Unit 1

.NET Framework & Language Constructs

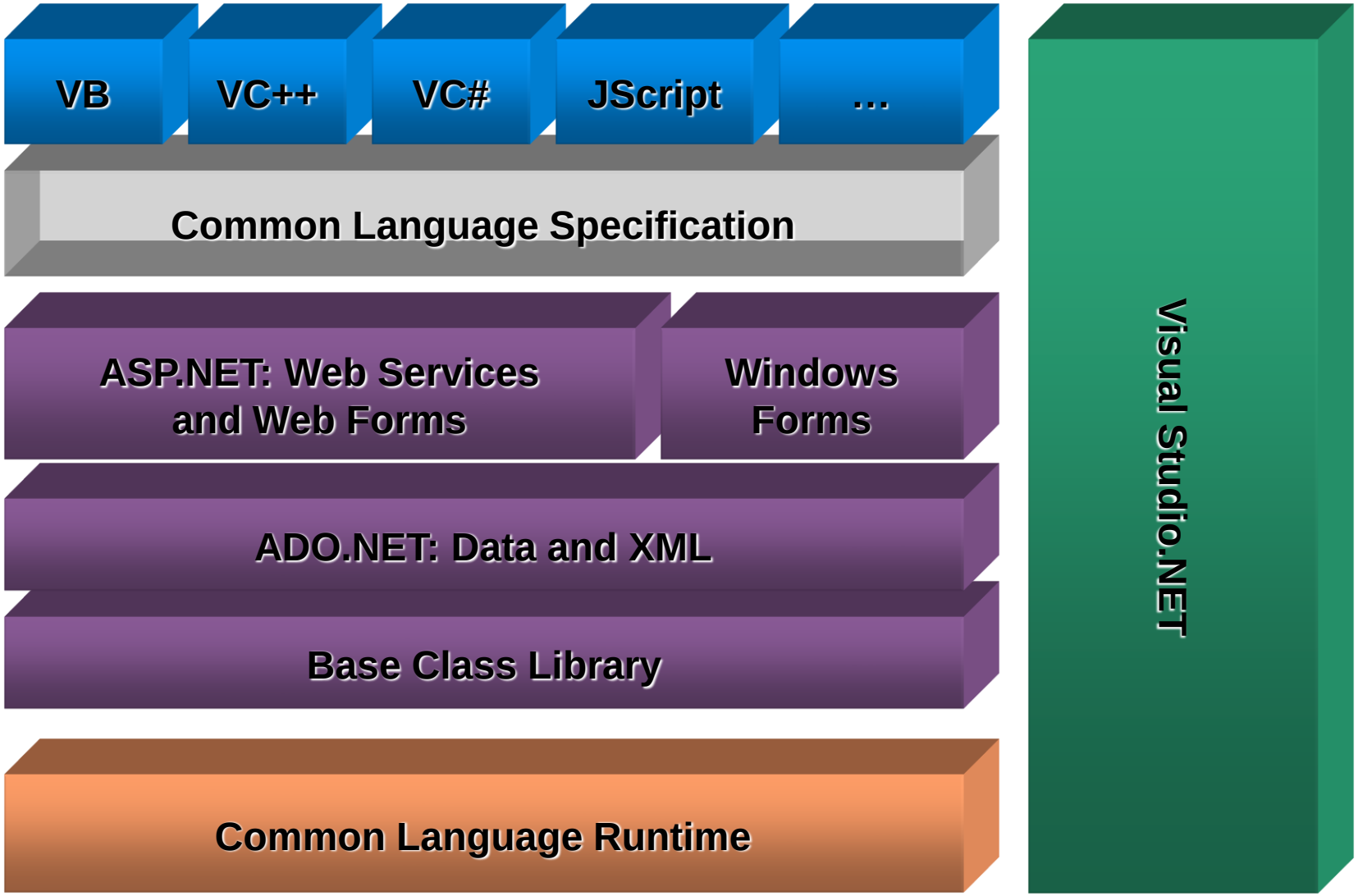
Introduction to .NET Framework

- .NET is a software framework which is designed and developed by Microsoft.
- first version of the .Net framework was 1.0 which came in the year 2002.
- and the current version is 4.7.1.
- In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#, VB.Net etc.
- used to develop Windows Form-based applications, Web-based applications, and Web services.
- VB.Net and C# being the most common ones.
- It is used to build applications for Windows, phone, web, etc.
- .NET is not a language (Runtime and a library for writing and executing written programs in any compliant language)

Introduction to .NET Framework

- .NET Framework supports more than 60 programming languages in which 11 are designed and developed by Microsoft,
- Some of them includes:
  - C#.NET
  - VB.NET
  - C++.NET
  - J#.NET
  - F#.NET
  - JSCRIPT.NET
  - WINDOWS POWERSHELL

Framework, Languages, And Tools



The .NET Framework

.NET Framework Services

- Common Language Runtime
- Windows® Forms
- ASP.NET
  - Web Forms
  - Web Services
- ADO.NET, evolution of ADO
- Visual Studio.NET

Common Language Runtime (CLR)

- CLR works like a virtual machine in executing all languages.
- All .NET languages must obey the rules and standards imposed by CLR. Examples:
  - Object declaration, creation and use
  - Data Types, language libraries
  - Error and exception handling
  - Interactive Development Environment (IDE)

Common Language Runtime(CLR)

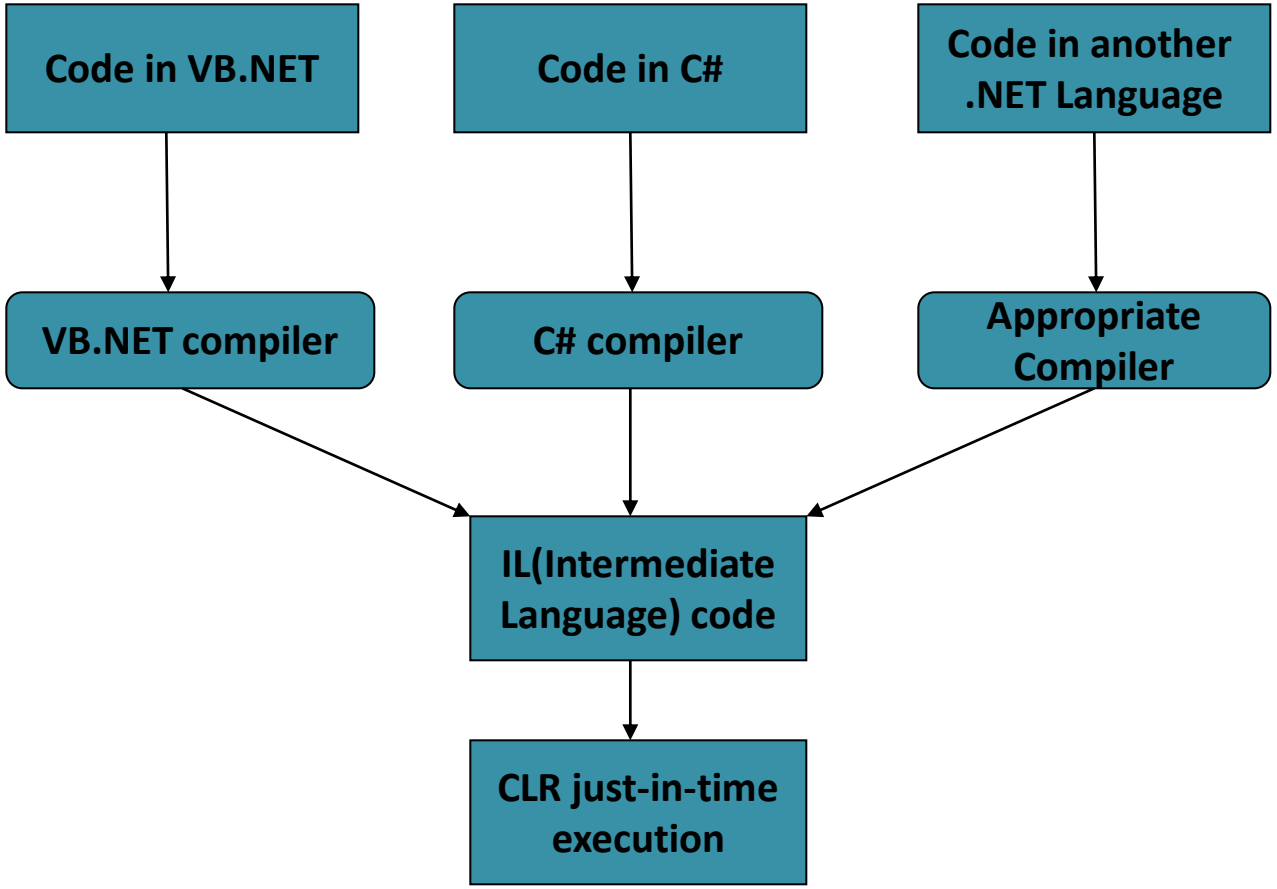
- Development
  - Mixed language applications
    - Common Language Specification (CLS)
    - Common Type System (CTS)
    - Standard class framework
    - Automatic memory management
  - Consistent error handling and safer execution
  - Potentially multi-platform
- Deployment
  - Removal of registration dependency
  - Safety – fewer versioning problems

Common Language Runtime

Multiple Language Support

- CTS is a rich type system built into the CLR
  - Implements various types (int, double, etc)
  - And operations on those types
- CLS is a set of specifications that language and library designers need to follow
  - This will ensure interoperability between languages

Compilation and Execution of .NET Application



Compilation and Execution of .NET Application

- Any code written in any .NET complaint languages when compiled, converts into MSIL (Microsoft Intermediate Language) code in form of an assembly through CLS, CTS.
- IL is the language that CLR can understand.
- On execution, this IL is converted into binary code(machine code) by CLR's just in time compiler (JIT) and these assemblies or DLL are loaded into the memory.
- Compilation can be done with Debug or Release configuration. The difference between these two is that in the debug configuration, only an assembly is generated without optimization. However, in release complete optimization is performed without debug symbols.

Basic Languages constructs

- Data Types
- Variables
- Conditional Statements
- Looping Statements
- Array
- Functions
- Class, Object, Methods, Properties
- Inheritance, Polymorphism

Lets Get Started

- Before we begin, download visual studio 2019
- Here is the download link
- https://visualstudio.microsoft.com/downloads/

C# Overview

- C# is general-purpose, object-oriented programming language developed by Microsoft.
- C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.
- Reasons - C# a widely used professional language:
  - It is object oriented & structured language
  - It is component oriented.
  - It is easy to learn & produces efficient programs.
  - It can be compiled on a variety of computer platforms.
  - a part of .Net Framework.

Strong Programming Features of C#

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
- Assembly Versioning
- Properties and Events
- Delegates and Events Management
- Easy-to-use Generics
- Indexers
- Conditional Compilation
- Simple Multithreading
- LINQ and Lambda Expressions
- Integration with Windows

C# Program Structure

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {



        }
    }
}
```

C# - Program.cs (First Program in C#)

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {



        }
    }
}
```

Data Types

- int
- short
- long
- char
- string
- bool
- float
- decimal
- double
- object

```
float f1 = 10.31f ;
decimal d1 = 23.34m ;
string name = “Your Name” ;

int x = 10;
object obj = x; // boxing, implicit
int y = (int) obj ;  //unboxing, explicit
```

C# - Program.cs (First Program in C#)

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("My First Program in C#");
            Console.ReadKey();
         }
    }
}
```

Example

```
static void Main(string[] args)
{
    Console.WriteLine("This is my First Program in C#");
    int n1, n2;
    Console.Write("Enter n1 : ");
    n1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter n2 : ");
    n2 = int.Parse(Console.ReadLine());
    int sum = n1 + n2;
    Console.WriteLine("The sum is : " + sum);
    Console.WriteLine("The sum of " + n1 + " and " + n2 + " " + sum);
    Console.WriteLine("The sum of {0} and {1} is {2}", n1, n2, sum);
    Console.ReadKey();
}
```

Operator

- Arithmetic :
  - +, -, *, /, %, ++, --
- Comparison :
  - >, <, >=, <=, ==, ===, !=
- Logical :
  - && (AND), || (OR), ! (NOT)
- Assignment :
  - =, +=, -=, *=, /=, %=
- Conditional or Ternary - **? :**

Arithmetic Operator

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operator

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operator

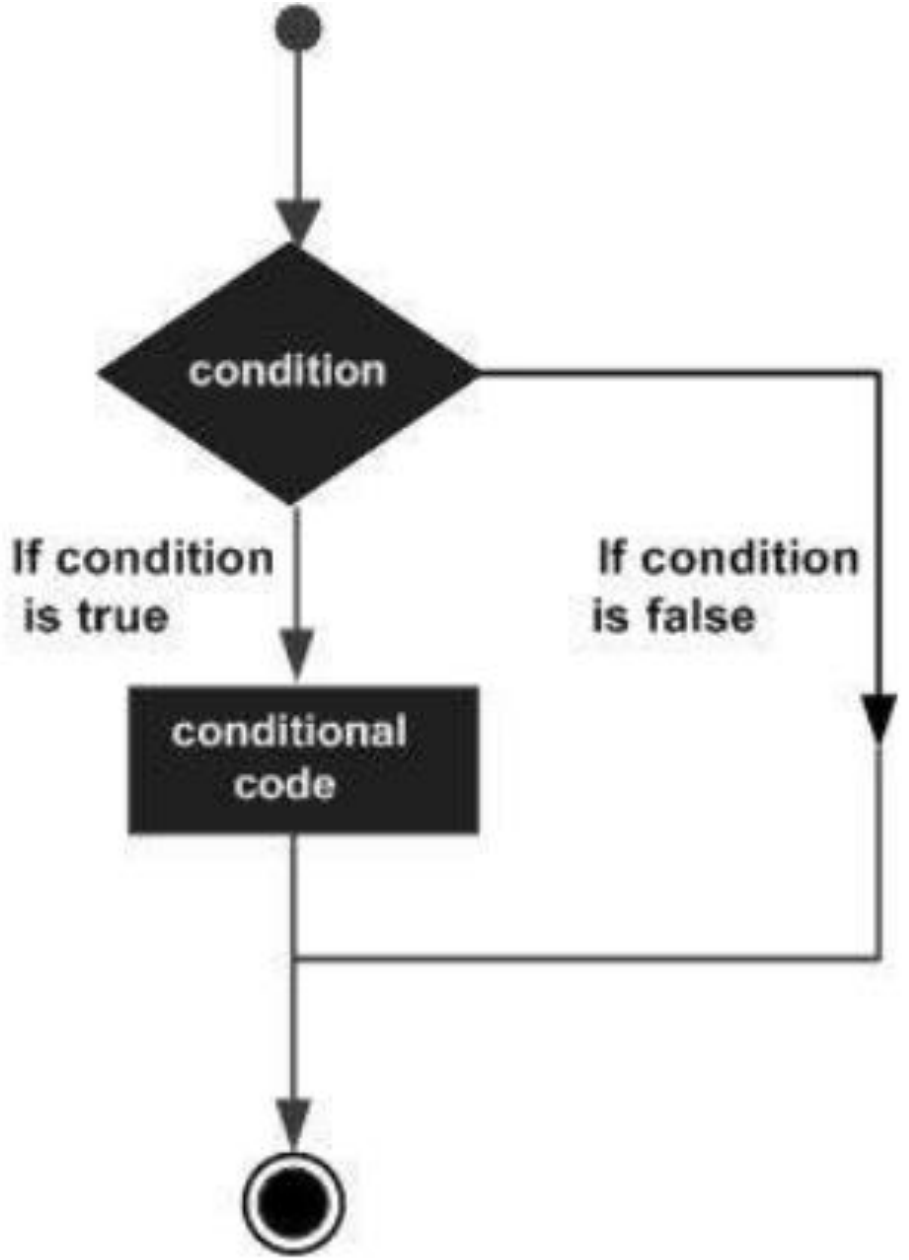
Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operator

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Controlling Program Flow

- Conditions: Making Decisions – 2 Ways
  - if … else statement
  - Switch ....case



Controlling Program Flow

- forms of **if..else** statement
  - if statement
  - if...else statement
  - if...else if... statement.

- Syntax
  - if(expression) {

    Statement(s) to be executed if true

    }

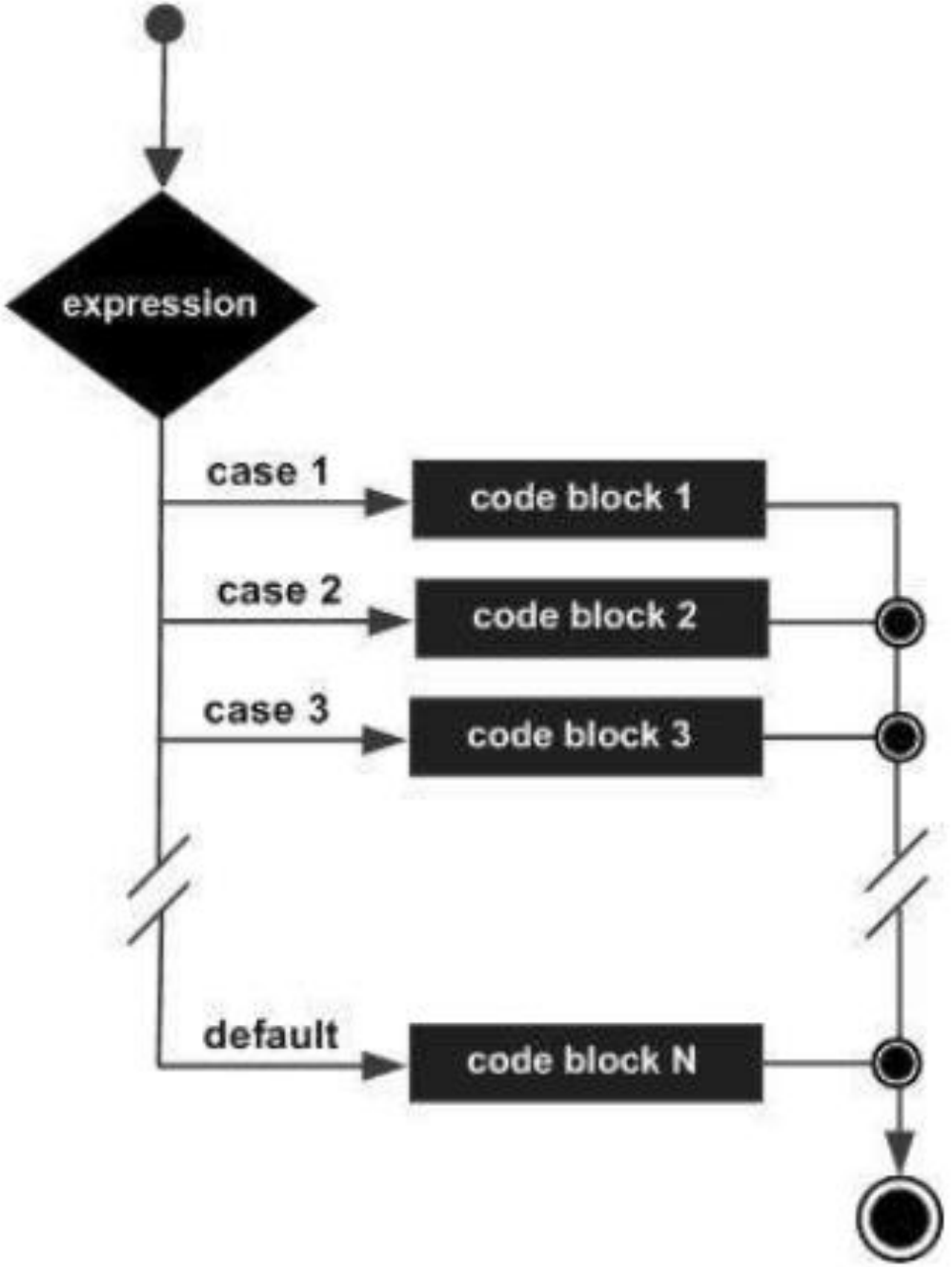
Example
If - else

