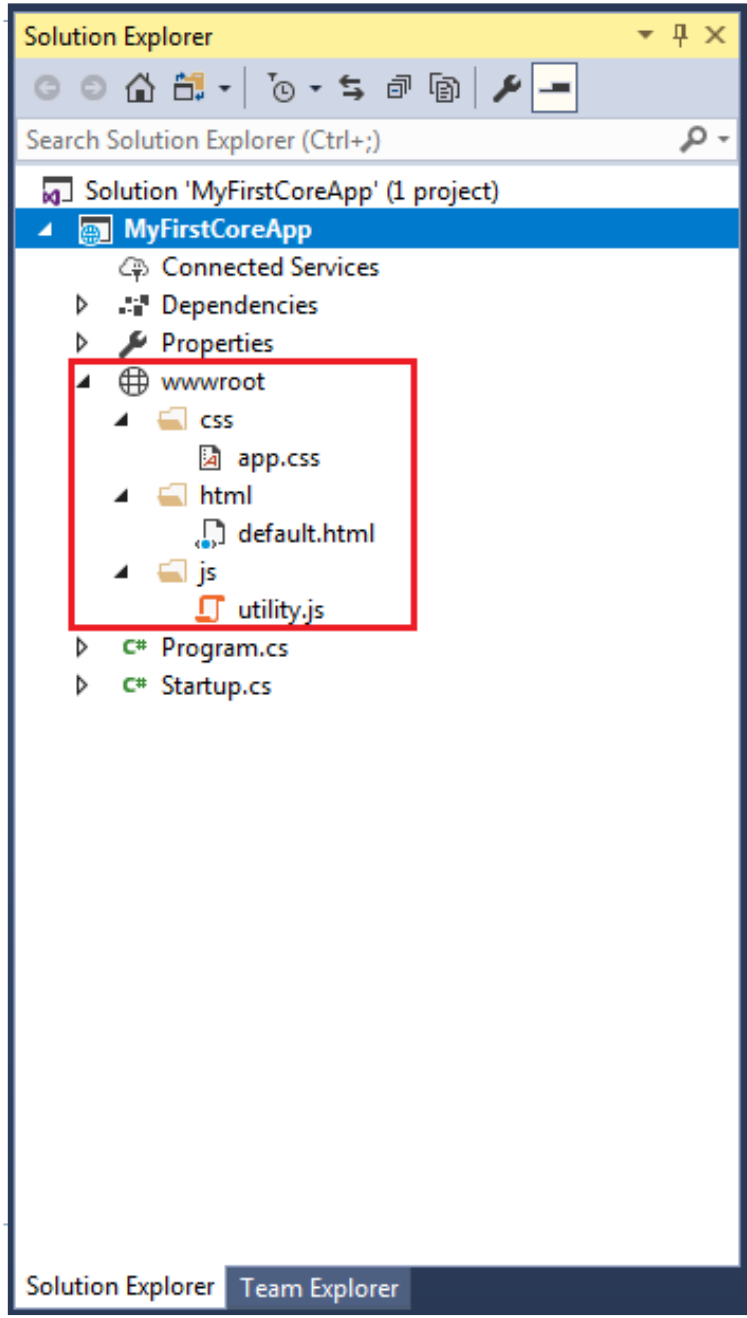
Properties

- Editing Project Properties
  - Right click on the project -> select Properties -> click Debug tab.
  - In the debug tab, select a profile which you want to edit as shown above. You may change environment variables, url etc.



ASP.NET Core - wwwroot Folder

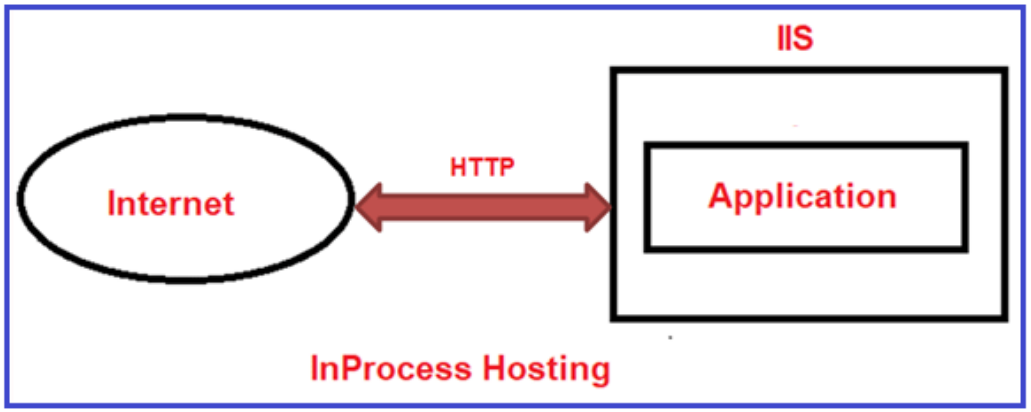
- Treated as a web root folder.
- Static files can be stored in any folder under the web root
- ASP.NET Core supports only those files that are in the web root - wwwroot folder can be served over an http request. All other files are blocked and cannot be served by default.
- For example, we can access above site.css file in the css folder by *http://localhost:<port>/css/app.css*.
- You will need to include a middleware for serving static files in the Configure method of Startup.cs.
- You can rename wwwroot folder to any other name as per your choice and set it as a web root while preparing hosting environment in the program.cs.

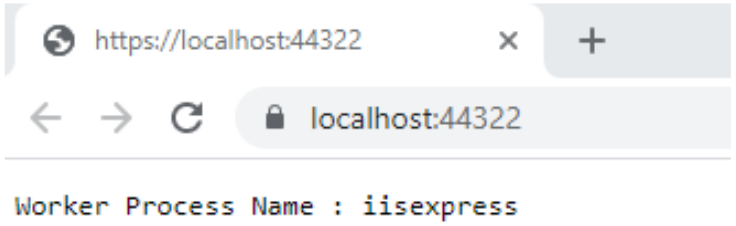


Hosting Model: InProcess

```
<Project Sdk="Microsoft.NET.Sdk.Web">
        <PropertyGroup>
                <TargetFramework>netcoreapp3.1</TargetFramework>
                <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
        </PropertyGroup>
</Project>
```

In ASP.NET Core, with InProcess Hosting Model our application is going to be hosted in the IIS worker process. The most important point that you need to remember is we have only one web server i.e. IIS Server in case of InProcess hosting which is going to host our application as shown in the below image.






```
await context.Response.WriteAsync("Worker Process Name : " +
System.Diagnostics.Process.GetCurrentProcess().ProcessName);
```

When we use the InProcess Hosting model, then the application is hosted inside the IIS worker process is w3wp.exe, and iisexpress.exe in the case of IIS Express. That means the Kestrel Web Server is not used with the InProcess hosting model.

IIS settings modify launchsettings

```
"IIS": {
      "commandName": "IIS",
      "launchBrowser": true,
      "launchUrl": "http://localhost/WebApplication8",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "ancmHostingModel": "OutOfProcess"
    }
```

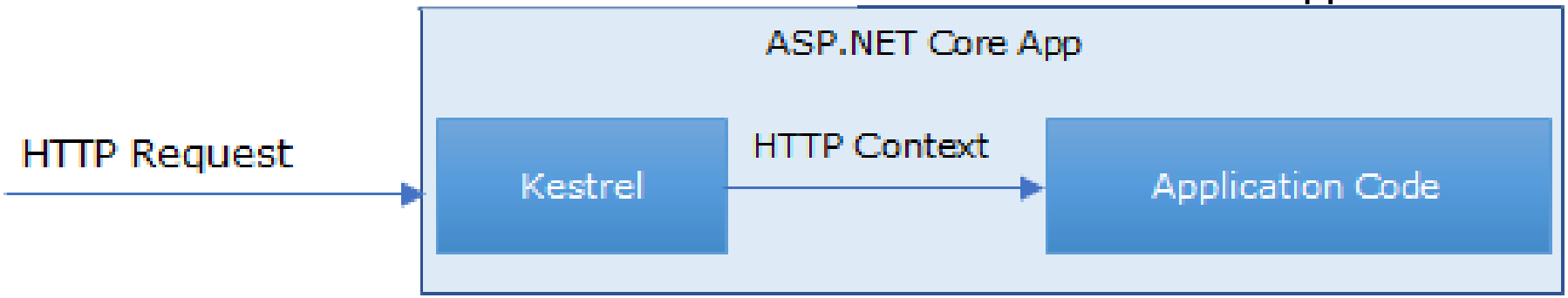
```
"IIS": {
    "commandName": "IIS",
    "launchBrowser": true,
    "launchUrl": "http://localhost/WebApplication8",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    },
    "ancmHostingModel": "InProcess"
  }
```

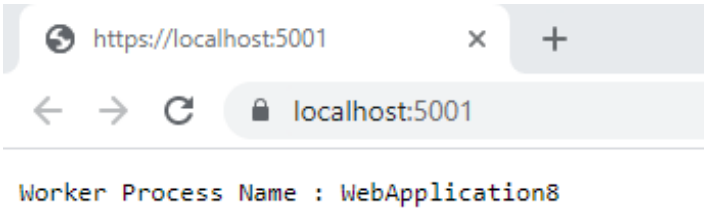
Hosting Model: Out of Process

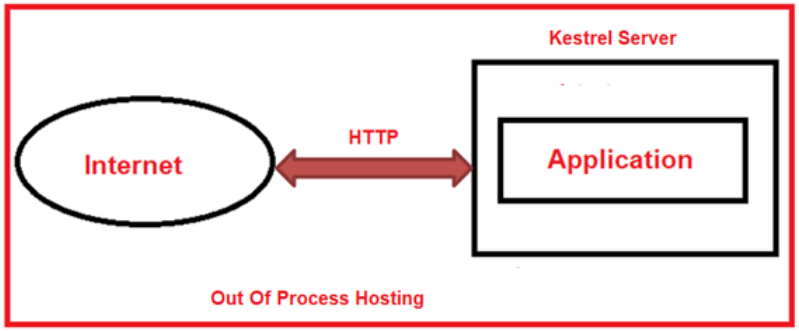
Kestrel as an Edge Server

ASP.NET Core is a cross-platform framework.

Kestrel is the cross-platform web server for the ASP.NET Core application.