```
namespace ConsoleApp1
{
    class CondtionalStmt
    {
        static void Main(string[] args)
        {
            Console.Write("Enter for n: ");
            int n = Convert.ToInt32(Console.ReadLine());
            string res = "";
            if (n > 0)
                res = "Greater than 0";
            else if (n < 0)
                res = "Less than 0";
            else
                res = "Equals 0";

            Console.WriteLine(res);
            Console.ReadKey();

        }
    }
}
```

Controlling Program Flow

- switch ....case

```
switch (expression) {
    case condition 1:
     statement(s);
     break;
    case condition 2:
     statement(s);
     break;
    ...
    case condition n:
     statement(s);
     break;
    default: statement(s)
}
```

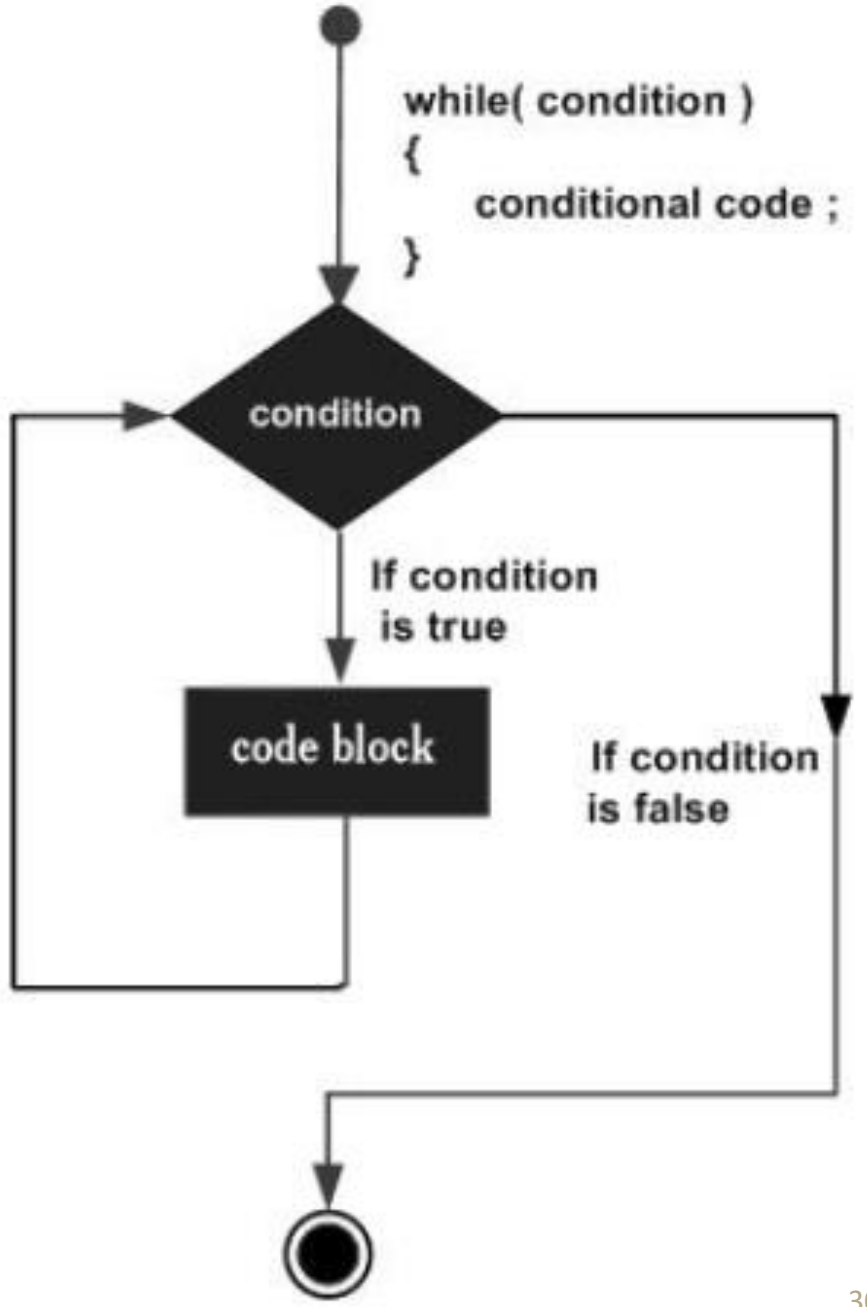


Example switch case

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

    int n = 1;
    switch (n)
    {
        case 1:
            Console.WriteLine("Case 1");
            break;
        case 2:
            Console.WriteLine("Case 2");
            break;
        default:
            Console.WriteLine("Default case");
            break;
    }
}
```

Loop

- Used to perform an action repeatedly till satisfied condition meets.
- 3 Types of Loops
  - While loop
  - Do While loop
  - For loop
- These loops have
  - Initialization statement
  - Condition statement
  - Update (increment or decrement) statement



While Loop

```
Console.WriteLine("Starting Loop");
int i = 1;
while(i <=10)
{
    Console.WriteLine("Count is : " + i);
    i++;
}
Console.WriteLine("Loop Stopped!");
```

```
Starting Loop
Count is : 1
Count is : 2
Count is : 3
Count is : 4
Count is : 5
Count is : 6
Count is : 7
Count is : 8
Count is : 9
Count is : 10
Loop stopped!
```

do while loop

```
Console.WriteLine("Starting Loop");
int i = 1;
do
{
    Console.WriteLine("Count is : " + i);
    i++;
} while (i <= 10);
Console.WriteLine("Loop Stopped!");
```

Loop – For loop

- Syntax

```
for (initialize; condition; iteration) {
        Statement(s) to be executed if test condition is true
}
```

- Ex

```
for (int i=1; i<=10; i++) {
        Console.WriteLine(”Count is : ”  + i);
}
```

Array & for each loop

```
class SecondProgram
{
    0 references
    public static void Main(string[] args)
    {
        Console.WriteLine("Second Program");
        int[] arr = { 10, 20, 30 };
        for (int i = 0; i < arr.Length; i++)
            Console.WriteLine("arr[{0}] = {1}", i, arr[i]);

        int sum = 0;
        foreach(int j in arr)
        {
            sum += j;
        }
        Console.WriteLine("The sum is : " + sum);
        Console.ReadKey();
    }
}
```

Strings

- Used for storing and manipulating text
- A string variable contains zero or more characters within double quotes.

```
// J a v a s c r i p t
// 0 1 2 3 4 5 6 7 8 9
string s = "Javascript";
Console.WriteLine("Length: " + s.Length);
Console.WriteLine("Index Of: " + s.IndexOf("va"));
Console.WriteLine("Upper Case: " + s.ToUpper());
Console.WriteLine("Lower Case: " +    s.ToLower());
Console.WriteLine("Substring: " + s.Substring(0,4));
Console.ReadKey();
```

Functions

```
class FunctionTest
{
    static void Main(string[] args)
    {
        F1();
        int sq = Square(10);
        Console.WriteLine(sq);
    }

    static void F1()
    {
        Console.WriteLine("This is F1 function");
    }
    static int Square(int n)
    {
        return n * n;
    }
}
```

Function Overloading

```
static void Main(string[] args)
{
    Console.WriteLine("Min : " + Min(5,10));
    Console.WriteLine("Min : " + Min("cow", "hen"));
}
static int Min(int n1, int n2)
{
    if (n1 < n2)
        return n1;
    else
        return n2;
}
static string Min(string s1, string s2)
{
    if (s1.CompareTo(s2) < 0)
        return s1;
    else
        return s2;
}
```

Class & Object

- A *class* consists of data declarations plus functions that act on the data.
  - Normally the data is private
  - The public functions (or methods) determine what clients can do with the data.

- An instance of a class is called an *object*.
  - Objects have identity and lifetime.
  - Like variables of built-in types.

37

Encapsulation

- By default the class definition *encapsulates*, or hides, the data inside it.
- Key concept of object oriented programming.
- The outside world can see and use the data only by calling the build-in functions.
  - Called “methods”

38

Class Members

- Methods and variables declared inside a class are called *members* of that class.
  - Member variables are called *fields*.
  - Member functions are called *methods*.

- In order to be visible outside the class definition, a member must be declared *public*.

39

Objects

- An instance of a class is called an *object*.
- You can create any number of instances of a given class.
  - Each has its own identity and lifetime.
  - Each has its own copy of the fields associated with the class.
- When you call a class method, you call it through a particular object.
  - The method sees the data associated with *that object*.

40

Creating a Class

```
class Point
{
    public int x, y;
    public void Show()
    {
        Console.WriteLine("x : " + x);
        Console.WriteLine("y : " + y);
    }
}
class OOPClassTest
{
    static void Main(string[] args)
    {
        Point p = new Point();
        p.x = 10;
        p.y = 20;
        p.Show();
        Console.ReadKey();
    }
}
```

Class - Properties

```
class Point1
{
    private int x;
    private int y;
    public int X
    {
        get { return x; }
        set { x = value;  }
    }
    public int Y
    {
        get { return y; }
        set { y = value;  }
    }
    public void Show()
    {
        Console.WriteLine("x : " + x);
        Console.WriteLine("y : " + y);
    }
}
```

```
static void Main(string[] args)
{
    Point1 p = new Point1();
    p.X = 10;
    p.Y = 20;
    Console.WriteLine("X : " + p.X);
    Console.WriteLine("Y : " + p.Y);
    Console.ReadKey();
}
```

Class - Constructor

```
class Point2
{
    public int x;
    public int y;
    public Point2() { }  // Default Constructor
    public Point2(int x, int y)  //Parameterized C.
    {
        this.x = x;
        this.y = y;
    }
}
```

```
class OOPConstructorTest
{
    static void Main(string[] args)
    {
        Point2 p1 = new Point2();
        p1.x = 11;
        p1.y = 12;
        Point2 p2 = new Point2(2,5);
    }
}
```

Task

Create a class “Employee” with following specs:

- Field Members : firstName, lastName, salary
- Properties : FirstName, LastName, Salary
- Methods : ShowFullName, IncrementSalary(double s)
- Constructor : Employee(__ , ___ , ___)

Now, create object of Employee(“Ram”, “Bahadur”, 20000)

- Show Employee Fullname & Salary
- Change FirstName to “Hari” & increment salary by 5000
- Show full name & salary

Inheritance

- New Classes called derived classes are created from existing classes called base classes

```
public class Class A
 {


 }
public class Class B : A
 {

 }
```

Inheritance Example

```
public class ParentClass
{
        public ParentClass() {
                Console.WriteLine("Parent Constructor");
        }
        public void Print() {
                Console.WriteLine("I'm a Parent Class.");
        }
}
public class ChildClass : ParentClass {
        public ChildClass (){
                Console.WriteLine("Parent Constructor");
        }
}
```

Inheritance Example

```
class Program
{
    static void Main(string[] args)
    {
                        ChildClass cc= new ChildClass();
                cc.Print();
     }
}
```

Use base key word

```
public class ParentClass
{
        public int x = 10;
        public ParentClass()
        {
                Console.WriteLine(“Parent Constructor”);
        }
        public void Print() {
                Console.WriteLine(“I’m a Parent Class.”);
        }
}
```

Use base key word

```
public class ChildClass : ParentClass
{
        public ChildClass() : base()
    {
        Console.WriteLine(“Child Constructor”);
        base.Print();
        Console.WriteLine(base.x);
    }
}
```

Inheritance Example

```
class Program
{
    static void Main(string[] args)
    {
                ChildClass cc= new ChildClass()
                cc.Print();
                Console.ReadKey();
     }
}
```

Indexer

- 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 ([ ]).

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

Indexer Example

```
class Student
{
    private int roll;
    public int Roll
    {
        get { return roll; }
        set { roll = value; }
    }
    private int[] marks = new int[3];
    public int this[int index]
    {
        get { return marks[index]; }
        set { marks[index] = value; }
    }
    public double GetPercent()
    {
        double total = 0.0;
        foreach (int m in marks)
        {
            total = total + m;
        }
        return total / marks.Length;
    }
}
```

Indexer Example

```
static void Main()
{
    Student s1 = new Student();
    s1.Roll = 1;
    s1[0] = 50;
    s1[1] = 25;
    s1[2] = 30;
    Console.WriteLine(s1.GetPercent());
    Student s2 = new Student();
    s2.Roll = 2;
    s2[0] = 20;
    s2[1] = 30;
    s2[2] = 40;
    Console.WriteLine(s2.GetPercent());
}
```

The sealed class

- Sealed classes are used to restrict the inheritance feature of object oriented programming. Once a class is defined as a sealed class, the class cannot be inherited.
- In C#, the sealed modifier is used to define a class as sealed

```
sealed class SealedClass
{

}
```

Abstract Class

- Classes can be declared as abstract by using keyword abstract.
- Abstract classes are one of the essential behaviors provided by .NET.
- If you like to make classes that only represent base classes, and don’t want anyone to create objects of these class types, use abstract class to implement such functionality.
- Object of this class can be instantiated, but can make derivations of this.
- The derived class should implement the abstract class members.

Abstract Class

```
abstract class AbsClass
{
    public abstract void AbstractMethod();
    public void NonAbstractMethod()
    {
        Console.WriteLine("NonAbstract Method");
    }
}

class Derived : AbsClass
{
    public override void AbstractMethod()
    {
        Console.WriteLine("Overriding AbstractMethod in Derived Class");
    }
}
```

Abstract Class

```
class OOPAbstractClass
{
    static void Main(string[] args)
    {
        Derived d = new Derived();
        d.NonAbstractMethod();
        d.AbstractMethod();
        Console.ReadKey();
    }
}
```

Interface

- An interface is not a class. It is an entity that is defined by the keyword Interface.
- By Convention, Interface Name starts with letter ‘I’
- has no implementation; just the declaration of the methods without the body.
- a class can implement more than one interface but can only inherit from one class.
- interfaces are used to implement multiple inheritance.

```
interface IFace
{

}
```

Partial Classes

- In C#, a class definition can be divided over multiple files.
  - Helpful for large classes with many methods.
  - Used by Microsoft in some cases to separate automatically generated code from user written code.

- If class defintion is divided over multiple files, each part is declared as a *partial* class.

Partial Classes

In file circ1.cs

```
partial class Circle
{
  // Part of class defintion
  ...
}
```

In file circ2.cs

```
partial class Circle
{
  // Another part of class definition
  ...
}
```

Exception Handling

- An exception is a problem that arises during the execution of a program.
- A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero, Array Index Out of Bounds, etc
- Exceptions provide a way to transfer control from one part of a program to another.
- C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

Exception Handling

- **try**: A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally**: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
- **throw**: A program throws an exception when a problem shows up. This is done using a throw keyword.

Exception Handling

Syntax

```
try
{
    // statements causing exception
}
catch( ExceptionName e1 )
{
    // error handling code
}
catch( ExceptionName e2 )
{
    // error handling code
}
catch( ExceptionName eN )
{
    // error handling code
}
finally
{
    // statements to be executed
}
```

Exception Handling

- What will happen to this program?
- In which line, we encounter the error?
- Will this execute all statements?
- Can this program display the last 2 lines?

1. int a = 10;
2. int b = 0;
3. int c = a / b;
4. Console.WriteLine(c);
5. Console.WriteLine(“This is last line”);

Exception Handling

```
try
{
    int a = 10;
    int b = 0;       // assign b = 2
    int c = a / b;
    Console.WriteLine(c);
    int[] arr = {10, 20, 12};
    Console.WriteLine(arr[5]);
}
catch (DivideByZeroException e1)
{
    Console.WriteLine(e1.ToString());
}
catch (IndexOutOfRangeException e2)
{
    Console.WriteLine("Array index problem");
}
```

Delegate

- C# delegates are similar to pointers to functions, in C or C++.
- A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.
- Delegates are especially used for implementing events and the call-back methods.
- Syntax – Delegate Declaration :

delegate <return-type> <delegate_name> <params>

Delegate

- Delegate Declaration :

delegate <return-type> DelegateName> <arg_list>

- Object Creation :

DelegateName d = new DelegateName<function to which the delegate points>

Invoking :

d<list of args that are to be passed to the functions>

Delegate - Ex

```
// 1. Declaration
public delegate void SimpleDelegate();
class DelegateTest
{
    static void Main(string[] args)
    {
        // 2. Instantiation
        SimpleDelegate d = new SimpleDelegate(MyFunc);
        d();          // 3. Invocation
    }
    public static void MyFunc()
    {
        Console.WriteLine("I was called by delegate");
    }
}
```

Collection Types

- Collection Types are specialized classes for data storage and retrieval.
- These classes provide support for stacks, queues, lists, and hash tables.
- Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.
- Namespaces:
  - System.Collection
  - System.Collection.Generic

Collection Types

- System.Collection
  - ArrayList, Hashtable, SortedList, Stack, Queue
- System.Collection.Generic
  - generic collection is strongly typed (type safe), that you can only put one type of object into it.
  - This eliminates type mismatches at runtime.
  - Another benefit of type safety is that performance is better
  - Ex: List, Dictionary

Array List – System.Collections

```
ArrayList al = new ArrayList();
al.Add(1);
al.Add("Hari");
al.Add(3.4);
Console.WriteLine(al.Count);

al.Remove(3.4);
al.RemoveAt(0);
Console.WriteLine(al.Count);
```

List – System.Collection.Generic

```
List<string> names = new List<string>();
names.Add("Ram");
names.Add("Hari");
names.Add("Sam");
Console.WriteLine(names.IndexOf("Ram"));

Console.WriteLine(names.Count);
names.RemoveAt(2);
names.Remove("Ram");
foreach(string n in names)
{
    Console.WriteLine(n);
}
```

Unit 2

Introduction to ASP.NET

ASP.NET

- ASP.NET is a web application framework designed and developed by Microsoft.
- a subset of the .NET Framework and successor of the classic ASP (Active Server Pages).
- With version 1.0 of the .NET Framework, it was first released in January 2002.
- before the year 2002 for developing web applications and services, there was Classic ASP.

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

.NET Core

- .NET Core is a new version of .NET Framework
- general-purpose development platform maintained by Microsoft.
- It is a cross-platform framework that runs on Windows, macOS, and Linux operating systems, used to build different types of applications such as mobile, desktop, web, cloud, IoT, machine learning, microservices, game, etc.
- .NET Core is written from scratch to make it modular, lightweight, fast, and cross-platform Framework.
- It includes the core features that are required to run a basic .NET Core app. Other features are provided as NuGet packages, which you can add it in your application as needed. In this way, the .NET Core application speed up the performance, reduce the memory footprint and becomes easy to maintain.

.NET Core Characteristics

- **Open-source Framework:** .NET Core is an open-source framework maintained by Microsoft and available on GitHub under MIT and Apache 2 licenses. It is a .NET Foundation project.
- **Cross-platform:** .NET Core runs on Windows, macOS, and Linux operating systems. There are different runtime for each operating system that executes the code and generates the same output.
- **Consistent across Architectures**: Execute the code with the same behavior in different instruction set architectures, including x64, x86, and ARM.
- **Wide-range of Applications:** Various types of applications can be developed and run on .NET Core platform such as mobile, desktop, web, cloud, IoT, machine learning, microservices, game, etc.

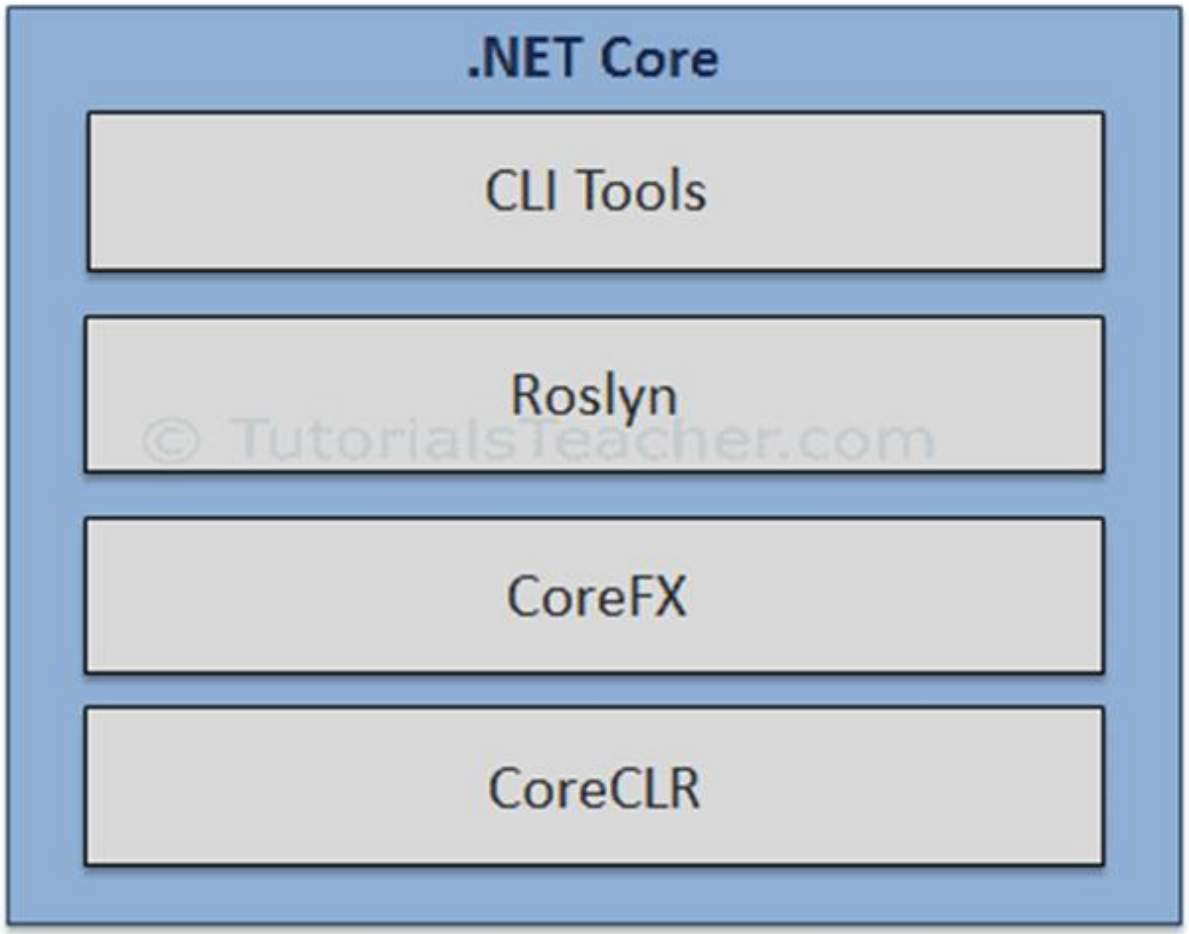
- **Supports Multiple Languages:** You can use C#, F#, and Visual Basic programming languages to develop .NET Core applications. You can use your favorite IDE, including Visual Studio 2017/2019, Visual Studio Code, Sublime Text, Vim, etc.
- **Modular Architecture**: supports modular architecture approach using NuGet packages for various features that can be added to the .NET Core project as needed. Even the .NET Core library is provided as a NuGet package. The NuGet package for the default .NET Core application model is Microsoft.NETCore.App. It reduces the memory footprint, speeds up the performance, and easy to maintain.
- **CLI Tools:** .NET Core includes CLI tools (Command-line interface) for development and continuous-integration.
- **Flexible Deployment:** .NET Core application can be deployed user-wide or system-wide or with Docker Containers.
- **Compatibility:** Compatible with .NET Framework and Mono APIs by using .NET Standard specification

.NET Core Version History

Version	Latest Version	Visual Studio	Release Date	End of Support
.NET 5	Preview 1	VS 2019	16th March, 2020	
.NET Core 3.x - latest	3.1.3	VS 2019	24th March, 2020	12th March, 2022
.NET Core 2.x	2.1.17	VS 2017, 2019	24th March, 2020	21st August, 2021
.NET Core 1.x	1.1.13	VS 2017	14th May, 2019	27th May, 2019

.NET Core Framework parts

- **CLI Tools:** A set of tooling for development and deployment.
- **Roslyn:** Language compiler for C# and Visual Basic
- **CoreFX:** Set of framework libraries.
- **CoreCLR:** A JIT based CLR (Command Language Runtime).



Mono

- Mono is an example of a cross-platform framework available on Windows, macOS, Linux, and more. It was first designed as an open source implementation of the .NET Framework on Linux.
- Mono (like .NET) is tied heavily around the C# programming language, known for its high level of portability.
- For example, the Unity game engine uses C# as a cross-platform way of creating video games. This is in part due to the language's design. C# can be turned into CIL (Common Intermediate Language), which can either be compiled to native code (faster, less portable), or run through a virtual machine (slower, more portable).
- Mono provides the means to compile, and run C# programs, similar to the .NET Framework.

ASP.NET Web Forms

- a part of the ASP.NET web application framework and is included with Visual Studio.
- you can use to create ASP.NET web applications, the others are ASP.NET MVC, ASP.NET Web Pages, and ASP.NET Single Page Applications.
- Web Forms are pages that your users request using their browser. These pages can be written using a combination of HTML, client-script, server controls, and server code.
- When users request a page, it is compiled and executed on the server by the framework, and then the framework generates the HTML markup that the browser can render.

ASP.NET Web Forms

- An ASP.NET Web Forms page presents information to the user in any browser or client device.
- The Visual Studio (IDE) lets you drag and drop server controls to lay out your Web Forms page. You can then easily set properties, methods, and events for controls on the page or for the page itself. These properties, methods, and events are used to define the web page's behavior, look and feel, and so on
- Based on Microsoft ASP.NET technology, in which code that runs on the server dynamically generates Web page output to the browser or client device.

Features of ASP.NET Web Forms

- **Server Controls-** ASP.NET Web server controls are similar to familiar HTML elements, such as buttons and text boxes. Other controls are calendar controls, and controls that you can use to connect to data sources and display data.
- **Master Pages-** ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior for all of the pages (or a group of pages) in your application. You can then create individual content pages along with the master page to render the web page.
- **Working with Data**- ASP.NET provides many options for storing, retrieving, and displaying data in web page UI elements such as tables and text boxes and drop-down lists.

Features of ASP.NET Web Forms

- **Client Script and Client Frameworks -** You can write client-script functionality in ASP.NET Web Form pages to provide responsive user interface to users. You can also use client script to make asynchronous calls to the Web server while a page is running in the browser.
- **Routing -** URL routing allows you to configure an application to accept request URL. A request URL is simply the URL a user enters into their browser to find a page on your web site. You use routing to define URLs that are semantically meaningful to users and that can help with search-engine optimization (SEO).
- **State Management -** ASP.NET Web Forms includes several options that help you preserve data on both a per-page basis and an application-wide basis.
- **Security -** offer features to develop secure application from various security threats.