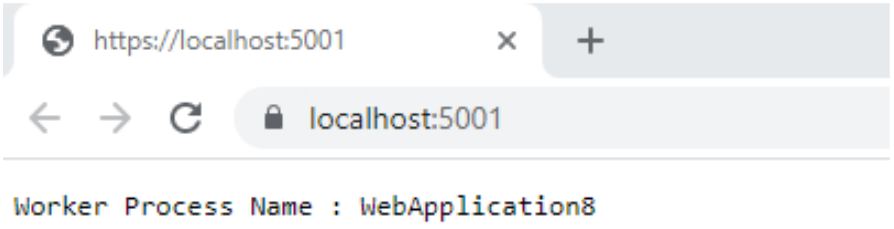


IIS/IIS Express is there But not in use

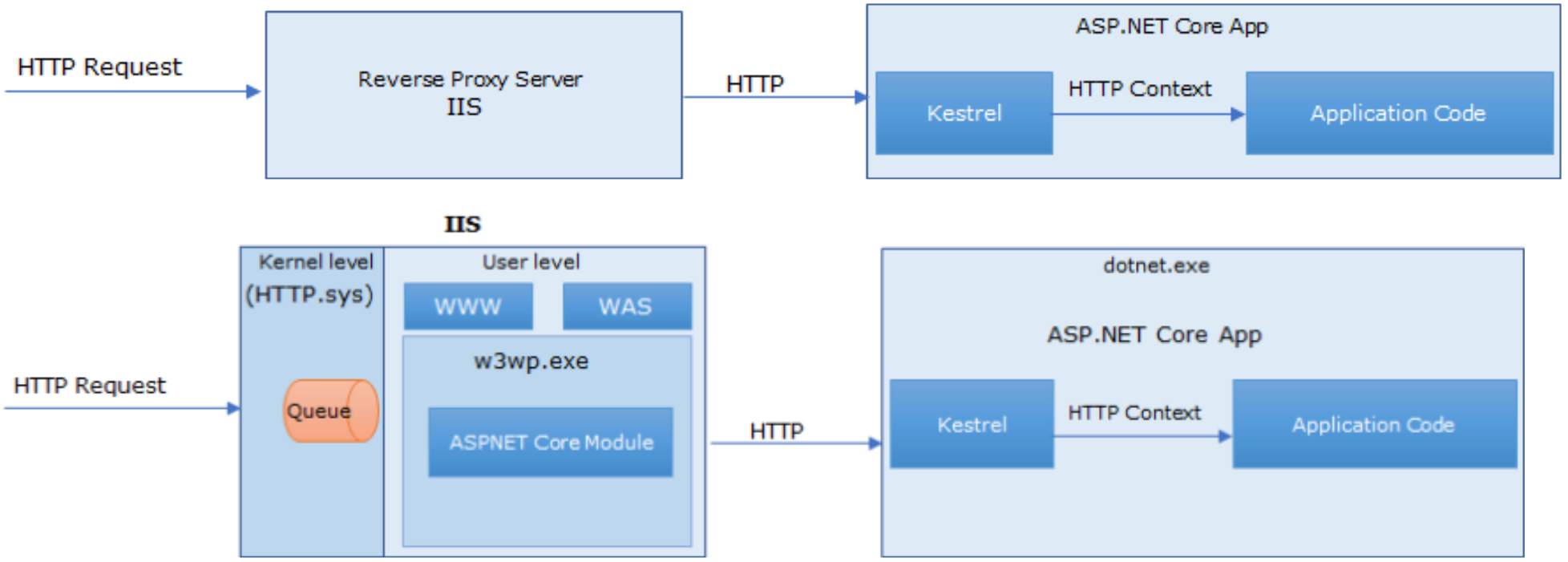
In the case of the ASP.NET Core OutOfProcess Hosting Model, there are two web servers.

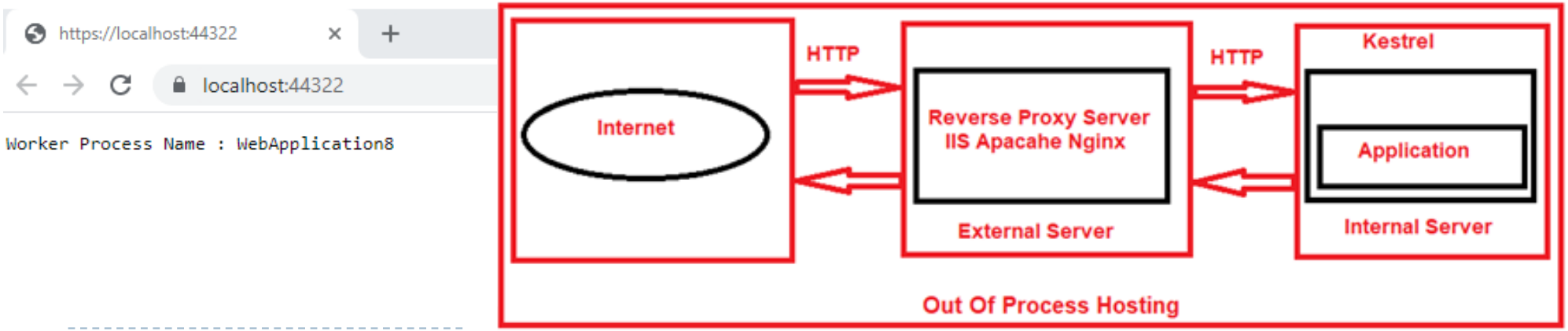
- An internal webserver which is the Kestrel web Server
- And an external web server which can be IIS, Apache, and Nginx.

- We can use the Kestrel Web Server as the internet-facing web server which will directly process the incoming HTTP requests.
- In this scenario, only the Kestrel Server is used and the other one i.e. external web server is not going to be used.
- So, when we run the application using the **.NET core CLI then Kestrel is the only web server that is going to be used to handle and process the incoming HTTP request** as shown in the below image.

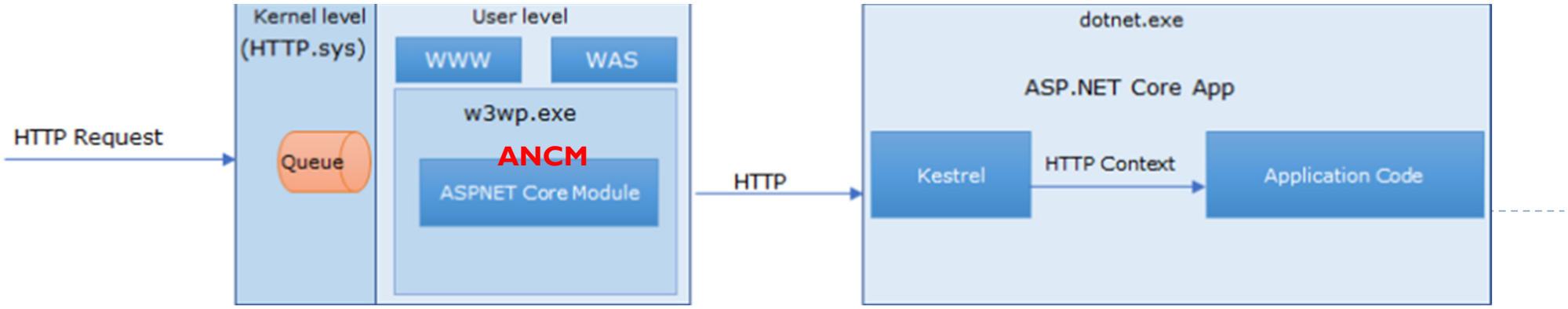


Using IIS as a Reverse Proxy:





https://codewala.net/2019/01/29/hosting-asp-net-core-applications-in-iis/



1. The request is received by the *HTTP.sys* from the network.
2. If response is cached at *HTTP.sys* then it is sent back from there else gets a place the corresponding Application Pool's queue.
3. When a thread is available in the thread pool, it picks up the request and start processing it.
4. The request goes through IIS processing pipeline. As mentioned earlier the request goes through few native IIS modules and once it reaches to *ANCM*, it forwards the request to *Kestrel* (under dotnet.exe).
5. ANCM has a responsibility to manage the process as well. If (re)starts the process (if not running or crashed) and IIS integration middleware configure the server to listen the request on port defined in environment variable. It only accepts the requests which originates from ANCM.
   Note -Please do note that in ASP.NET Webforms/MVC the application is hosted under the worker process w3wp.exe which is managed by Windows Activation Service (WAS) which was part of IIS.
6. Once the request is received by Kestrel, it creates the *HTTPContext* object and request is handed over to ASP.NET Core middleware pipeline.
7. The request is passed to routing middleware which invokes the right controller and action method (model binding, various filters almost similar way as earlier versions).
8. Finally, the response is returned from the action and passed to kestrel via Middlewares and later sent back to client via IIS.

ASP.NET Core - Program.cs 1.x

Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;

namespace MyFirstCoreApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();

            host.Run();
        }
    }
}
```

Kestrel as an internal web server.

Root folder where the content files are located such as MVC view files, CSS, images etc.

IIS as the external web server

ASP.NET Core - Program.cs 2.0

Main method is the entry point for that console application execution.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;

namespace MyFirstCoreApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}
```

The WebHost is a static class which can be used for creating an instance of IWebHost and IWebHostBuilder with pre-configured defaults.

The CreateDefaultBuilder() method creates a new instance of WebHostBuilder with pre-configured defaults.

Internally, it configures Kestrel, IISIntegration and other configurations.

```
public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

public static IWebHost BuildWebHost(string[] args)
{
    return WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();
}
```

ASP.NET Core Web Application initially starts as a Console Application and the Main() method is the entry point to the application. So, when we execute the ASP.NET Core Web application, first it looks for the Main() method and this is the method from where the execution starts. The Main() method then configures ASP.NET Core and starts it. At this point, the application becomes an ASP.NET Core web application.

CreateDefaultBuilder()

```
public static IWebHostBuilder CreateDefaultBuilder(string[] args)
{
    var builder = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
        {
            var env = hostingContext.HostingEnvironment;

            config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                  .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true, reloadOn(

            if (env.IsDevelopment())
            {
                var appAssembly = Assembly.Load(new AssemblyName(env.ApplicationName));
                if (appAssembly != null)
                {
                    config.AddUserSecrets(appAssembly, optional: true);
                }
            }

            config.AddEnvironmentVariables();

            if (args != null)
            {
                config.AddCommandLine(args);
            }
        })
        .ConfigureLogging((hostingContext, logging) =>
        {
            logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
            logging.AddConsole();
            logging.AddDebug();
        })
        .UseIISIntegration()
        .UseDefaultServiceProvider((context, options) =>
        {
            options.ValidateScopes = context.HostingEnvironment.IsDevelopment();
        });

    return builder;
}
```

Kestrel as an internal web server.

CreateDefaultBuilder method creates an instance of WebHostBuilder and sets up Kestrel, content root directory, IIS integration which is same as ASP.NET Core 1.x Main() method.

Root folder where the content files are located such as MVC view files, CSS, images etc.

It also calls ConfigureAppConfiguration() to load configurations from appsettings.json files, environment variables and user secrets.

The ConfigureLogging() method setup logging to console and debug window.

IIS as the external web server

ASP.NET Core - Startup Class

- Must include a Configure method and can optionally include ConfigureService method.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddMvc();
}
```

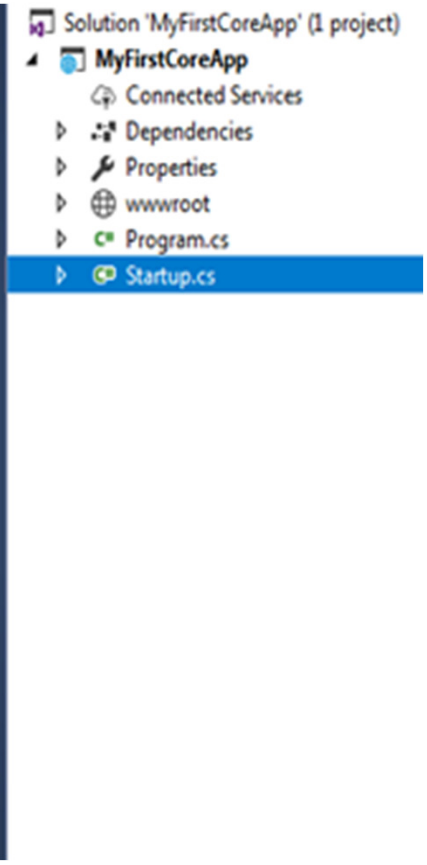
```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?Li
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    }
}
```

Dependency Injection

Register Dependent Types (Services) with IoC Container here

Configure HTTP request pipeline (Middleware) here



ConfigureServices()

- The ConfigureServices method is a place where we can register our dependent classes with the built-in IoC container.
  - Inversion of control (IoC) is a programming principle. IoC inverts the flow of control as compared to traditional control flow.
- After registering dependent class, it can be used anywhere in the application.
- You just need to include it in the parameter of the constructor of a class where you want to use it. The IoC container will inject it automatically.
- ASP.NET Core refers dependent class as a Service. So, whenever you read "Service" then understand it as a class which is going to be used in some other class.
- ConfigureServices method includes IServiceCollection parameter to register services to the IoC container.

At run time, the ConfigureServices method is called before the Configure method. This is so that you can register your custom service with the IoC container which you may use in the Configure method.

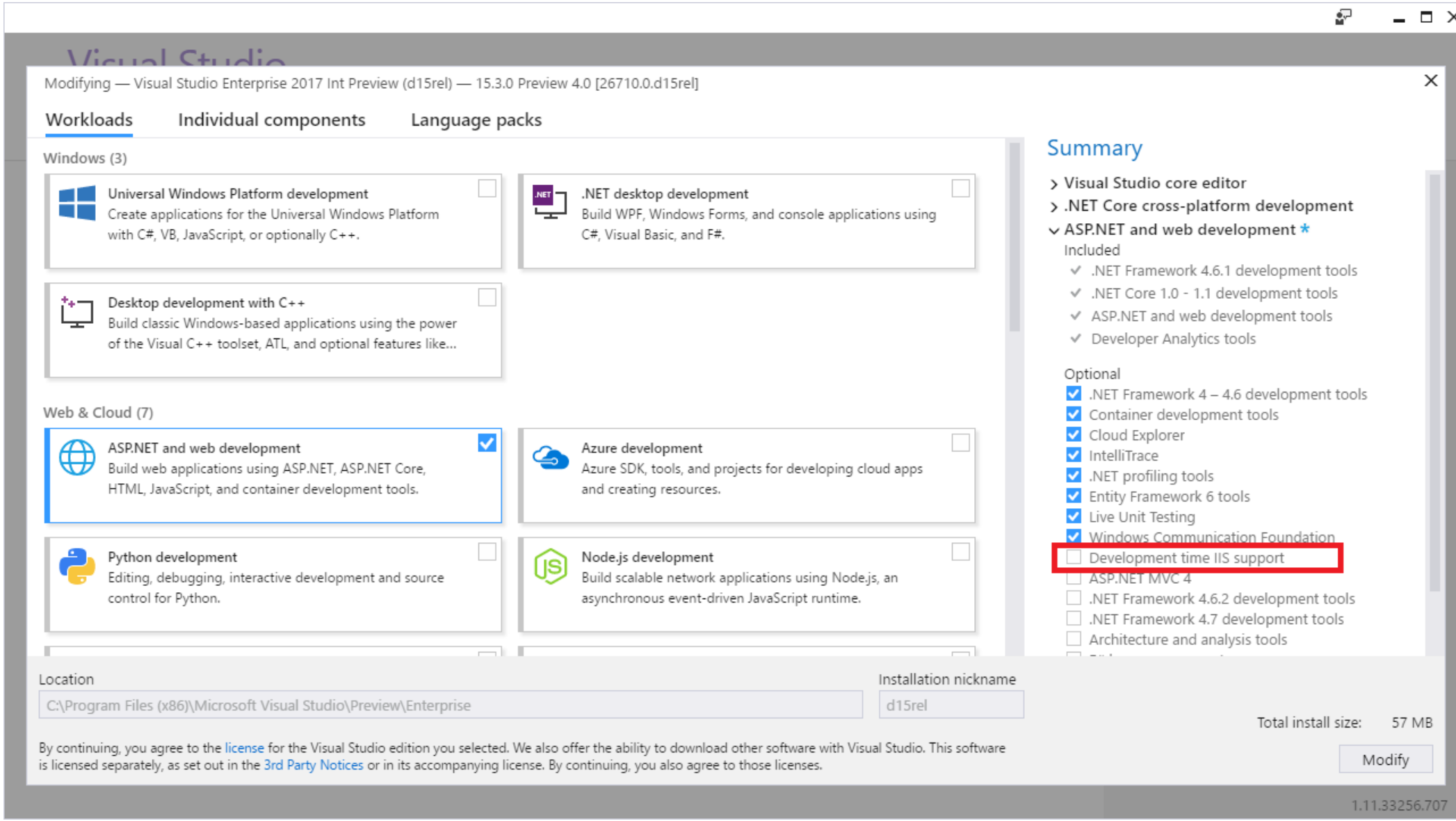
Configure( )

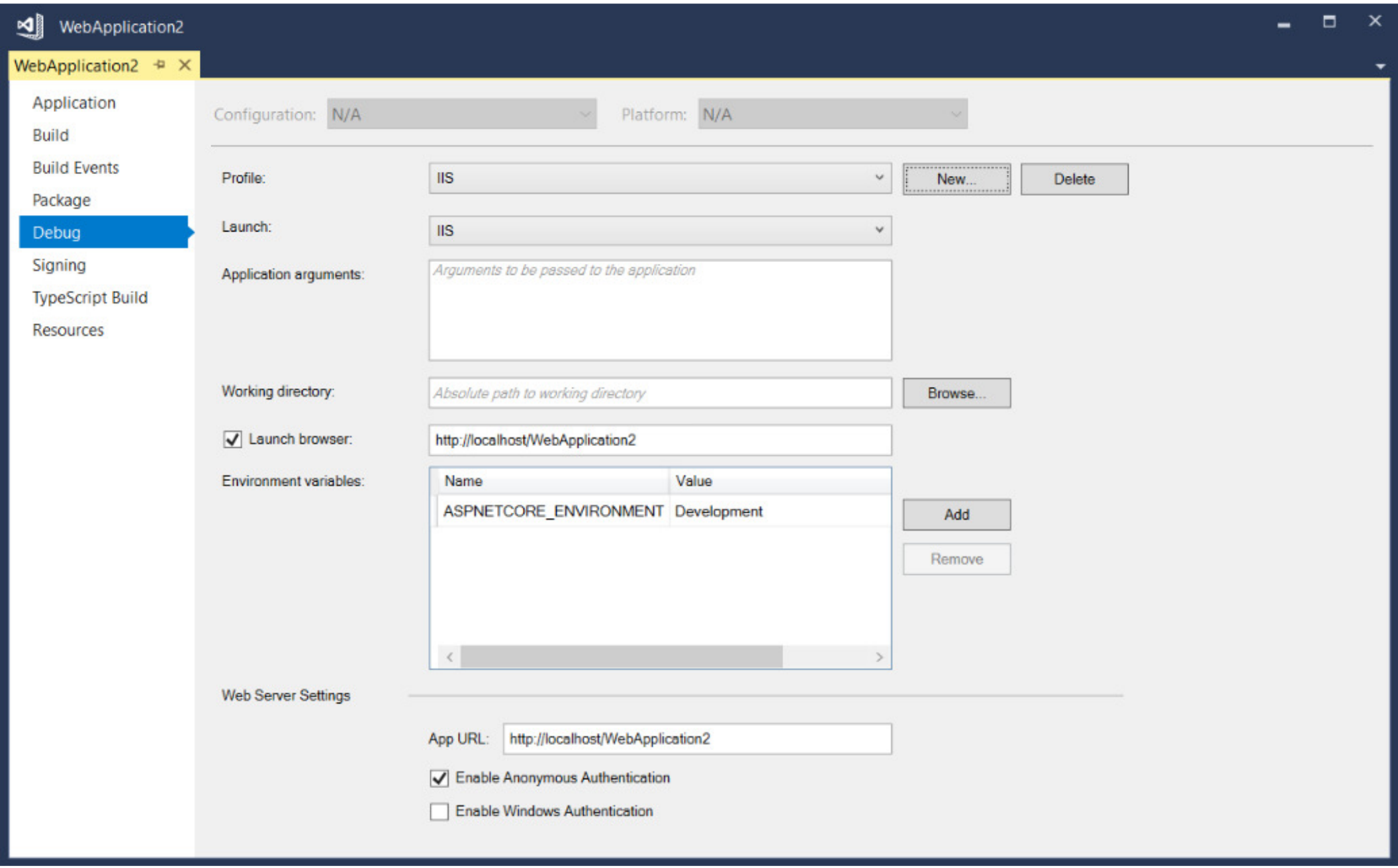
- You can configure application request pipeline using IApplicationBuilder instance that is provided by the built-in IoC container.
- ASP.NET Core introduced the middleware components to define a request pipeline,
  - which will be executed on every request.
  - You include only those middleware components which are required by your application and thus increase the performance of your application.

https://www.tutorialsteacher.com/core/aspnet-core-startup

Development time IIS support

- Once you've installed IIS, you can launch the Visual Studio installer to modify your existing Visual Studio installation. In the installer select the *Development time IIS support* component which is listed as optional component under the *ASP.NET and web development* workload. This will install the ASP.NET Core Module which is a native IIS module required to run ASP.NET Core applications on IIS.


Visual Studio
Modifying — Visual Studio Enterprise 2017 Int Preview (d15rel) — 15.3.0 Preview 4.0 [26710.0.d15rel]
Workloads
Individual components
Language packs
Windows (3)
Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.
.NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.
Desktop development with C++
Build classic Windows-based applications using the power of the Visual C++ toolset, ATL, and optional features like...
Web & Cloud (7)
ASP.NET and web development
Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and container development tools.