Features of ASP.NET Web Forms

- **Performance –** offers performance related to page and server control processing, state management, data access, application configuration and loading, and efficient coding practices.
- **Internationalization -** enables you to create web pages that can obtain content and other data based on the preferred language setting or localized resource for the browser or based on the user's explicit choice of language. Content and other data is referred to as resources and such data can be stored in resource files or other sources.
- **Debugging and Error Handling -** diagnose problems that might arise in application. Debugging and error handling are well so that applications compile and run effectively.
- **Deployment and Hosting-** Visual Studio, ASP.NET, Azure, and IIS provide tools that help you with the process of deploying and hosting your application

Let’s create first ASP.NET Web Forms Project in Visual Studio 2017/2019

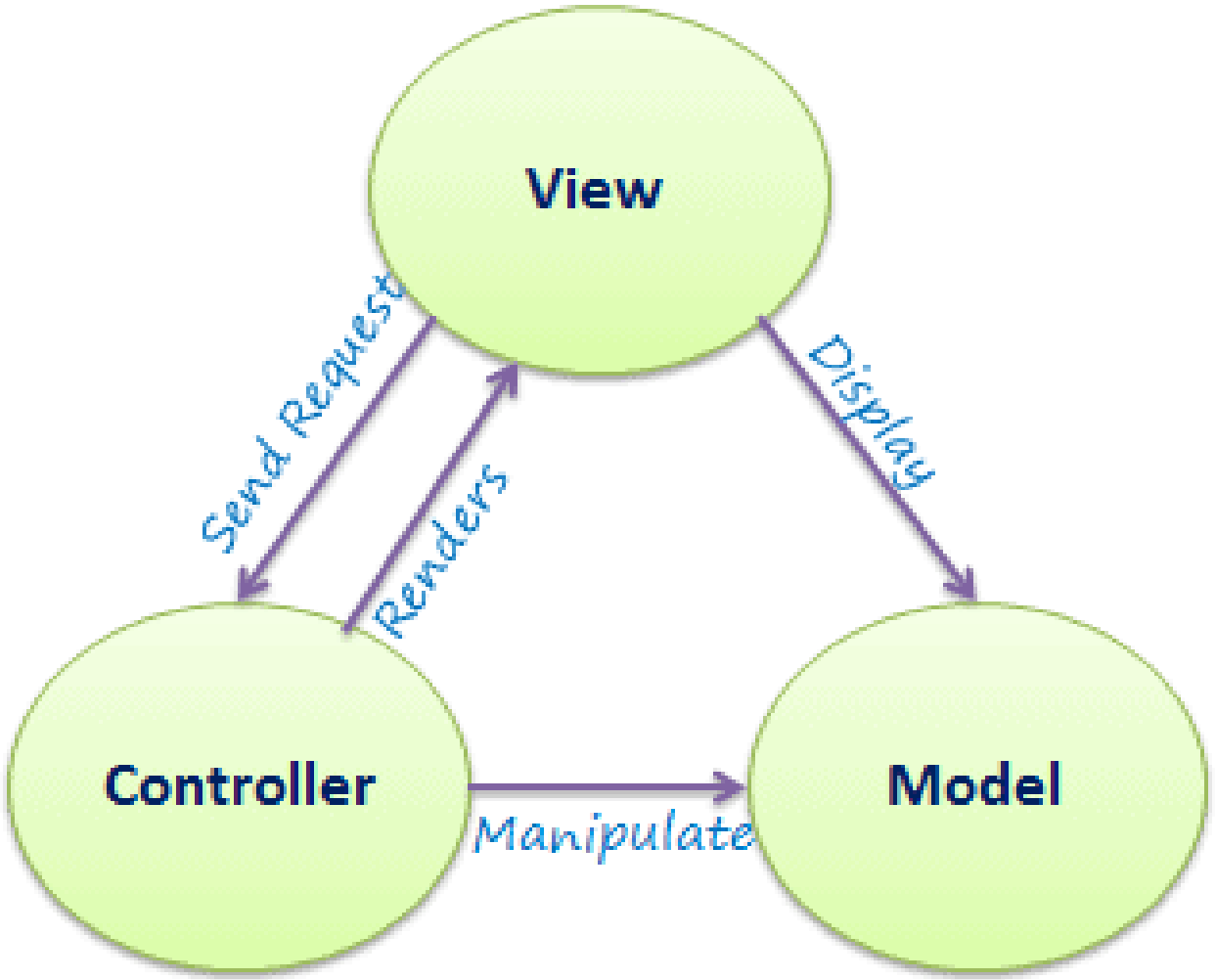
ASP.NET MVC

- ASP.NET MVC is an open source web development framework from Microsoft that provides a Model View Controller architecture.
- ASP.net MVC offers an alternative to ASP.net web forms for building web applications.
- It is a part of the .Net platform for building, deploying and running web apps.
- You can develop web apps and website with the help of HTML, CSS, jQuery, Javascript, etc.

ASP.NET MVC Architecture

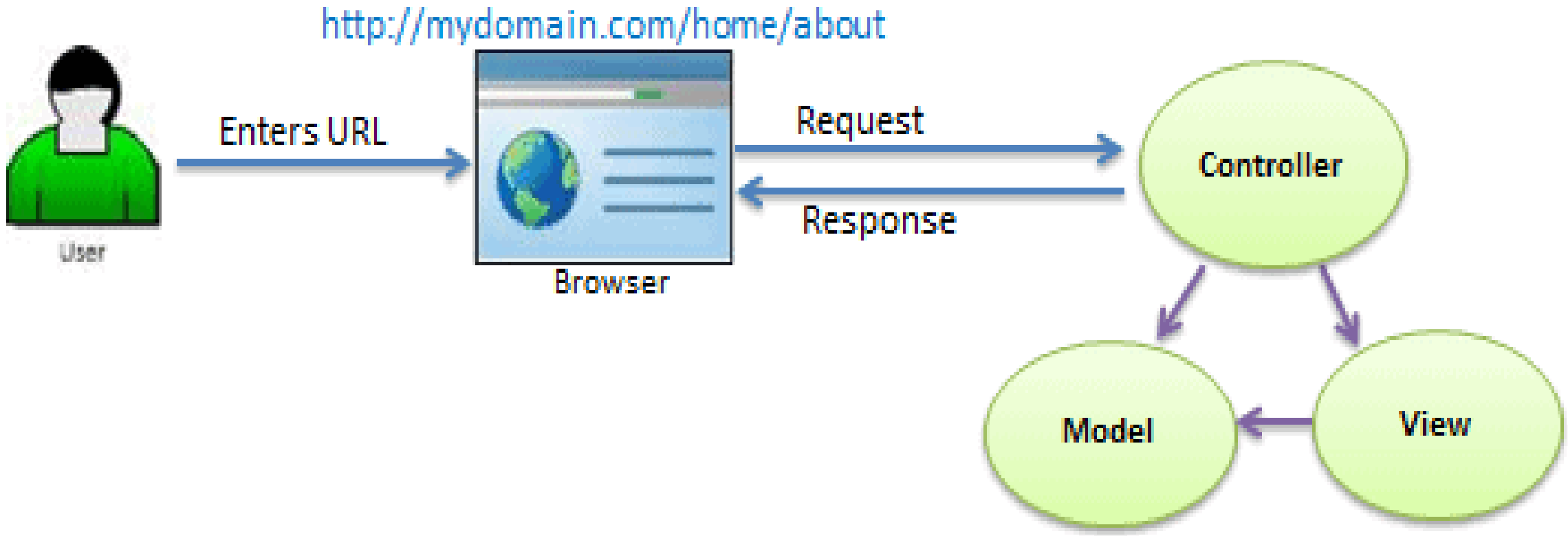
- MVC stands for Model, View, and Controller. MVC separates an application into three components - Model, View, and Controller.
- **Model**: represents the shape of the data. A class in C# is used to describe a model. Model objects store data retrieved from the database. Model represents the data.
- **View**: View in MVC is a user interface. View display model data to the user and also enables them to modify them. View in ASP.NET MVC is HTML, CSS, and some special syntax (Razor syntax) that makes it easy to communicate with the model and the controller.
- **Controller**: handles the user request. Typically, the user uses the view and raises an HTTP request. Controller processes request and returns the appropriate view as a response. Controller is the request handler.

ASP.NET MVC Architecture



Request Flow in MVC Architecture

The following figure illustrates the flow of the user's request in ASP.NET MVC.

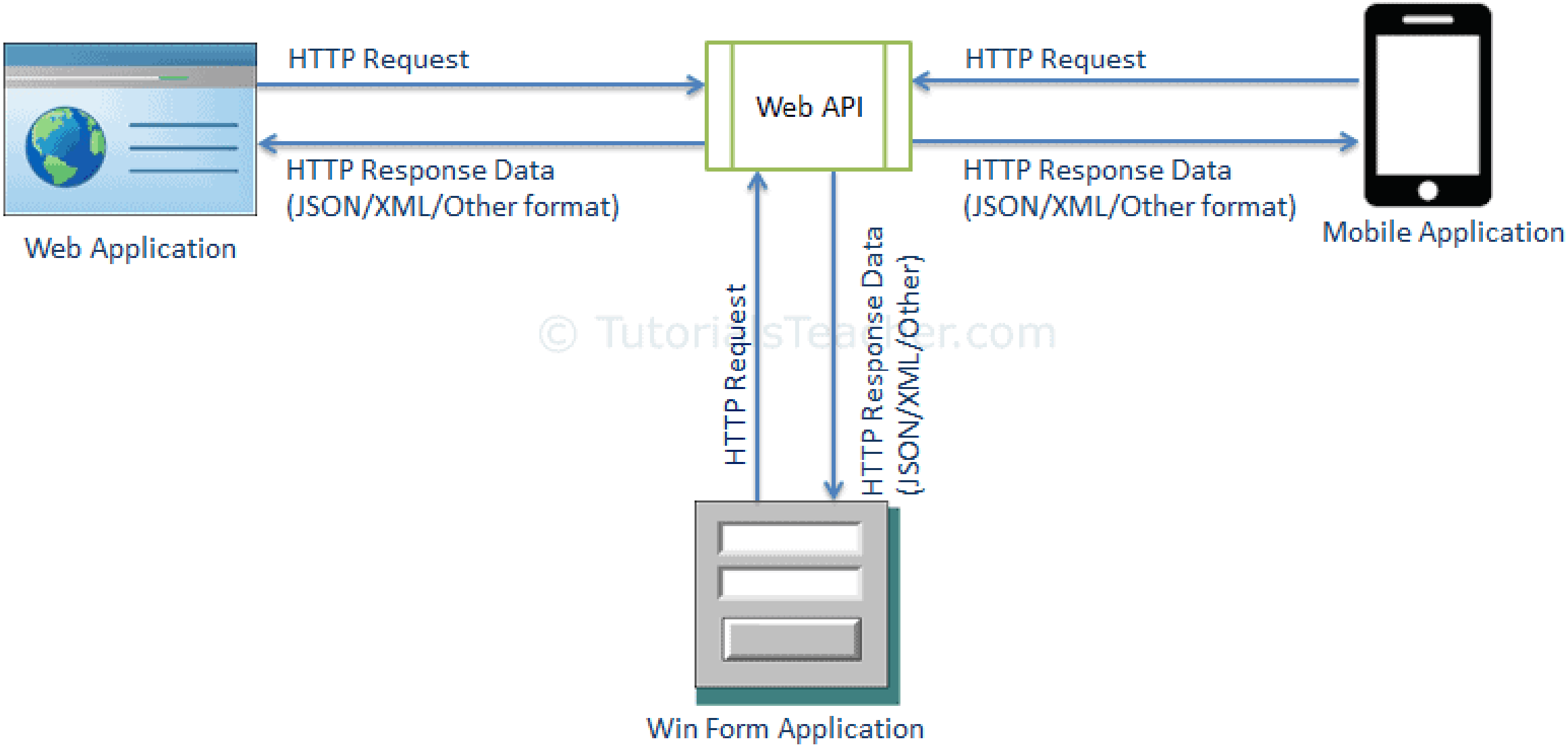


Let’s create first ASP.NET MVC Project in Visual Studio 2017/2019

ASP.NET Web API

- ASP.NET Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices.
- It is an ideal platform for building RESTful applications on the .NET Framework.
- It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view.
- like a webservice or WCF service but the exception is that it only supports HTTP protocol.

ASP.NET Web API



ASP.NET Web API Characteristics

- a framework for building HTTP services that can be accessed from any client including browsers and mobile devices.
- ideal for building RESTful applications on the .NET Framework.
- The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc.
- It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view.
- like a webservice or WCF service but the exception is that it only supports HTTP protocol.

ASP.NET Web API Project

You can create a Web API project in two ways.

- Web API with MVC Project
- Stand-alone Web API Project

ASP.NET Core

- new version of the ASP.NET web framework
- free, open-source, and cross-platform framework
- ASP.NET Core applications can run on Windows, Linux, and Mac. So you don't need to build different apps for different platforms using different frameworks.
- allows you to use and manage modern UI frameworks such as AngularJS, ReactJS, Umber, Bootstrap, etc. using Bower (a package manager for the web).

.NET Core Vs ASP.NET Core

.NET Core	ASP.NET Core
Open-source and Cross-platform	Open-source and Cross-platform
.NET Core is a runtime to execute applications build on it.	ASP.NET Core is a web framework to build web apps, IoT apps, and mobile backends on the top of .NET Core or .NET Framework.
Install .NET Core Runtime to run applications and install .NET Core SDK to build applications.	There is no separate runtime and SDK are available for ASP.NET Core. .NET Core runtime and SDK includes ASP.NET Core libraries.
.NET Core 3.1 - latest version	ASP.NET Core 3.1 There is no separate versioning for ASP.NET Core. It is the same as .NET Core versions.

ASP.NET Core

- **Supports Multiple Platforms**
- **Hosting:** ASP.NET Core web application can be hosted on multiple platforms with any web server such as IIS, Apache etc. It is not dependent only on IIS as a standard .NET Framework.
- **Fast** - This reduces the request pipeline and improves performance and scalability.
- **IoC Container:** It includes the built-in IoC container for automatic dependency injection which makes it maintainable and testable.
- **Integration with Modern UI Frameworks**
- **Code Sharing:** allow to build a class library that can be used with other .NET frameworks such as .NET Framework 4.x or Mono. Thus a single code base can be shared across frameworks.

ASP.NET Core

- **Side-by-Side App Versioning:** ASP.NET Core runs on .NET Core, which supports the simultaneous running of multiple versions of applications.
- **Smaller Deployment Footprint:** ASP.NET Core application runs on .NET Core, which is smaller than the full .NET Framework. So, the application which uses only a part of .NET CoreFX will have a smaller deployment size. This reduces the deployment footprint.

Compilation and Execution of .NET applications: CLI, MSIL and CLR

- C# programs run on the .NET Framework, which includes the common language runtime (CLR) and a unified set of class libraries. The CLR is the commercial implementation by Microsoft of the common language infrastructure (CLI), an international standard that is the basis for creating execution and development environments in which languages and libraries work together seamlessly.
- Source code written in C# is compiled into an Microsoft Intermediate Language (MSIL) or simply(IS) that conforms to the CLI specification. The IL code are stored on disk in an executable file called an assembly, typically with an extension of .exe or .dll.
- CLR performs just in time (JIT) compilation to convert the IL code to native machine instructions. The CLR also provides other services related to automatic garbage collection, exception handling, and resource management.

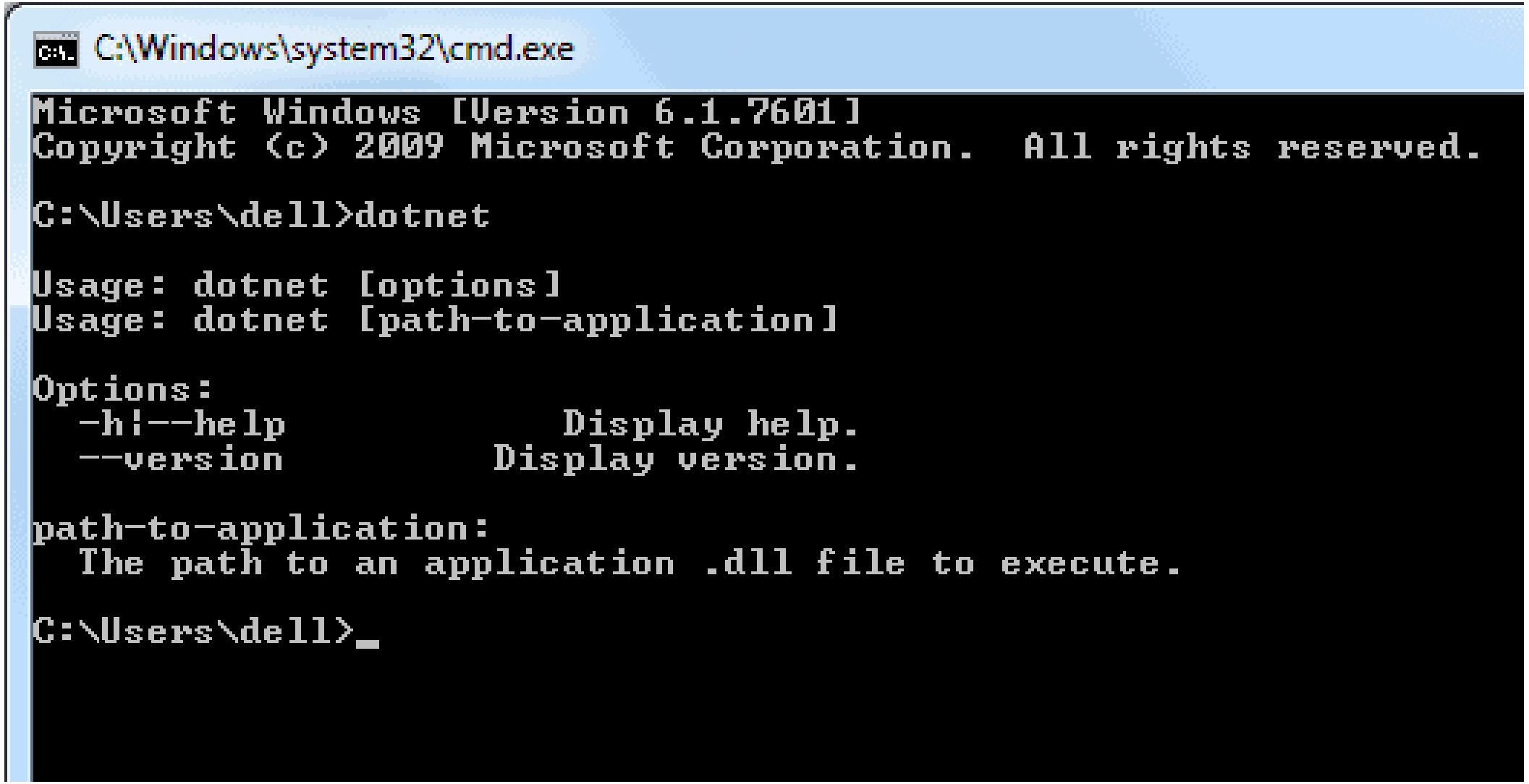
Compilation and Execution of .NET applications: CLI, MSIL and CLR

- Code that is executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code" which is compiled into native machine language that targets a specific system.
- Language interoperability is a key feature of the .NET Framework. Because the IL code produced by the C# compiler conforms to the Common Type Specification (CTS), IL code generated from C# can interact with code that was generated from the .NET versions of Visual Basic, Visual C++, or any of more than 20 other CTS-compliant languages. A single assembly may contain multiple modules written in different .NET languages, and the types can reference each other just as if they were written in the same language.