Azure development
Azure SDK, tools, and projects for developing cloud apps and creating resources.
Python development
Editing, debugging, interactive development and source control for Python.
Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.
Summary
Visual Studio core editor
.NET Core cross-platform development
ASP.NET and web development *
Included
.NET Framework 4.6.1 development tools
.NET Core 1.0 - 1.1 development tools
ASP.NET and web development tools
Developer Analytics tools
Optional
.NET Framework 4 – 4.6 development tools
Container development tools
Cloud Explorer
IntelliTrace
.NET profiling tools
Entity Framework 6 tools
Live Unit Testing
Windows Communication Foundation
Development time IIS support
ASP.NET MVC 4
.NET Framework 4.6.2 development tools
.NET Framework 4.7 development tools
Architecture and analysis tools
Location
C:\Program Files (x86)\Microsoft Visual Studio\Preview\Enterprise
Installation nickname
d15rel
Total install size: 57 MB
By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.
Modify
1.11.33256.707


WebApplication2
WebApplication2
Application
Build
Build Events
Package
Debug
Signing
TypeScript Build
Resources
Configuration: N/A
Platform: N/A
Profile: IIS
New...
Delete
Launch: IIS
Application arguments:
Arguments to be passed to the application
Working directory:
Absolute path to working directory
Browse...
Launch browser:
http://localhost/WebApplication2
Environment variables:
Name
Value
ASPNETCORE_ENVIRONMENT
Development
Add
Remove
Web Server Settings
App URL:
http://localhost/WebApplication2
Enable Anonymous Authentication
Enable Windows Authentication

EndPoint Routing(new in 3.1)

- EndPoint Routing is the new way to implement the Routing in ASP.NET Core.
- It splits up the old routing middleware into two separate middleware's and also decouples the MVC from the Routing Middleware.
- Learn what is Endpoint is and how to register these routing middleware's using the UseRouting & UseEndpoints methods in the Configure method of the startup class.

What is Endpoint Routing in ASP.NET Core

- An Endpoint is an object that contains everything that you need to execute the incoming Request. The Endpoint object contains the following information

- Metadata of the request.
- The delegate (Request handler) that ASP.NET core uses to process the request.
- We define the Endpoint at the application startup using the UseEndpoints method.

https://www.tektutorialshub.com/asp-net-core/asp-net-core-endpoint-routing/

Cloud computing service categories

Lab1: Introduction

Prof. Dr. Shamim Akhter

IUBAT, Bangladesh

Visual Studio 2019

Open recent

Today

WebApplication4.sln C:\Users\Student\source\repos\WebApplication4	1/2/2020 11:40 AM
WebApplication4 C:\Users\Student\source\repos\WebApplication4	1/2/2020 11:27 AM
WebApplication6.sln C:\Users\Student\source\repos\WebApplication6	1/2/2020 11:27 AM
WebApplication5 C:\Users\Student\source\repos\WebApplication5	1/2/2020 11:23 AM
WebApplication2 C:\Users\Student\source\repos\WebApplication2	1/2/2020 11:20 AM
WebApplication1 C:\Users\Student\source\repos\WebApplication1	1/2/2020 11:18 AM

Get started

Clone or check out code
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

Create a new project

Recent project templates

- ASP.NET Core Web Application — C#
- Console App (.NET Core) — C#
- Windows Forms App (.NET Framework) — C#

All languages | All platforms | All project types

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

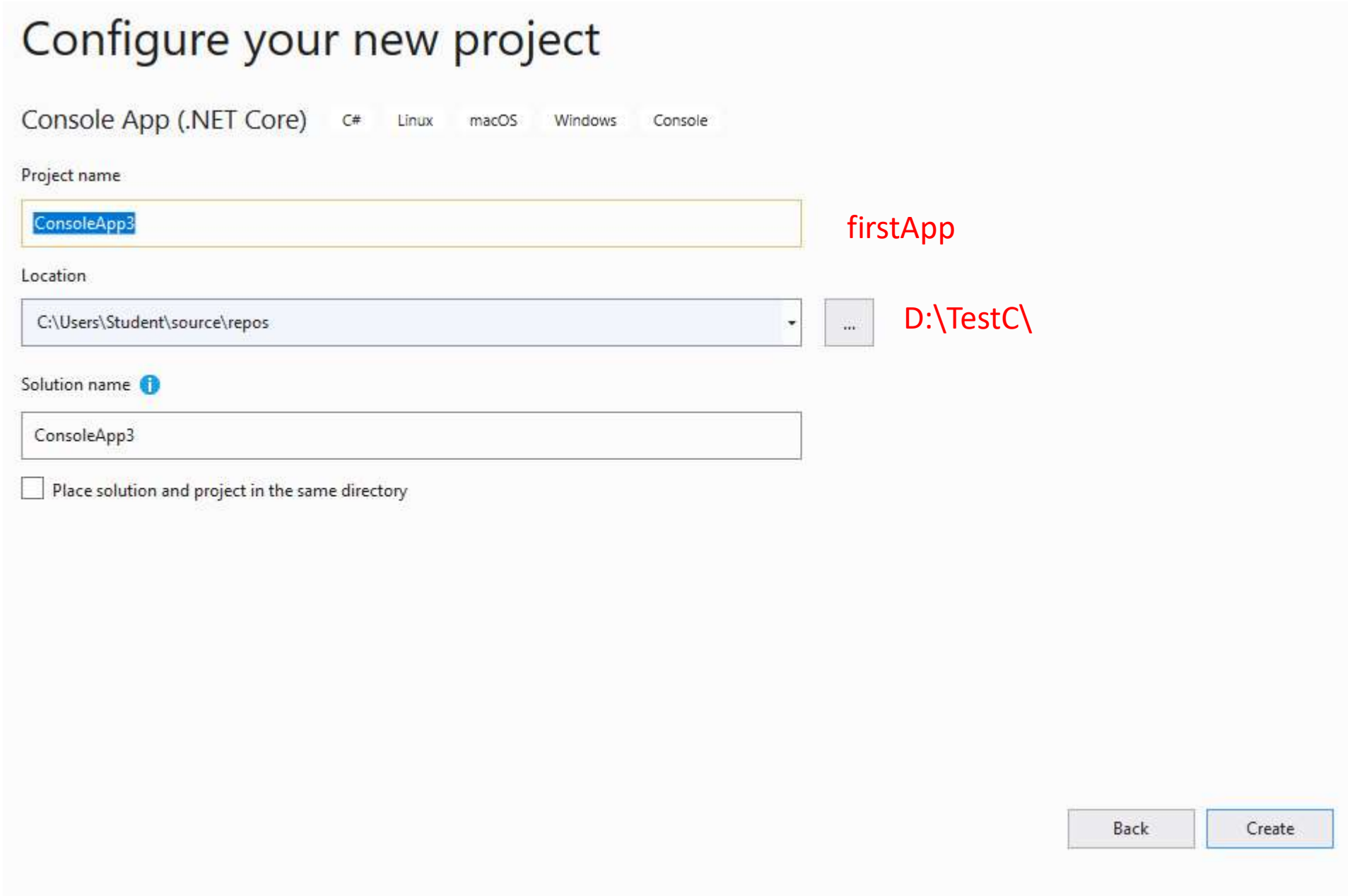
Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Windows Linux macOS Console

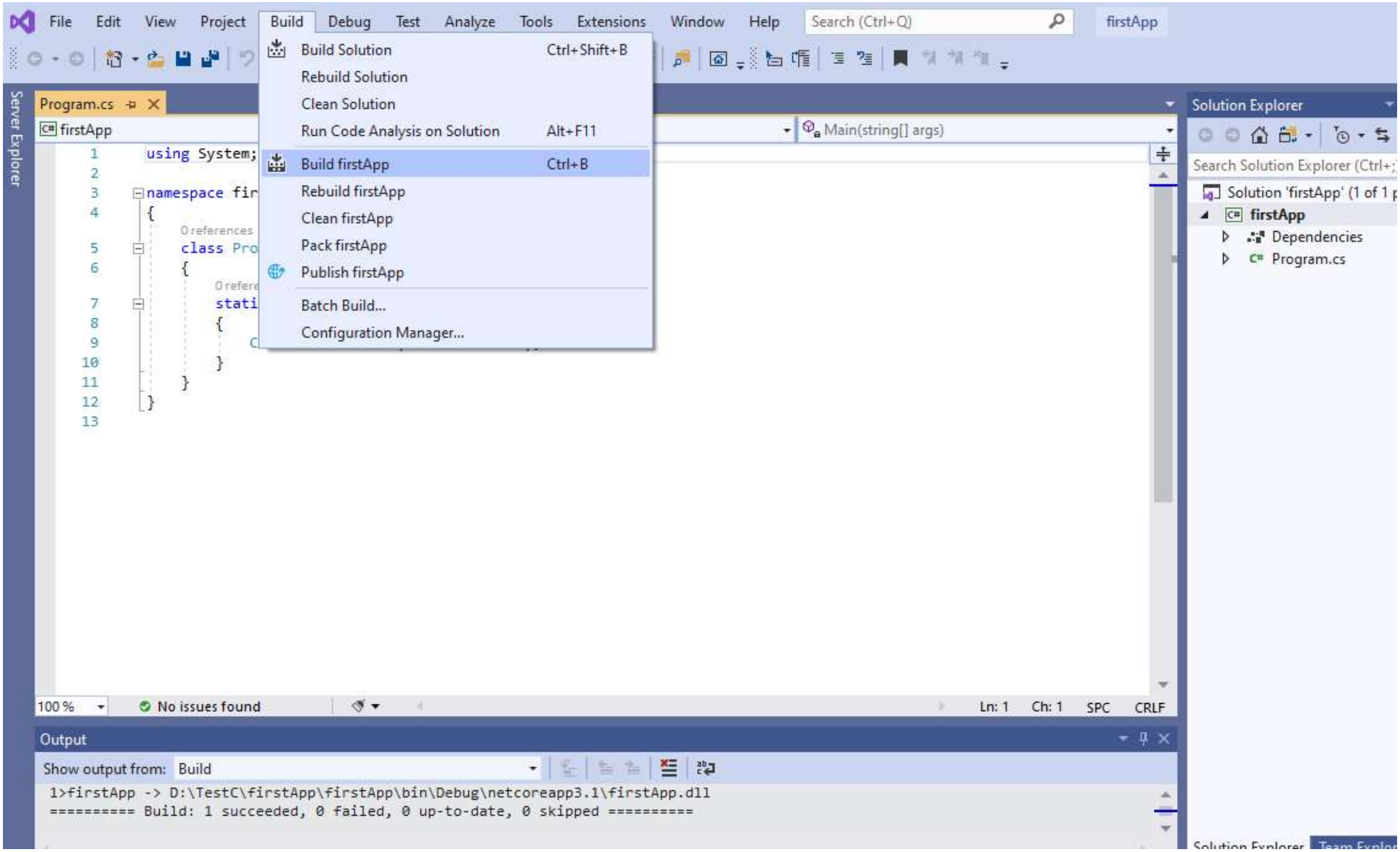
ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web

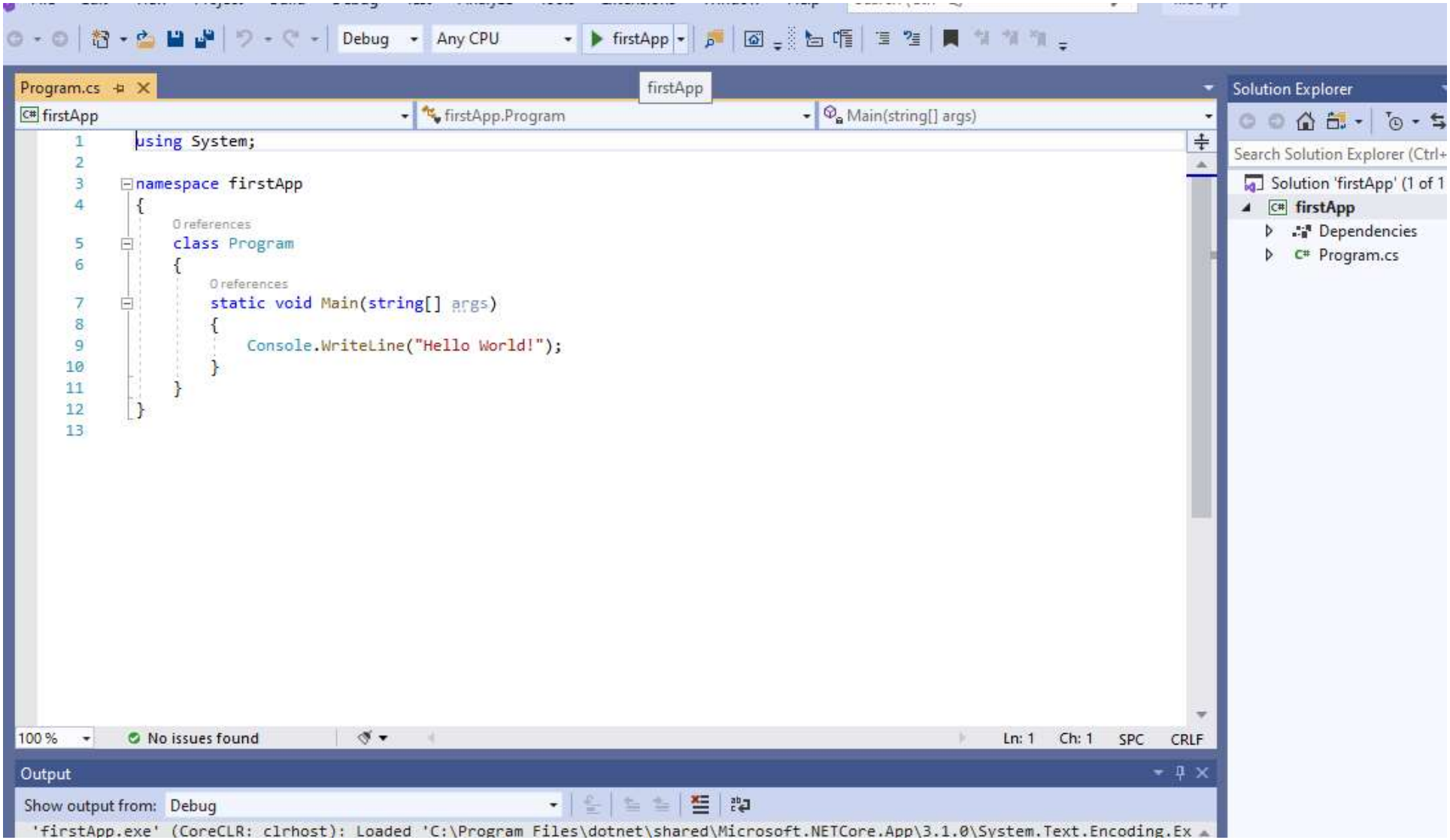
Blazor App
Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

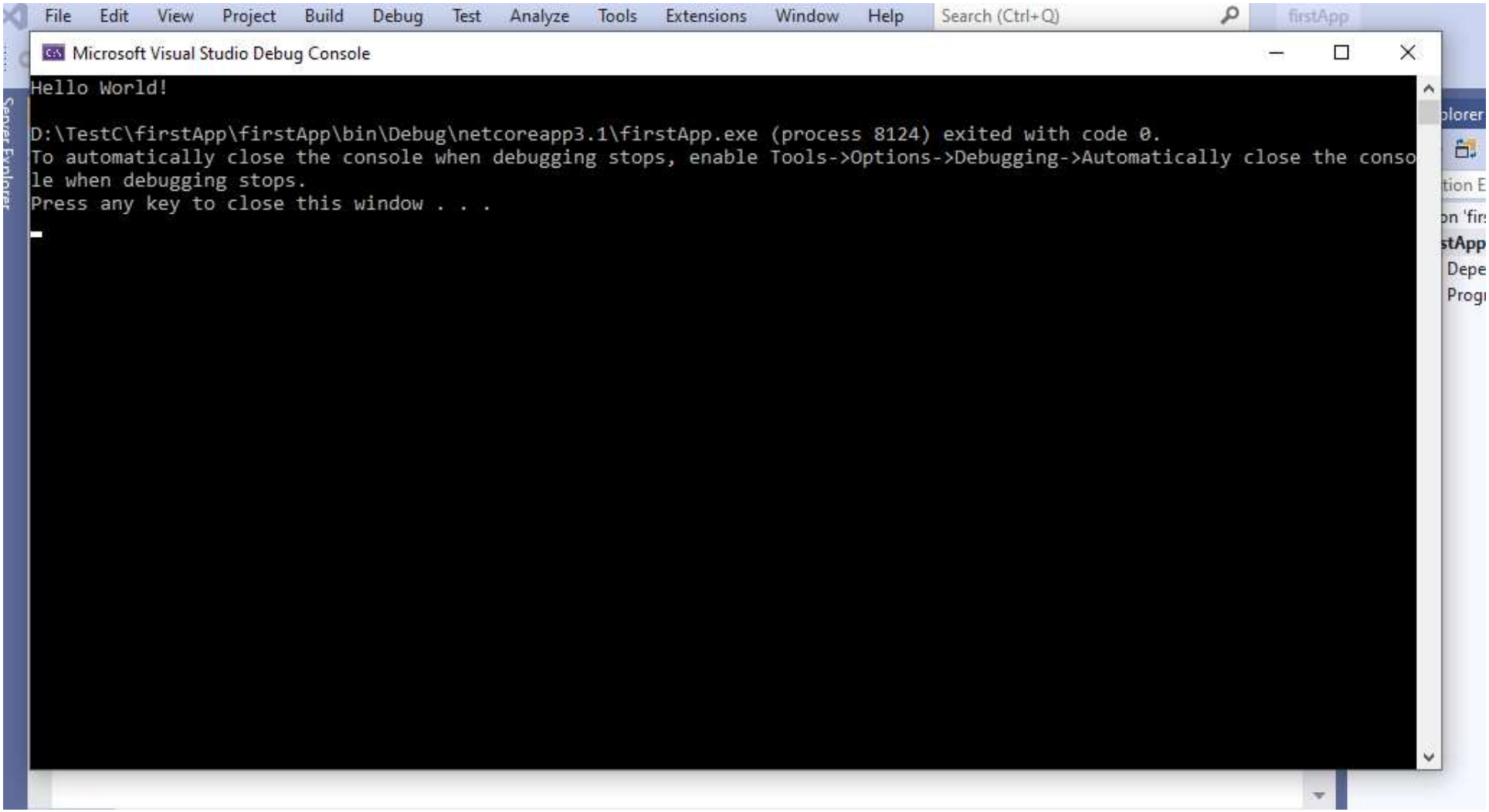
ASP.NET Web Application (.NET Framework)

Back Next


Configure your new project
Console App (.NET Core)
C#
Linux
macOS
Windows
Console
Project name
ConsoleApp3
firstApp
Location
C:\Users\Student\source\repos
...
D:\TestC\
Solution name
ConsoleApp3
Place solution and project in the same directory
Back
Create


File
Edit
View
Project
Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
Search (Ctrl+Q)
firstApp
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Run Code Analysis on Solution Alt+F11
Build firstApp Ctrl+B
Rebuild firstApp
Clean firstApp
Pack firstApp
Publish firstApp
Batch Build...
Configuration Manager...
Server Explorer
Program.cs
firstApp
Main(string[] args)
using System;
namespace fir
{
0 references
class Pro
{
0 refere
stati
{
}
}
}
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'firstApp' (1 of 1 p
firstApp
Dependencies
Program.cs
100 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF
Output
Show output from: Build
1>firstApp -> D:\TestC\firstApp\firstApp\bin\Debug\netcoreapp3.1\firstApp.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========


Debug
Any CPU
firstApp
Program.cs
firstApp
firstApp.Program
Main(string[] args)
using System;
namespace firstApp
{
0 references
class Program
{
0 references
static void Main(string[] args)
{
Console.WriteLine("Hello World!");
}
}
}
Solution Explorer
Search Solution Explorer (Ctrl+
Solution 'firstApp' (1 of 1
firstApp
Dependencies
Program.cs
100 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF
Output
Show output from:
Debug
'firstApp.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.0\System.Text.Encoding.Ex


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
firstApp
Microsoft Visual Studio Debug Console
Hello World!
D:\TestC\firstApp\firstApp\bin\Debug\netcoreapp3.1\firstApp.exe (process 8124) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
Server Explorer

Problem Solves -1

- Read an integer input from the console, and print reverse of that number in console.
- You can call a reverse() method from Main() and pass the integer values.
- Reverse() method will return the reverse value to Main().
- Main() will catch and print the value.