NET CLI: build, run, test and deploy .NET Core Applications

- The .NET Core command-line interface (CLI) is a new cross-platform tool for creating, restoring packages, building, running and publishing .NET applications.
- Visual Studio internally uses this CLI to restore, build and publish an application. Other higher level IDEs, editors and tools can use CLI to support .NET Core applications.
- The .NET Core CLI is installed with .NET Core SDK for selected platforms. So we don't need to install it separately on the development machine. We can verify whether the CLI is installed properly by opening command prompt in Windows and writing dotnet and pressing Enter. If it displays usage and help as shown below then it means it is installed properly.

NET CLI: build, run, test and deploy .NET Core Applications



Creating and running the Hello World console application

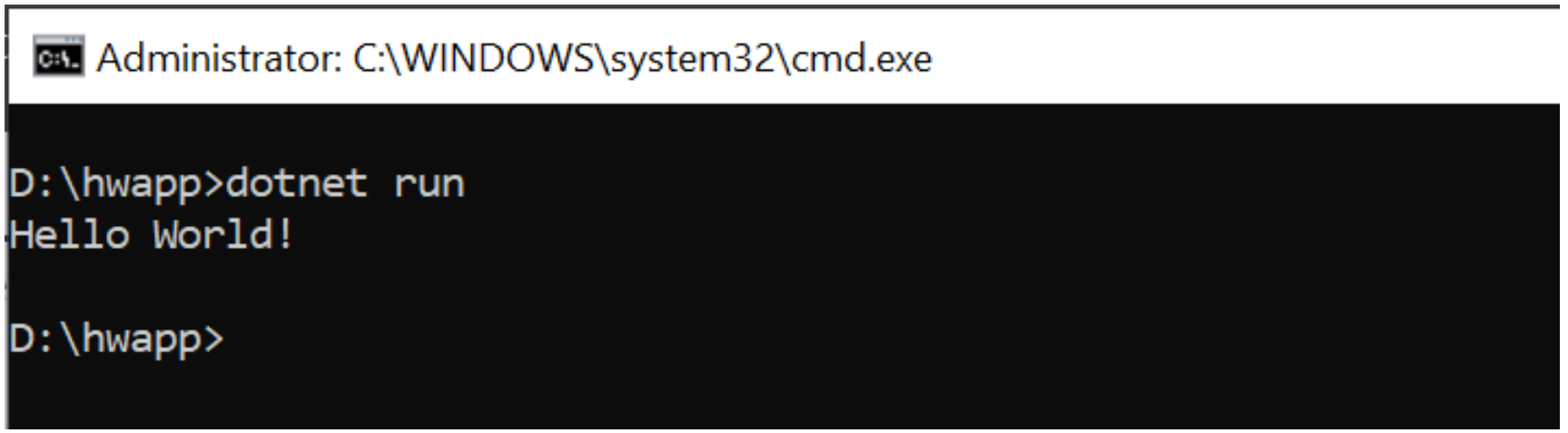
- Execute the following commands on the command line or terminal:
  - `mkdir hwapp`
  - `cd hwapp`
  - `dotnet new console`
- The command **dotnet new console** creates a new Hello World console application in the current folder.
- The dotnet new console command creates two files:
  - Program.cs and
  - hwapp.csproj

Program.cs should look similar to the following listing

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

Running the Hello World console application

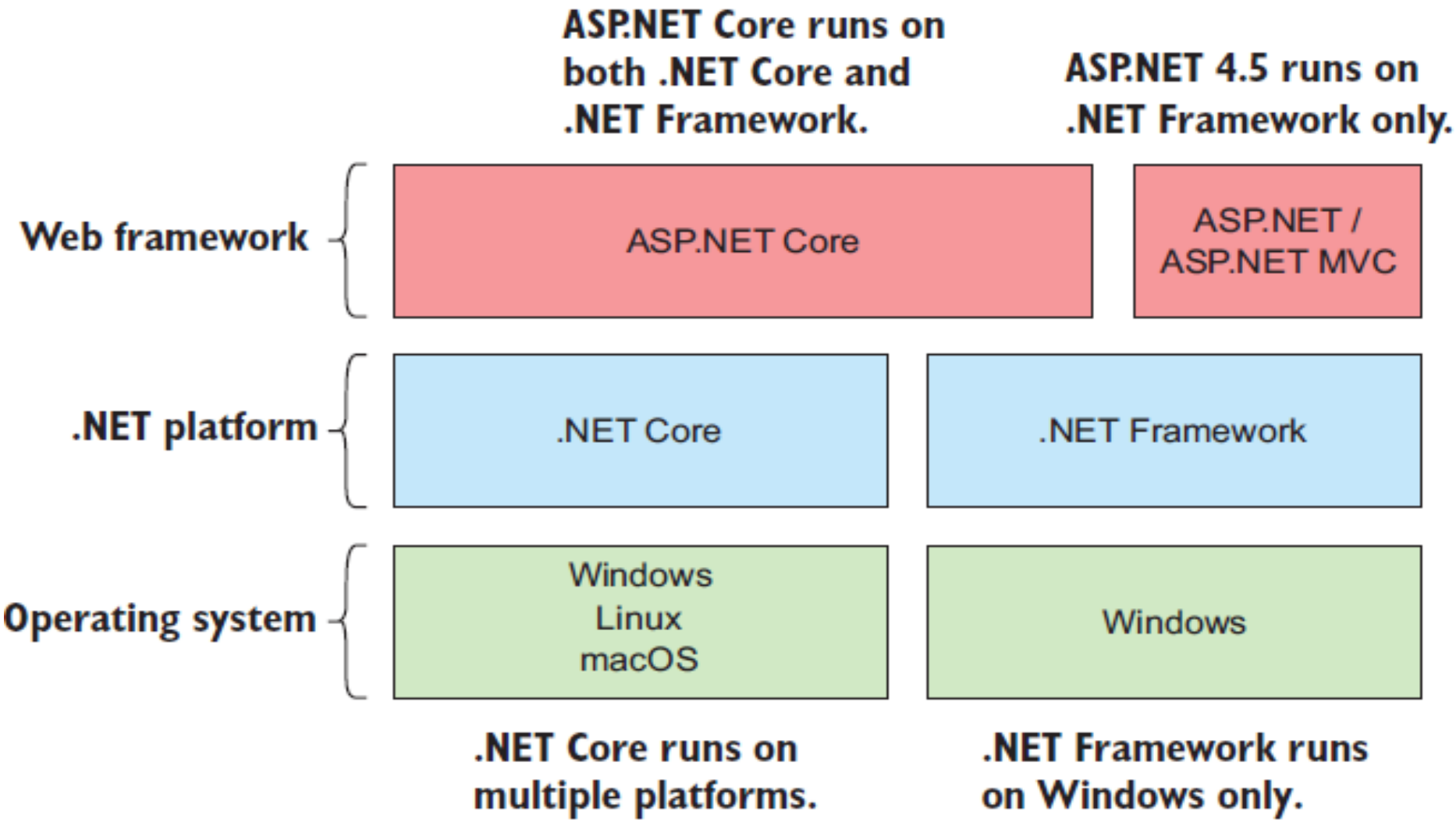
- When you're using the .NET Core SDK, your application will be built automatically when needed. There's no need to worry about whether or not you're executing the latest code.
- Try running the Hello World application by executing *dotnet run* at the command line or terminal.



Unit 3

HTTP & ASP.NET Core

The relationship between ASP.NET Core, ASP.NET, .NET Core, and .NET Framework. ASP.NET Core runs on both .NET Framework and .NET Core, so it can run cross-platform.

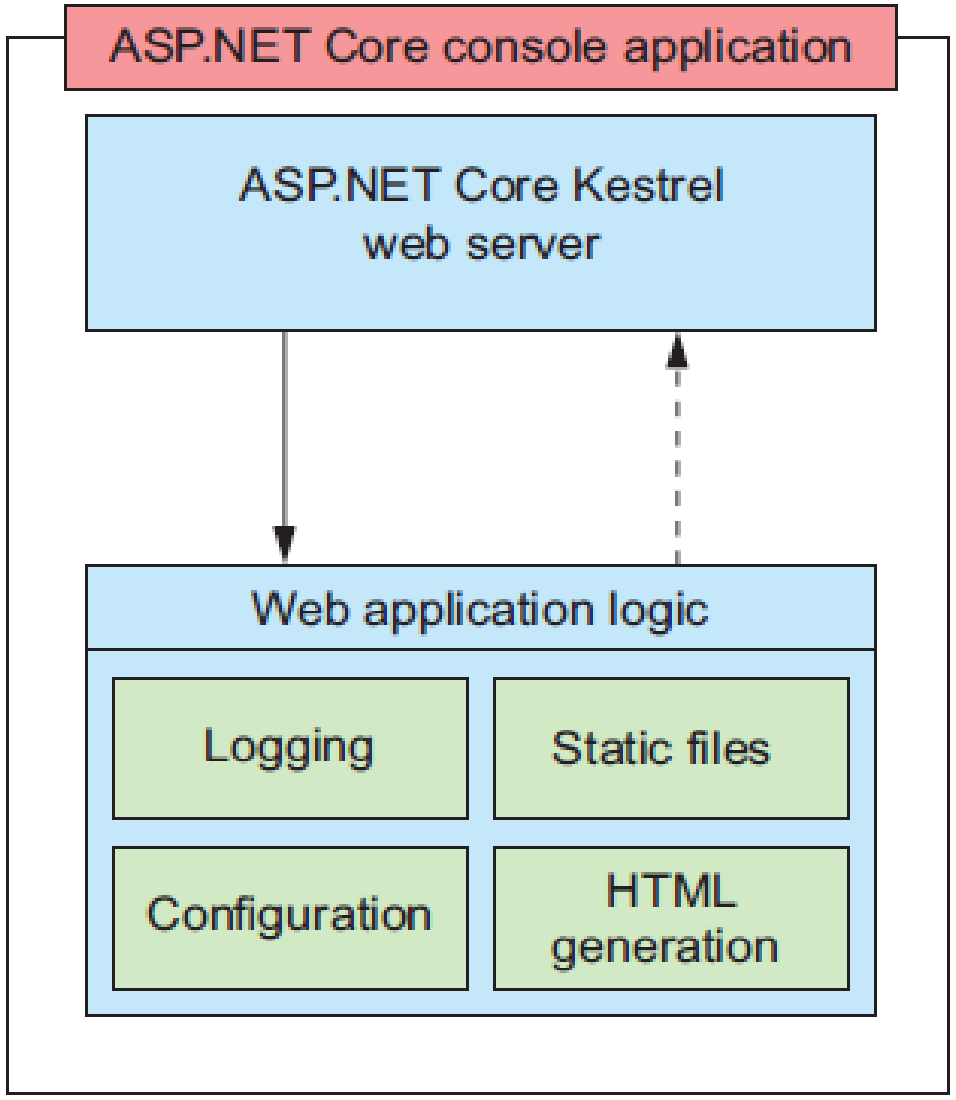


ASP.NET Core application model

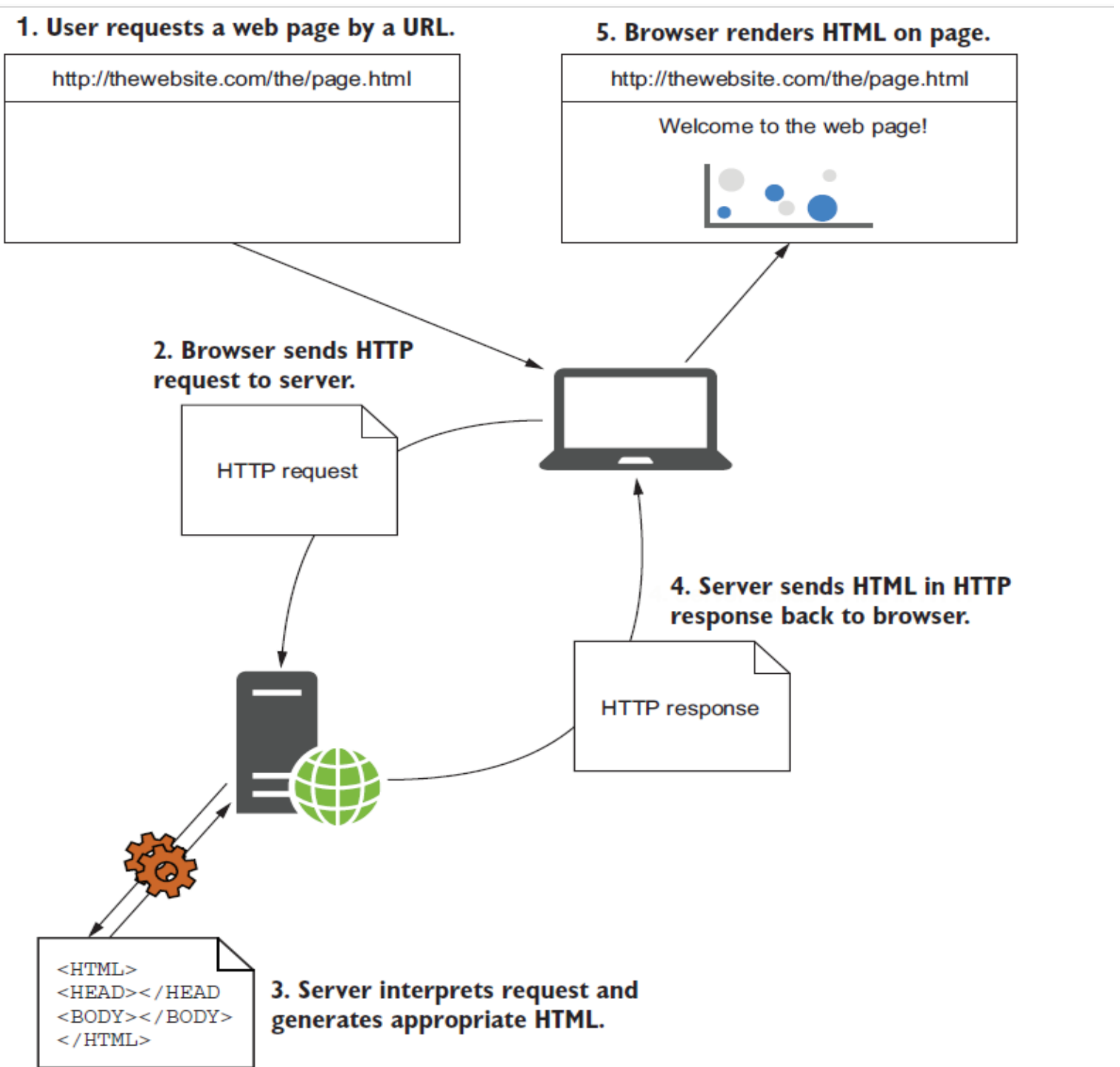
You write a .NET Core console app that starts up an instance of an ASP.NET Core web server.