```
using System;

namespace ConsoleApp
{
   public class Program
   {
      public static int reverse(int N, int E)
      {
                int P=0;
                int M=(int)Math.Pow(10,E-1);
                while(N!=0)
                {
                          P+=(N%10)*M;
                          N=N/10;
                          M=M/10;
                }
        return P ;
      }
     }
}
```

```
public class program2
{
                public static void Main()
                {
                     Console.WriteLine("Hello World");
                      int i;
                      i=Convert.ToInt32(Console.ReadLine());
                Console.WriteLine(ConsoleApp.Program.reverse(i, 3));
                }
}
```

Problem Solves -2

- Take 10 inputs from console and print their average in console.
- You should use separate function to take input and find average.
- Main() will call those functions and print the average.

```
type [ ] array_name= new type[size];
int [] sample = new int [10];

int [ ] sample;
sample=new int [10];

int Size= sample.Length;
```

Problem Solves -3

- Write a program so that if user gives input ‘d’ or ‘D’; ‘k’ or ‘K’ and ‘s’ or ‘S’ the program will print Dhaka, Khulna and Sylhet respectively.
- You need to use switch statement.
- Dhaka, Khulna and Sylhet will be stored in string array.

Lab2: Class Members

Prof. Dr. Shamim Akhter

Problem-1

- **Implement** a class named Account that contains:
  - A private int data field named id for the account (default 0).
  - A private double data field named balance for the account (default 0.0).
  - A no-arg constructor that creates a default account with id=0 and balance=0.0
  - An argument constructor that creates an account with the specified id, initial balance
  - A method named setBalance() that sets a given amount to instance variable balance if the given amount is more than 100.
  - A method named getBalance() that returns the value of current balance.

Problem-2

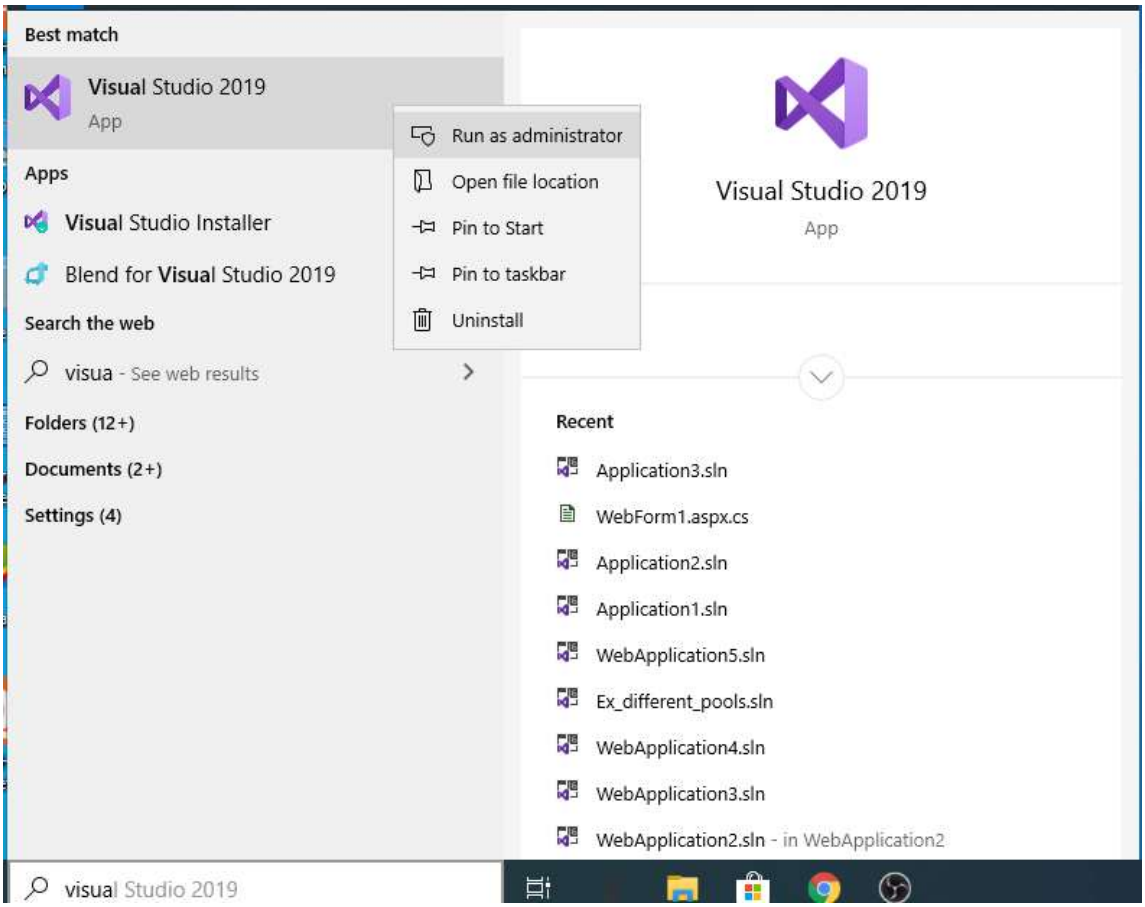
- Change Problem-1 so that we can add a properties for balance to call set and get automatically.
- The set accessor automatically receives a parameter called value and assign it to property
- Get accessor automatically returns the property value

Problem-3

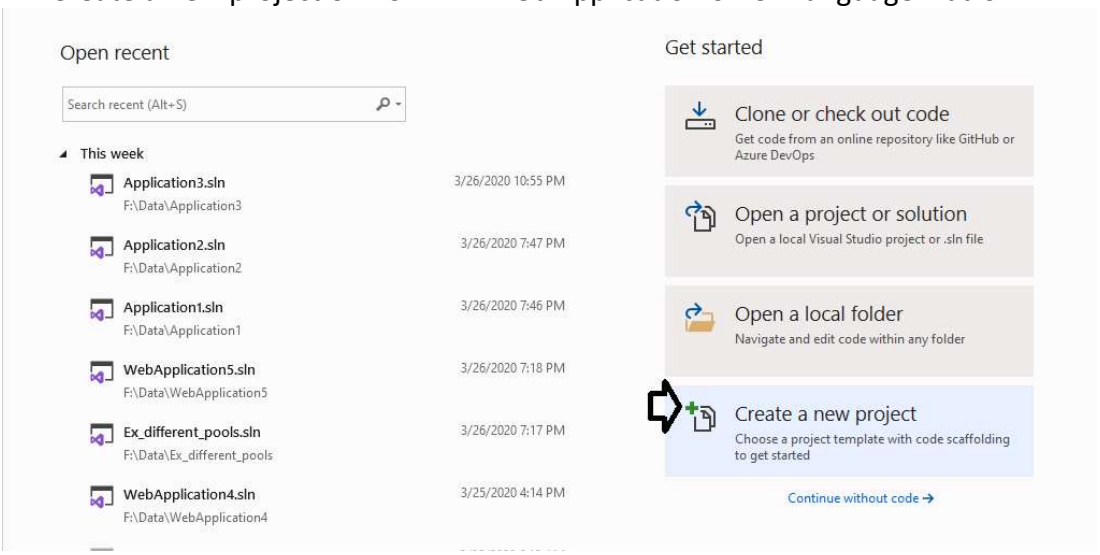
- **Implement** a class named Account that contains:
  - A private integer data field array named Balance. Which will hold the account balance of against a customer id.
  - An argument constructor that allocates the size of the Balance and given during Account instantiation.
  - An indexer with set and get accessor will be added in Account class.
    - set accessor will check the valid id (0 to size) and set the value corresponding customer id if and only if the id is valid.
    - get accessor will return the Balance value of a customer only if the customer id is valid, otherwise return -2 value.