Microsoft provides, by default, a cross-platform web server called Kestrel.

Your web application logic is run by Kestrel. You'll use various libraries to enable features such as logging and HTML generation as required.



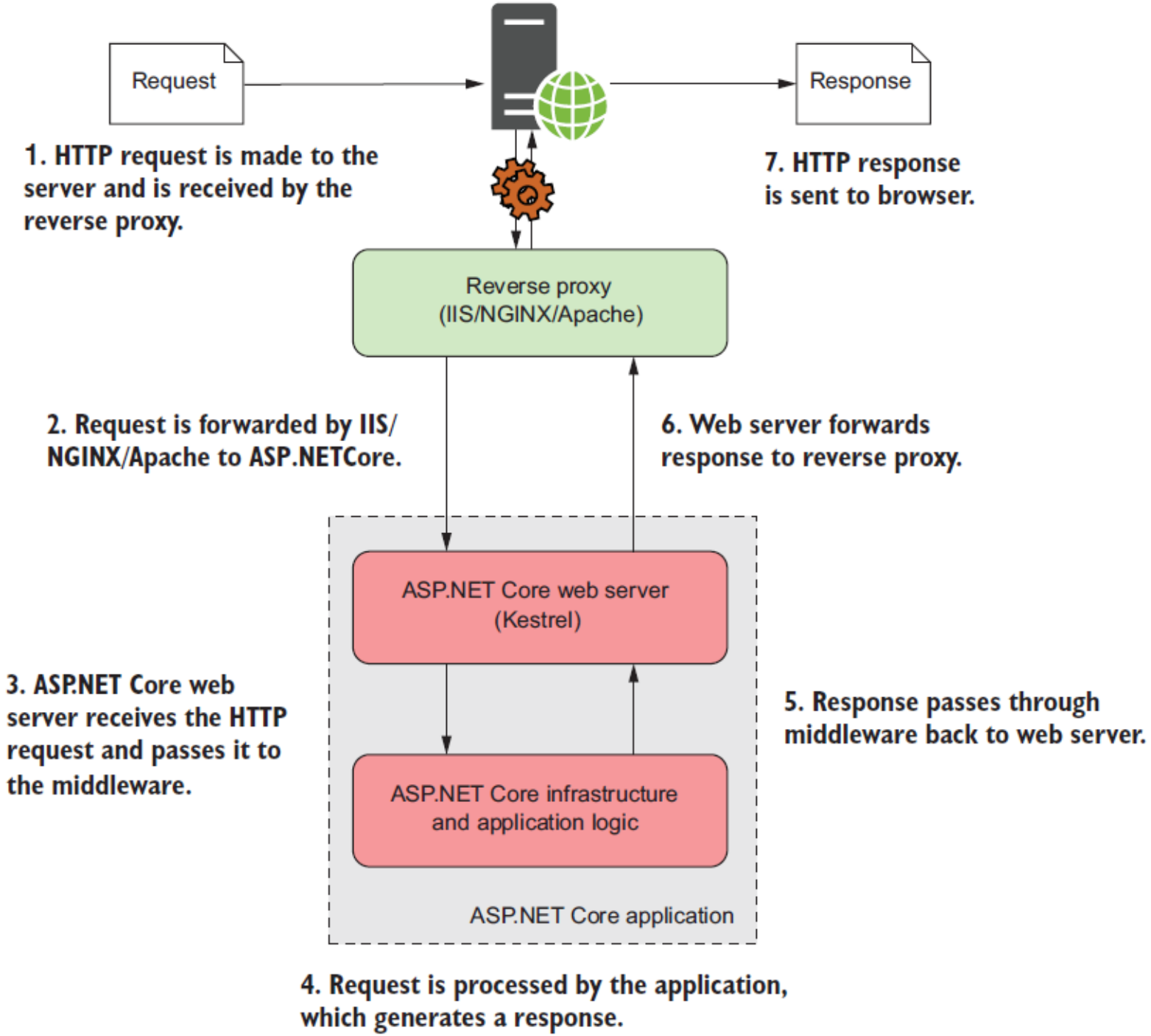
How does an HTTP web request work?



How does an HTTP web request work

- the user starts by requesting a web page, which causes an HTTP request to be sent to the server. The server interprets the request, generates the necessary HTML, and sends it back in an HTTP response. The browser can then display the web page.
- Once the server receives the request, it will check that it makes sense, and if it does, will generate an HTTP response. Depending on the request, this response could be a web page, an image, a JavaScript file, or a simple acknowledgment.
- As soon as the user's browser begins receiving the HTTP response, it can start displaying content on the screen, but the HTML page may also reference other pages and links on the server.

How does ASP.NET Core process a request?



How does ASP.NET Core process a request?

- A request is received from a browser at the reverse proxy, which passes the request to the ASP.NET Core application, which runs a self-hosted web server.
- The web server processes the request and passes it to the body of the application, which generates a response and returns it to the web server. The web server relays this to the reverse proxy, which sends the response to the browser.
- benefit of a reverse proxy is that it can be hardened against potential threats from the public internet. They're often responsible for additional aspects, such as restarting a process that has crashed. Kestrel can stay as a simple HTTP server. Think of it as a simple separation of concerns: Kestrel is concerned with generating HTTP responses; a reverse proxy is concerned with handling the connection to the internet.

Common web application architectures

- monolithic application
- All-in-one applications
- Layered Architecture
- Traditional "N-Layer" architecture applications
- Clean architecture

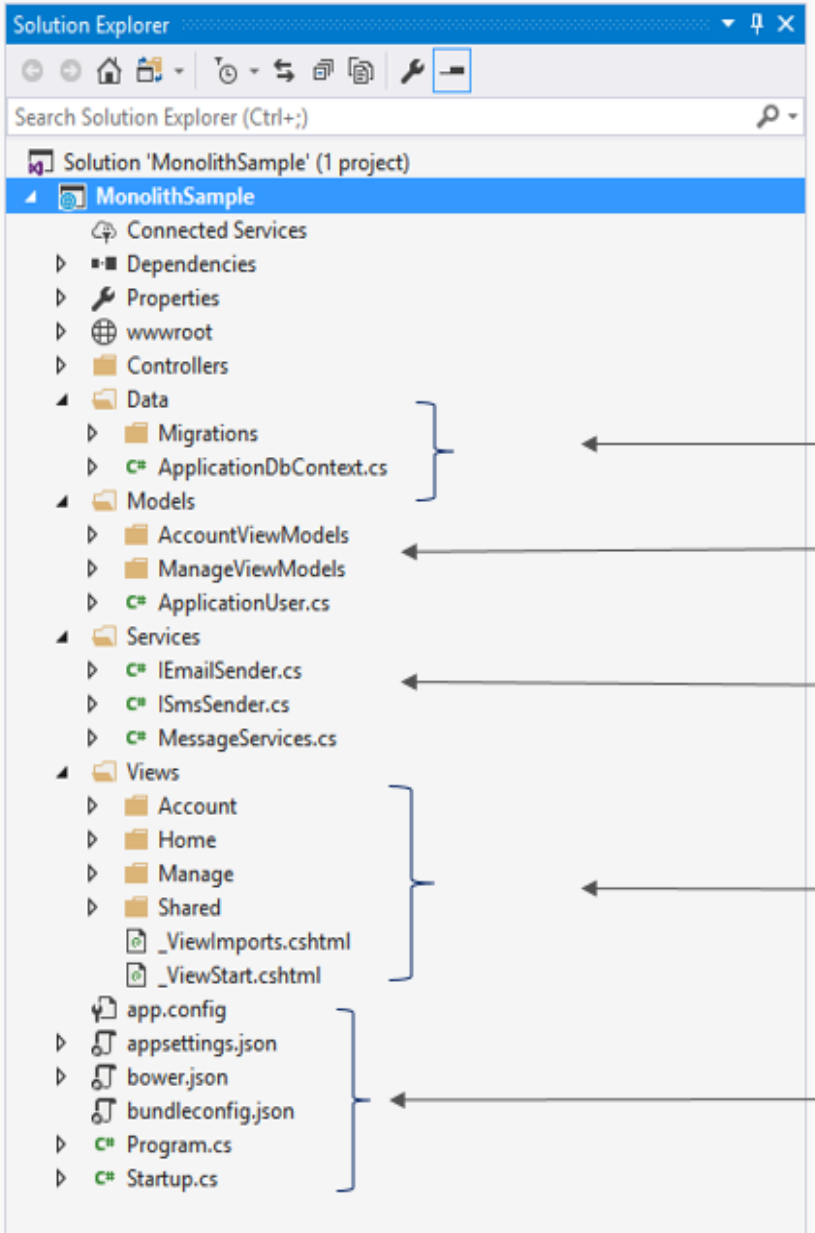
Monolithic Application

- A monolithic application is one that is entirely self-contained, in terms of its behavior.
- It may interact with other services or data stores in the course of performing its operations, but the core of its behavior runs within its own process and the entire application is typically deployed as a single unit.
- If such an application needs to scale horizontally, typically the entire application is duplicated across multiple servers or virtual machines.

All-in-one applications

- The smallest possible number of projects for an application architecture is one. In this architecture, the entire logic of the application is contained in a single project, compiled to a single assembly, and deployed as a single unit.
- A new ASP.NET Core project, whether created in Visual Studio or from the command line, starts out as a simple "all-in-one" monolith. It contains all of the behavior of the application, including presentation, business, and data access logic. In a single project scenario, separation of concerns is achieved through the use of folders. The default template includes separate folders for MVC pattern responsibilities of Models, Views, and Controllers, as well as additional folders for Data and Services. Figure shows the file structure of a single-project app.

VS Solution Structure



Data Access Logic
- EF Migrations
- EF DbContext and model design

UI Models

Application Services (interfaces and implementations)

Presentation Logic

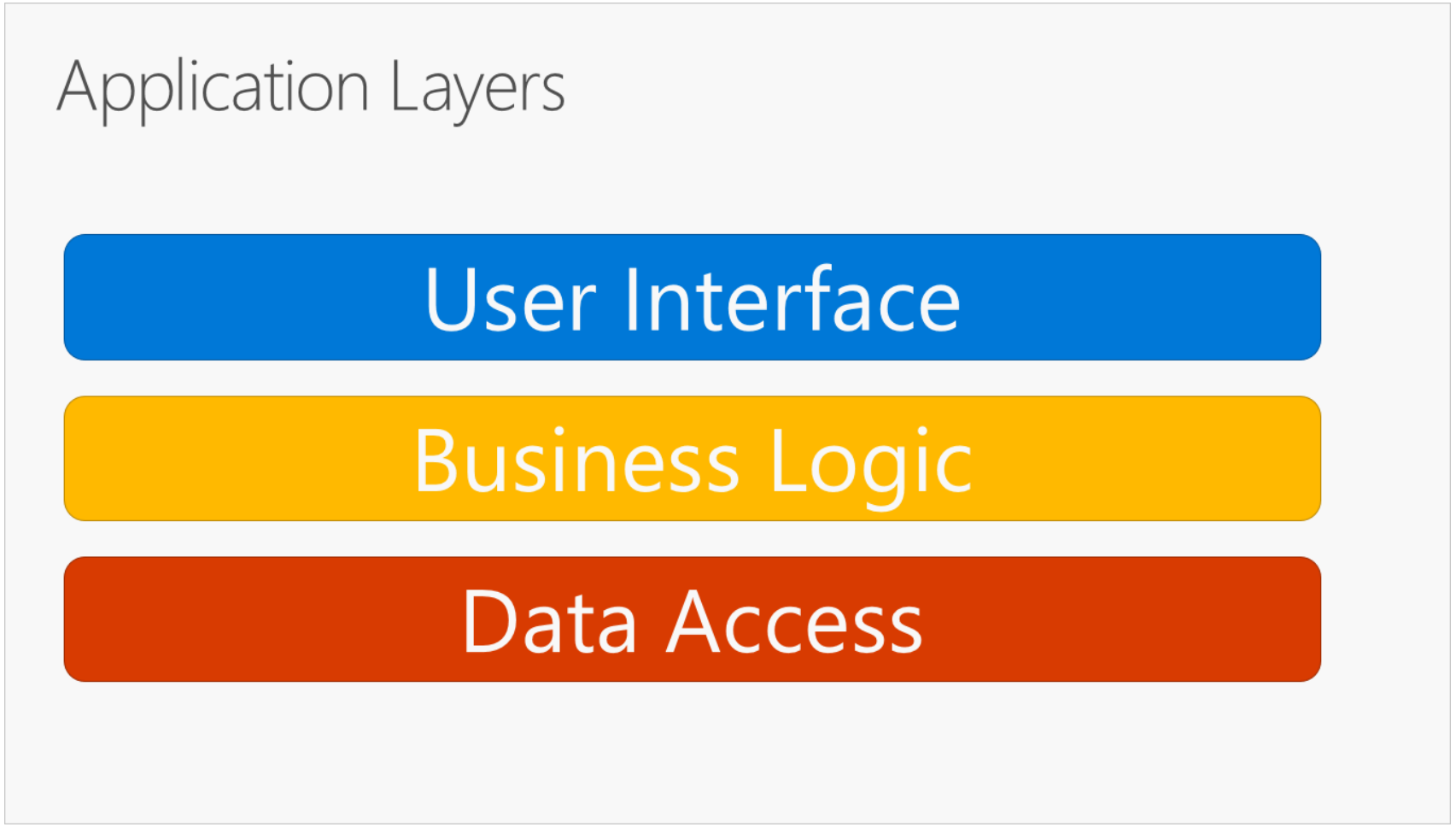
Application Entry Point and Configuration

- Presentation details should be limited as much as possible to the Views folder, and data access implementation details should be limited to classes kept in the Data folder. Business logic should reside in services and classes within the Models folder.
- Although simple, the single-project monolithic solution has some disadvantages:
  - As the project's size and complexity grows, the number of files and folders will continue to grow as well. User interface (UI) reside in multiple folders, which aren't grouped together alphabetically.
  - Business logic is scattered between the Models and Services folders, and there's no clear indication of which classes in which folders should depend on which others. This lack of organization at the project level frequently leads to spaghetti code.
  - To address these issues, applications often evolve into multi-project solutions, where each project is considered to reside in a particular layer of the application.

Layered Architechture

- As applications grow in complexity, one way to manage that complexity is to break up the application according to its responsibilities or concerns. This follows the separation of concerns principle and can help keep a growing codebase organized so that developers can easily find where certain functionality is implemented.
- Layered architecture offers a number of advantages beyond just code organization, though. By organizing code into layers, common low-level functionality can be reused throughout the application.
- With a layered architecture, applications can enforce restrictions on which layers can communicate with other layers. This helps to achieve encapsulation. When a layer is changed or replaced, only those layers that work with it should be impacted. By limiting which layers depend on which other layers, the impact of changes can be mitigated so that a single change doesn't impact the entire application.

Traditional "N-Layer" architecture applications



Traditional "N-Layer" architecture applications

- These layers are frequently abbreviated as UI, BLL (Business Logic Layer), and DAL (Data Access Layer). Using this architecture, users make requests through the UI layer, which interacts only with the BLL. The BLL, in turn, can call the DAL for data access requests. The UI layer shouldn't make any requests to the DAL directly, nor should it interact with persistence directly through other means. Likewise, the BLL should only interact with persistence by going through the DAL.
- One disadvantage of this traditional layering approach is that compile-time dependencies run from the top to the bottom. That is, the UI layer depends on the BLL, which depends on the DAL. This means that the BLL, which usually holds the most important logic in the application, is dependent on data access implementation details (and often on the existence of a database). Testing business logic in such an architecture is often difficult, requiring a test database. The dependency inversion principle can be used to address this issue.

VS Solution Structure

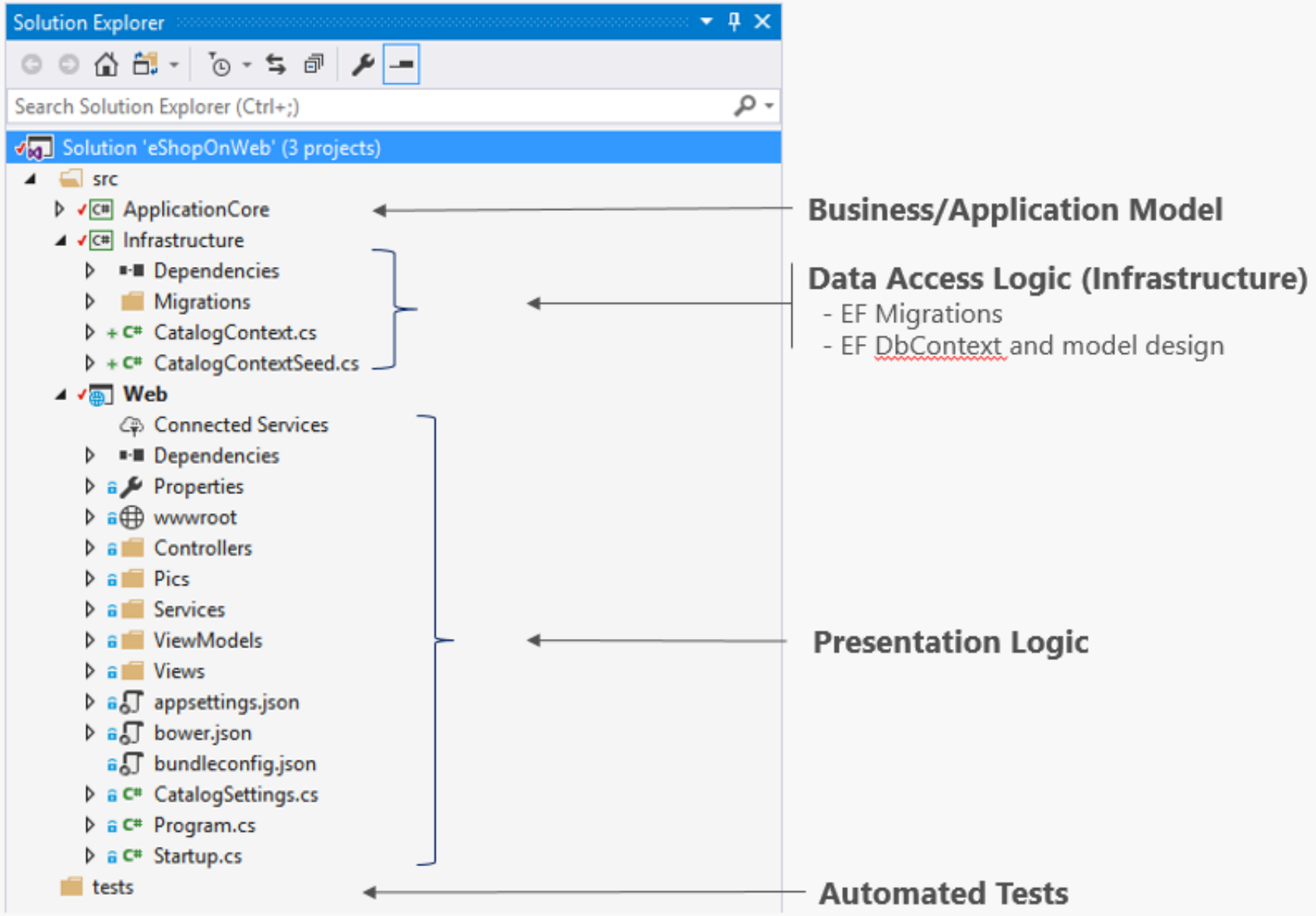


Figure shows an example solution, breaking the application into three projects by responsibility (or layer).

Clean architecture

- Applications that follow the Dependency Inversion Principle as well as the Domain-Driven Design (DDD) principles tend to arrive at a similar architecture. It's been cited as the Onion Architecture or Clean Architecture.
- Clean architecture puts the business logic and application model at the center of the application. Instead of having business logic depend on data access or other infrastructure concerns, this dependency is inverted: infrastructure and implementation details depend on the Application Core.
- This is achieved by defining abstractions, or interfaces, in the Application Core, which are then implemented by types defined in the Infrastructure layer. A common way of visualizing this architecture is to use a series of concentric circles, similar to an onion.

Clean Architecture Layers (Onion view)

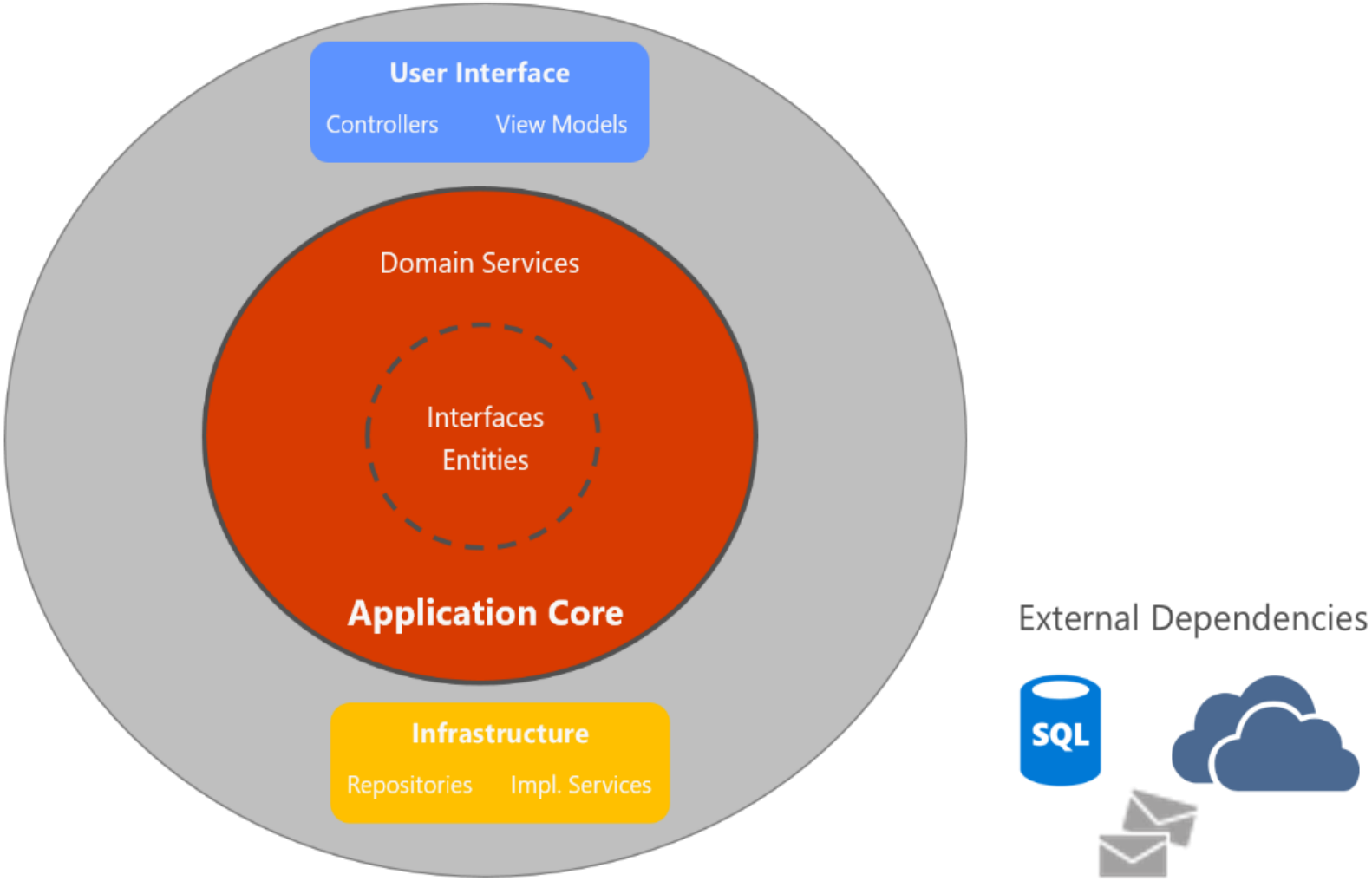


Figure: style of architectural representation.

Clean Architecture

- In the diagram, dependencies flow toward the innermost circle. The Application Core takes its name from its position at the core of this diagram. And you can see on the diagram that the Application Core has no dependencies on other application layers.
- The application's entities and interfaces are at the very center.
- Just outside, but still in the Application Core, are domain services, which typically implement interfaces defined in the inner circle.
- Outside of the Application Core, both the UI and the Infrastructure layers depend on the Application Core, but not on one another (necessarily).

Clean Architecture Layers

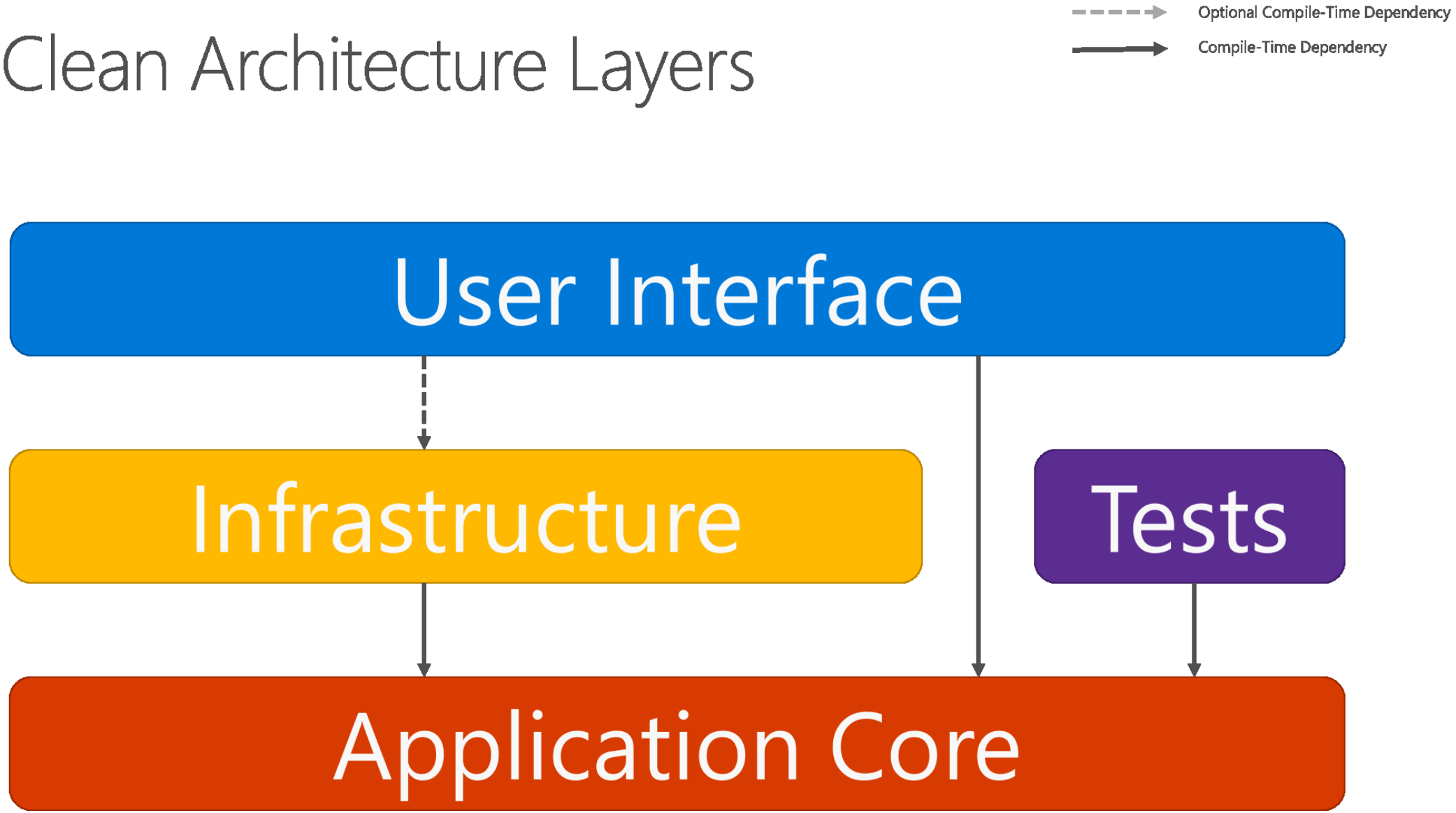


Figure shows a more traditional horizontal layer diagram that better reflects the dependency between the UI and other layers.

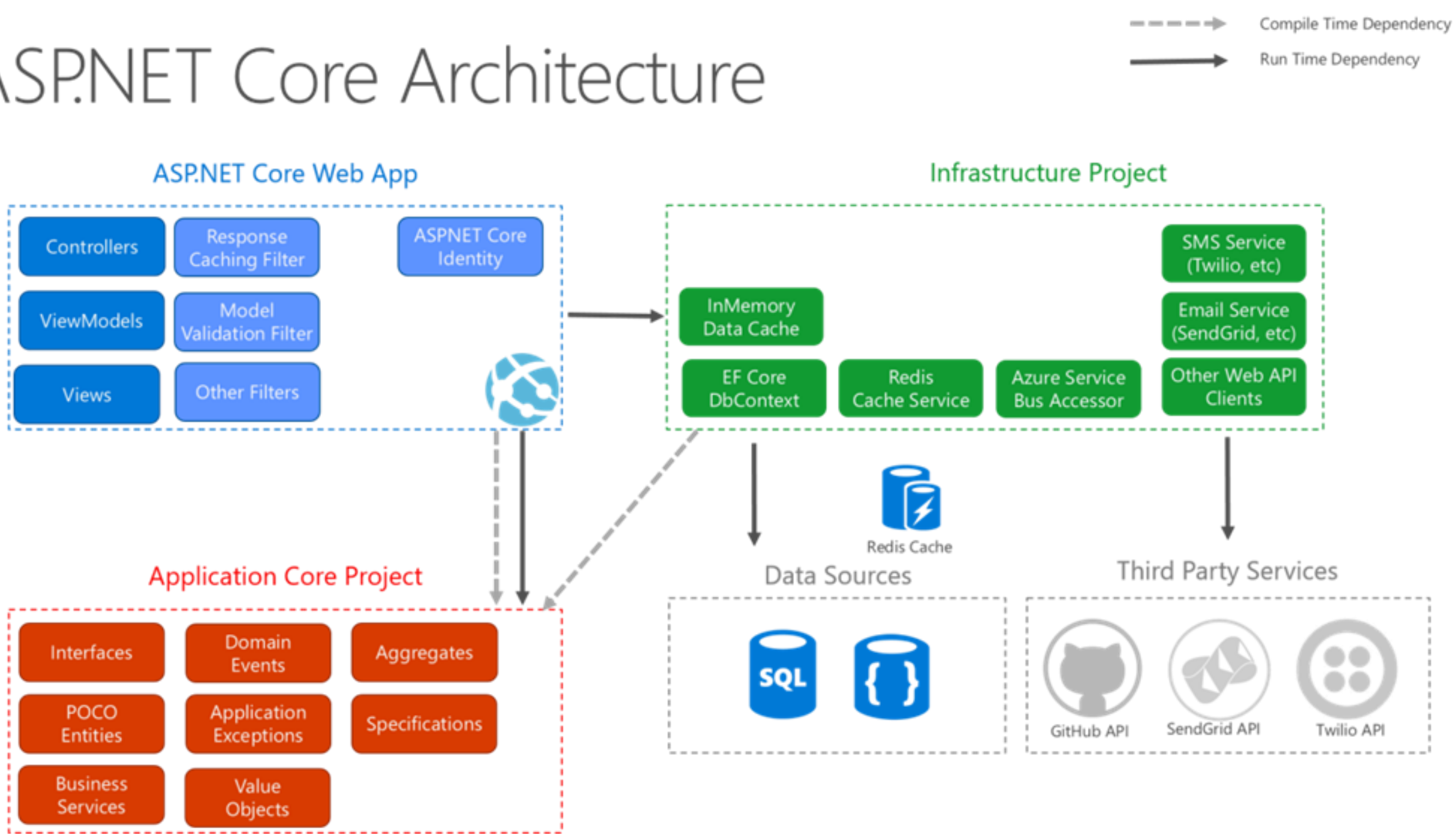
ASP.NET Core Architecture Overview

- The ideology behind ASP.NET Core in general, as the name suggests, is to lay out web logic, infrastructure, and core components from each other in order to provide a more development-friendly environment.
- The concept is somewhat similar to "N" tier/layer architecture, the only difference is that ASP.NET Core defines the layers as the core component of the platform which relieves the developer from redefining it in order to make a solution more modular and reusable.
- What happens in ASP.NET Core is that the main business logic and UI logic are encapsulated in ASP.NET Core Web App layer, while the database access layer, cache services, and web API services are encapsulated in infrastructure layer and common utilities, objects, interfaces and reusable business services are encapsulated as micro-services in application core layer.