  - Test will be the driver class will create an account object
  - Now use object indexed (not operator method) like array to create 10 integer elements
  - Using indexer to store and to retrieve elements values.

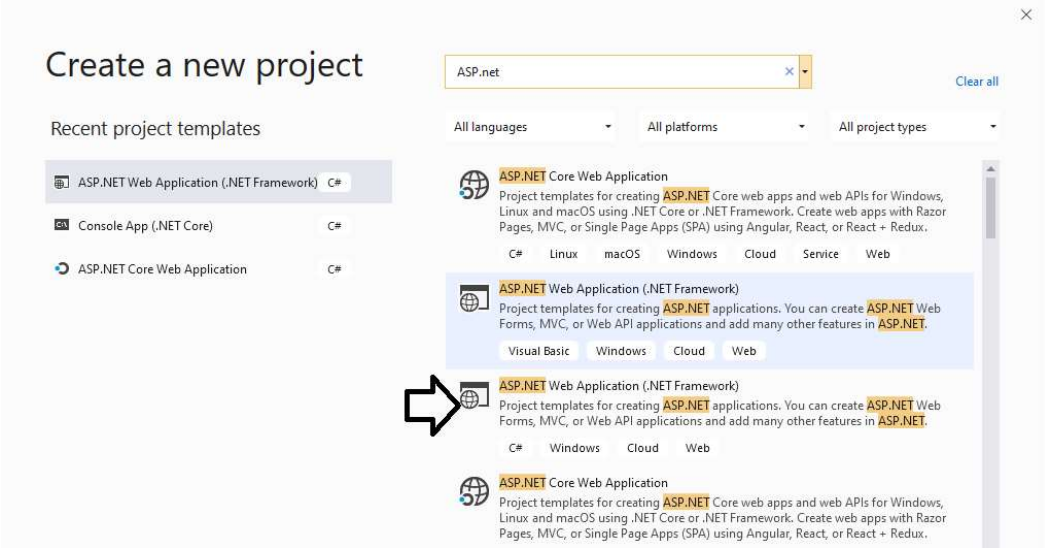
Running an ASP.NET Framework Program through Local IIS

1. At the first step, you need to run Visual Stdio 2019 with Administrator Mode

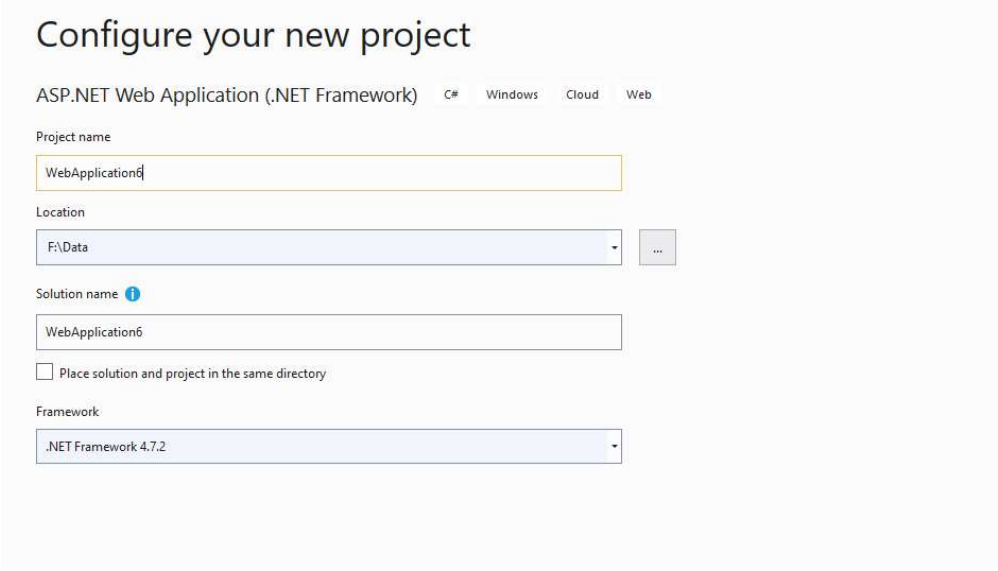


2. Create a new project on ASP.NET Web Application on C# Language Platform

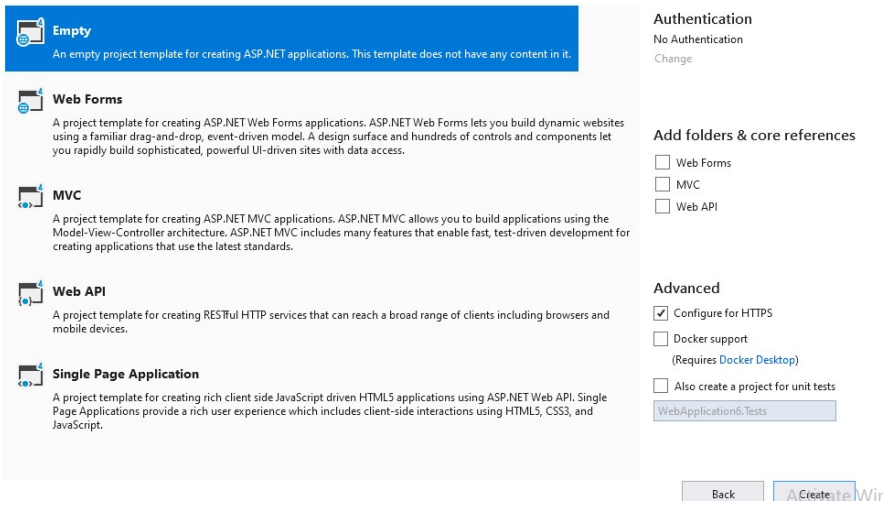




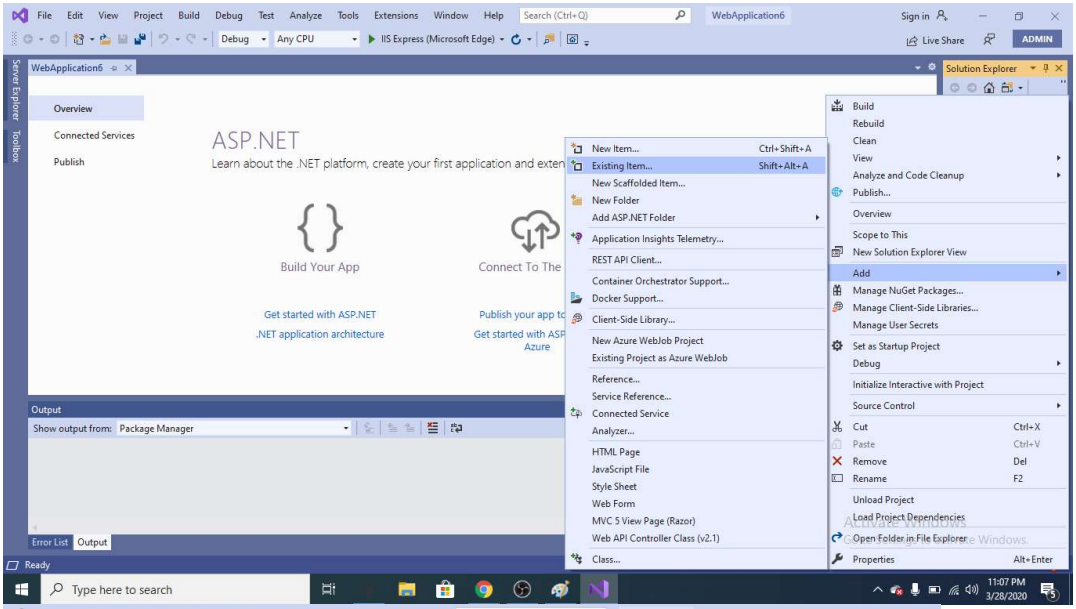
3. Give the name of the project

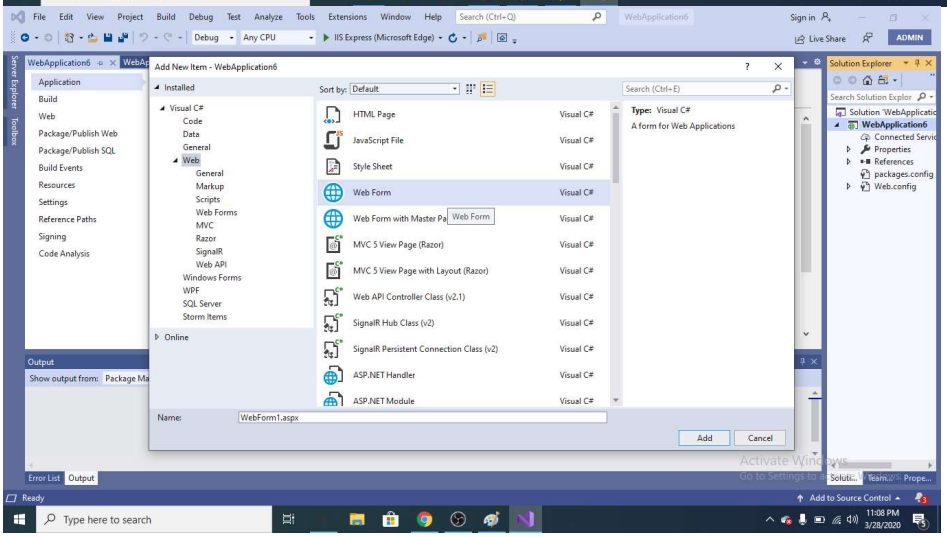


4. Choose empty templete

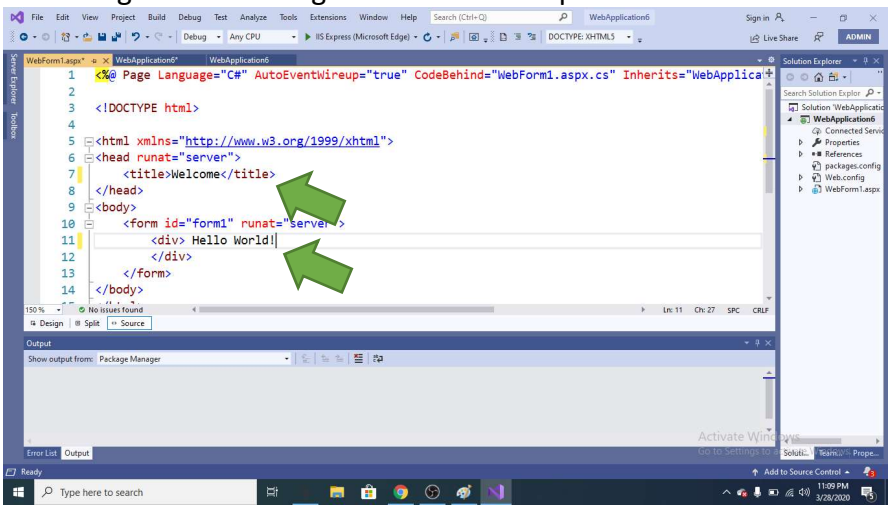


5. Add an existing item(WebForm1.aspx) into the project.

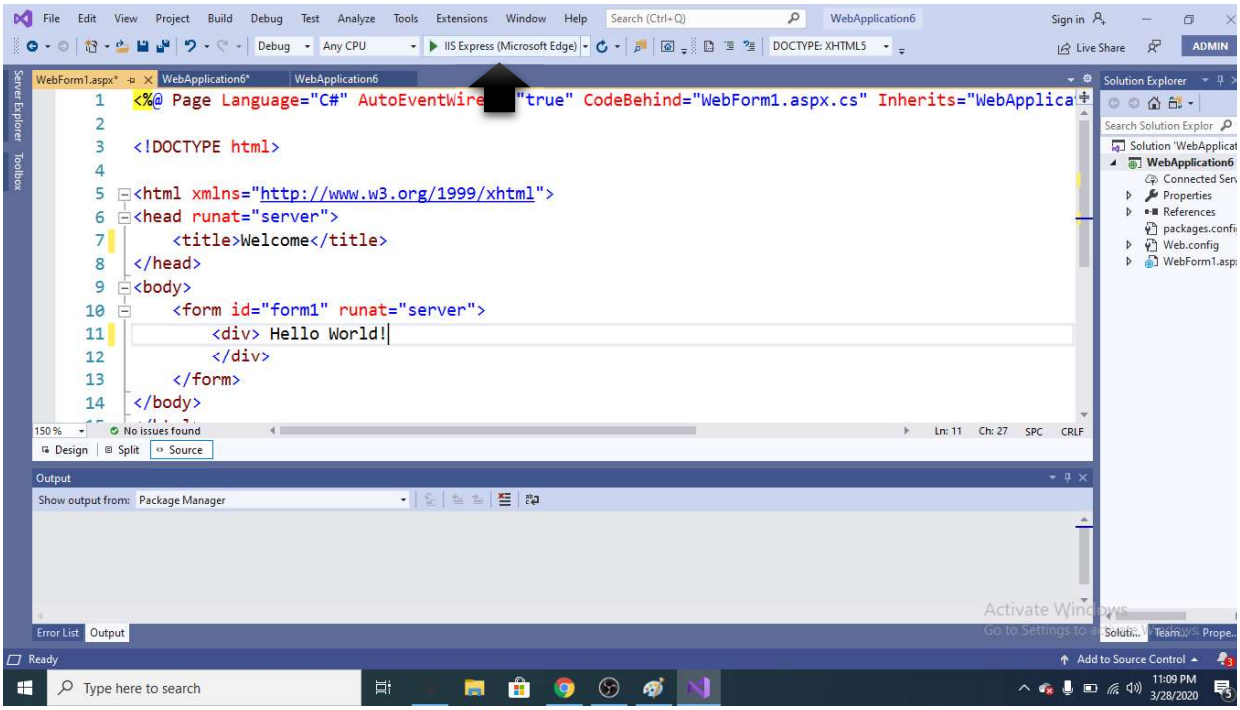




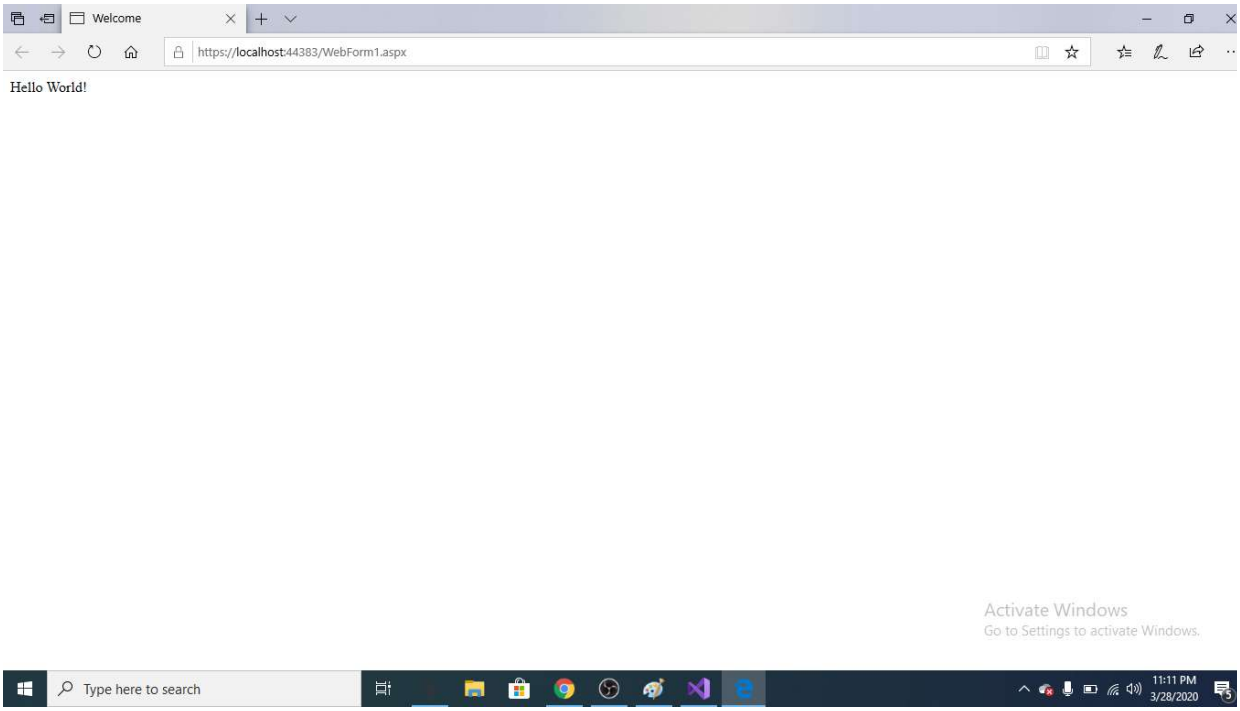
6. Change the followings in WebForm1.aspx



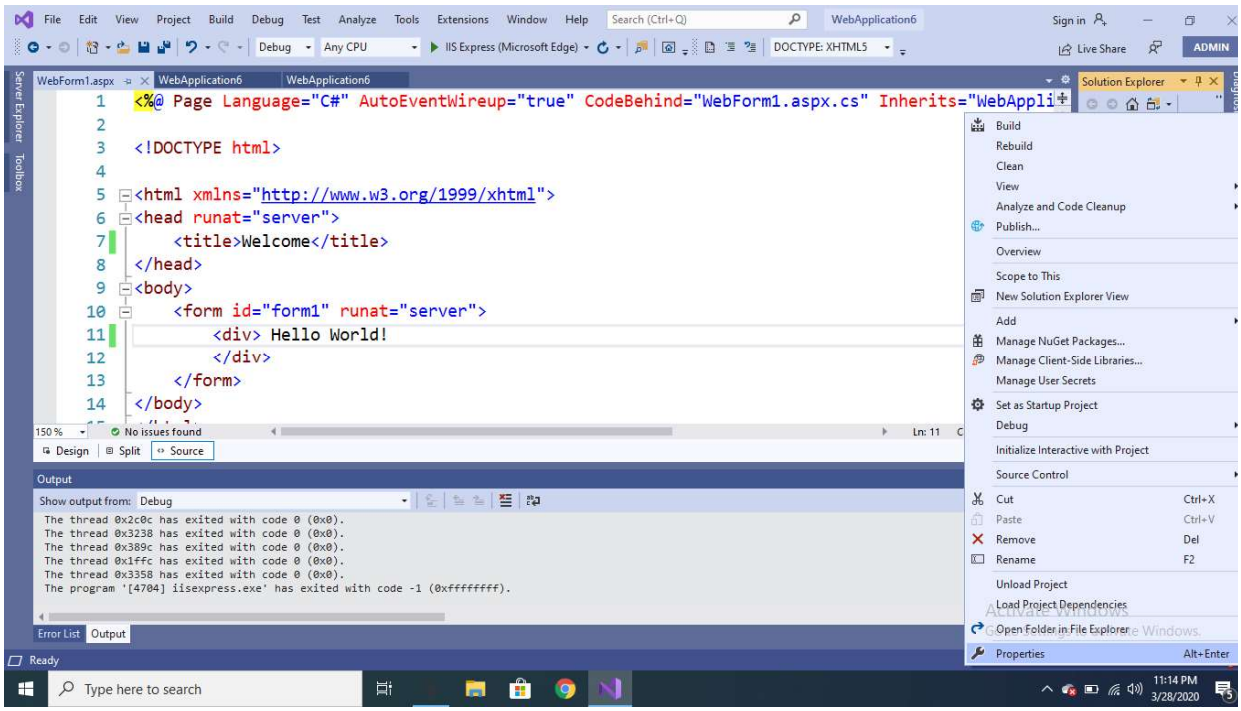
7. Click on IIS Express and run the program.



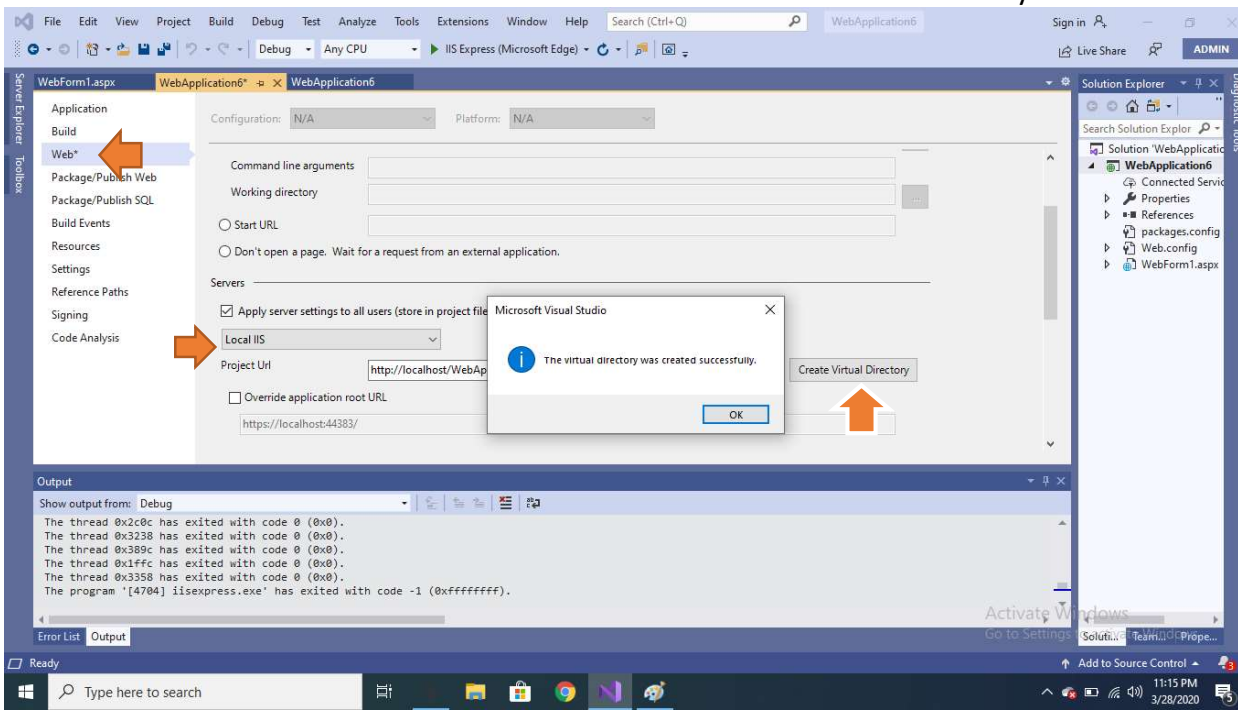
8. Output of the program.



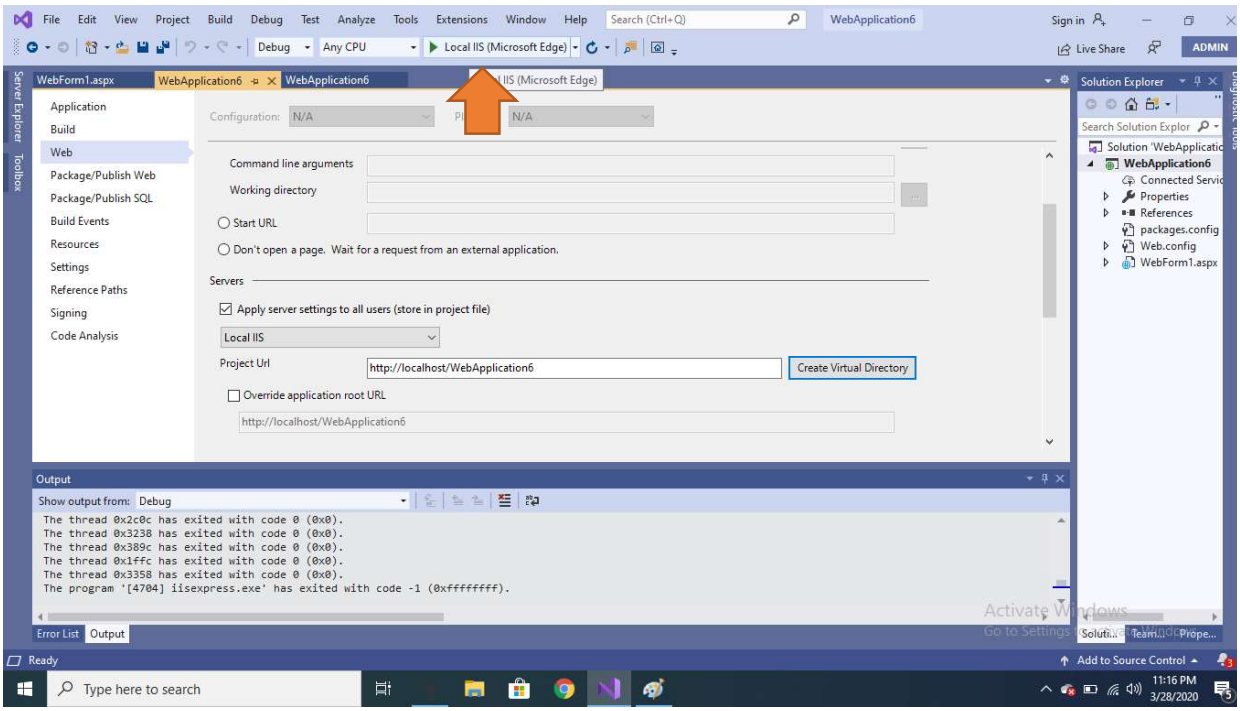
9. Now right click on the project name and select properties



10. Choose Web Menu and select Local IIS and Click on Create Virtual Directory



11. Click the save button



12 We already changed the Local IIS web server as a host. Now click on Local IIS button and execute your program. Give full path to your WebForm1.aspx.