ASP.NET Core Architecture Overview

- ASP.NET Core creates necessary pre-defined "N" tier/layers architecture for us developers automatically, which saves our time and effort to worry less about the complexity of necessary "N" tier/architecture of the web project and focus more on the business logic.
- ASP.NET Core that brings the benefit of a pre-built architectural framework that eases out tier deployment of the project along with providing pre-build Single Page Application (SPA) design pattern, Razor Pages (Page based more cleaner MVC model) design pattern, and traditional MVC (View based model) design pattern.
- These design patterns are mostly used in a hybrid manner but can be utilized as an individual-only pattern as well.

ASP.NET Core Architecture



MVC(Model – View - Controller) Design Pattern

- The MVC design has actually been around for a few decades, and it's been used across many different technologies.
- The MVC design pattern is a popular design pattern for the user interface layer of a software application.
- In larger applications, you typically combine a model-view-controller UI layer with other design patterns in the application, like data access patterns and messaging patterns.
- These will all go together to build the full application stack.

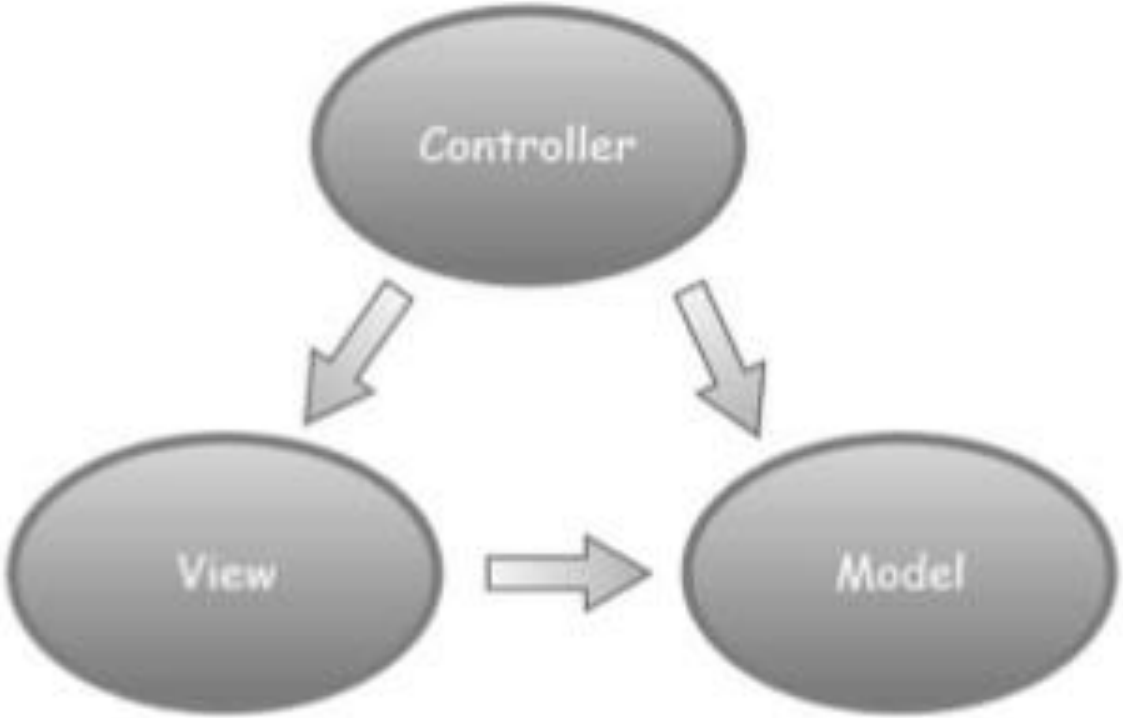
MVC(Model – View - Controller) Design Pattern

- The MVC separates the user interface (UI) of an application into the following three parts −
- **The Model** − A set of classes that describes the data you are working with as well as the business logic.
- **The View** − Defines how the application's UI will be displayed. It is a pure HTML which decides how the UI is going to look like.
- **The Controller** − A set of classes that handles communication from the user, overall application flow, and application-specific logic.

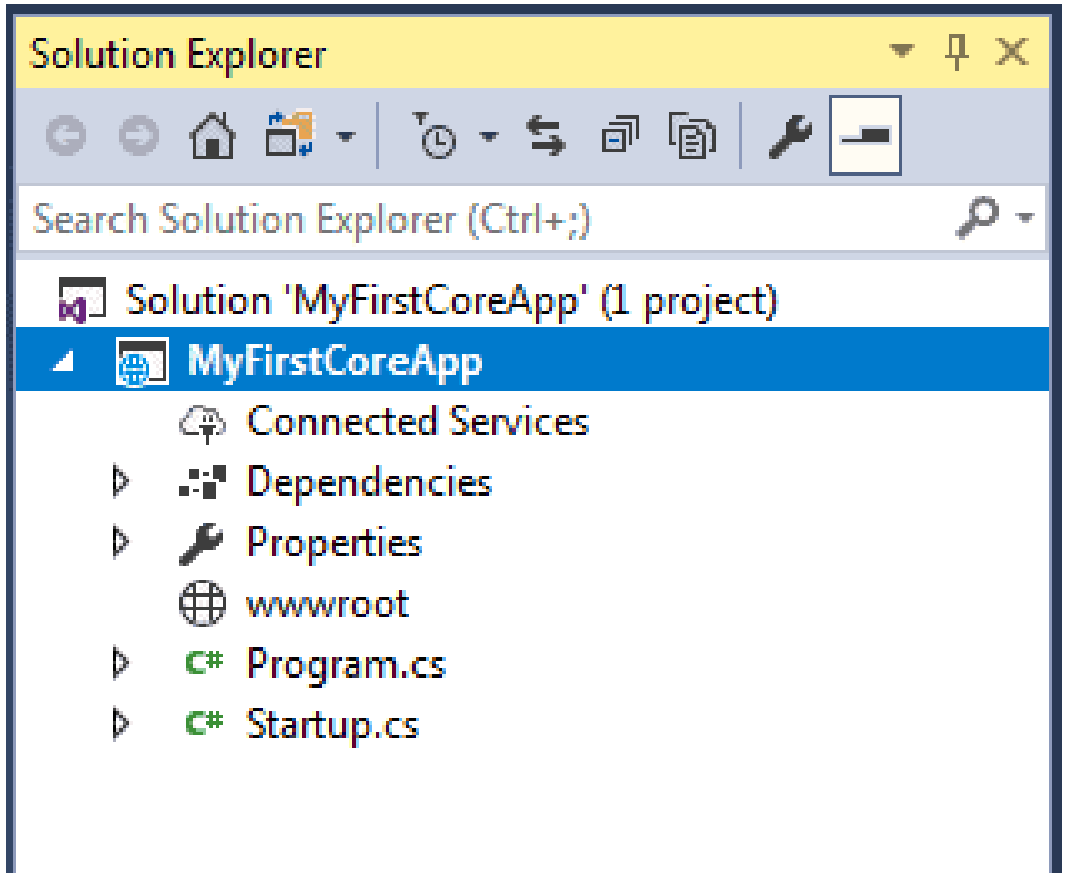
Idea Behind MVC

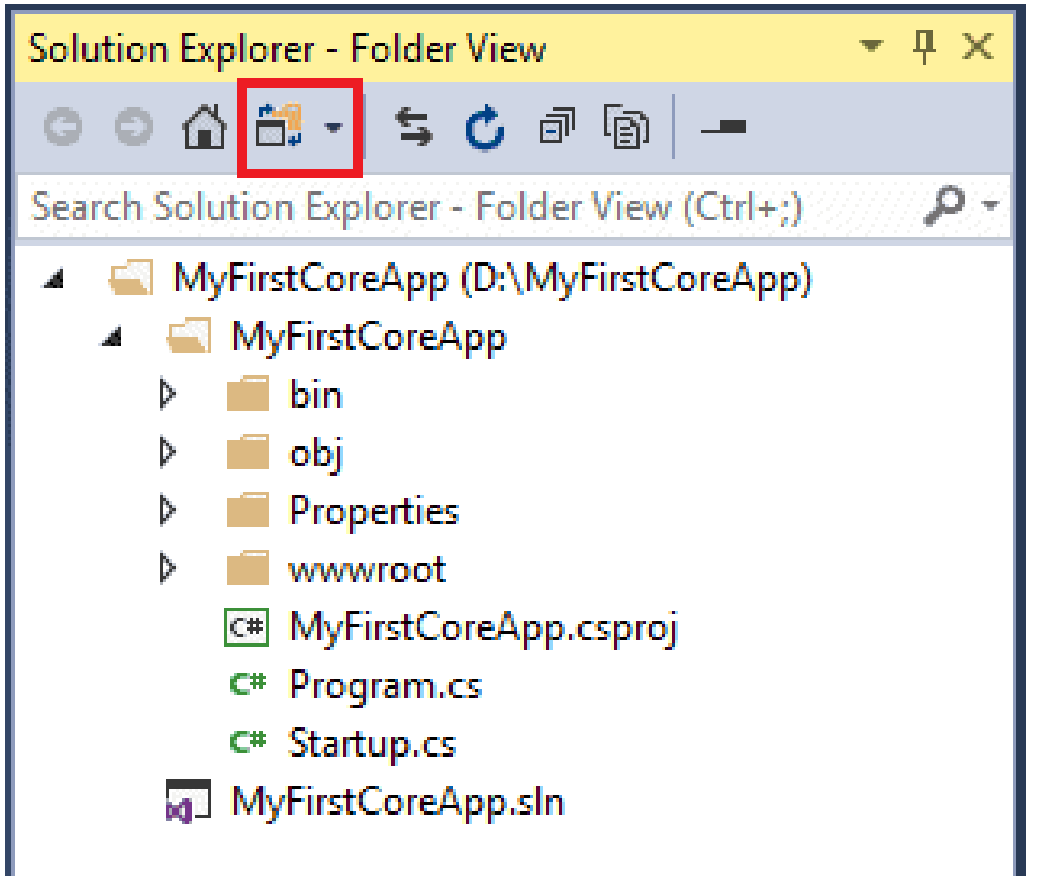
- The idea is that you'll have a component called the view which is solely responsible for rendering this user interface whether it should be HTML or whether it actually should be a UI widget on a desktop application.
- The view talks to a model, and that model contains all the data that the view needs to display.
- In a web application, the view might not have any code associated with it at all.
- It might just have HTML and then some expressions of where to take the pieces of data from the model and plug them into the correct places inside the HTML template that you've built in the view.
- The controller organizes everything. When an HTTP request arrives for an MVC application, the request gets routed to a controller, and then it's up to the controller to talk to either the database, the file system, or a model.

Idea Behind MVC



Projects and Conventions





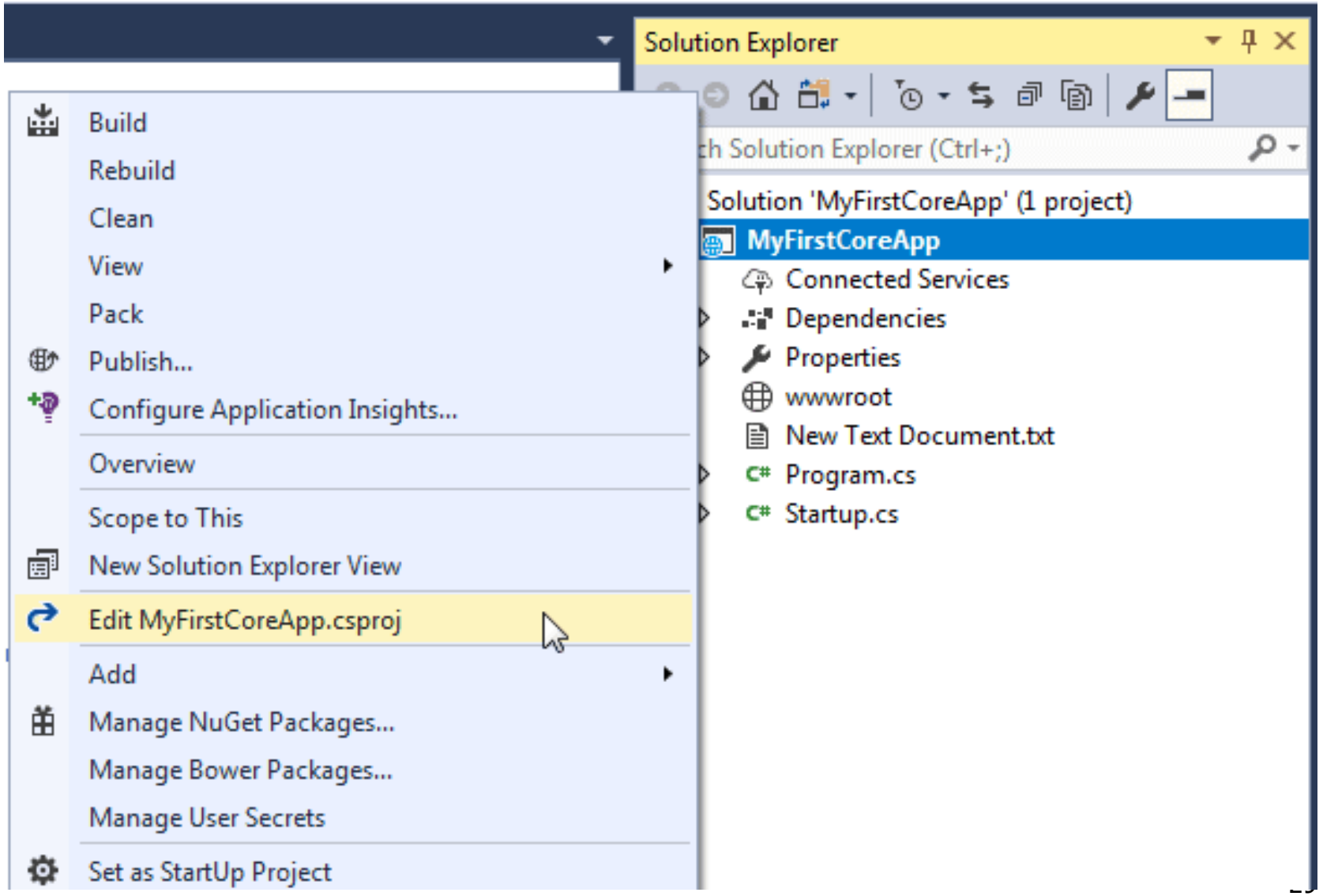
Projects and Conventions

- **.csproj –**

Visual Studio now uses .csproj file to manage projects.

We can edit the .csproj settings by :

- right click on the project
- Select Edit < project-name>.csproj as shown below.



Projects and Conventions

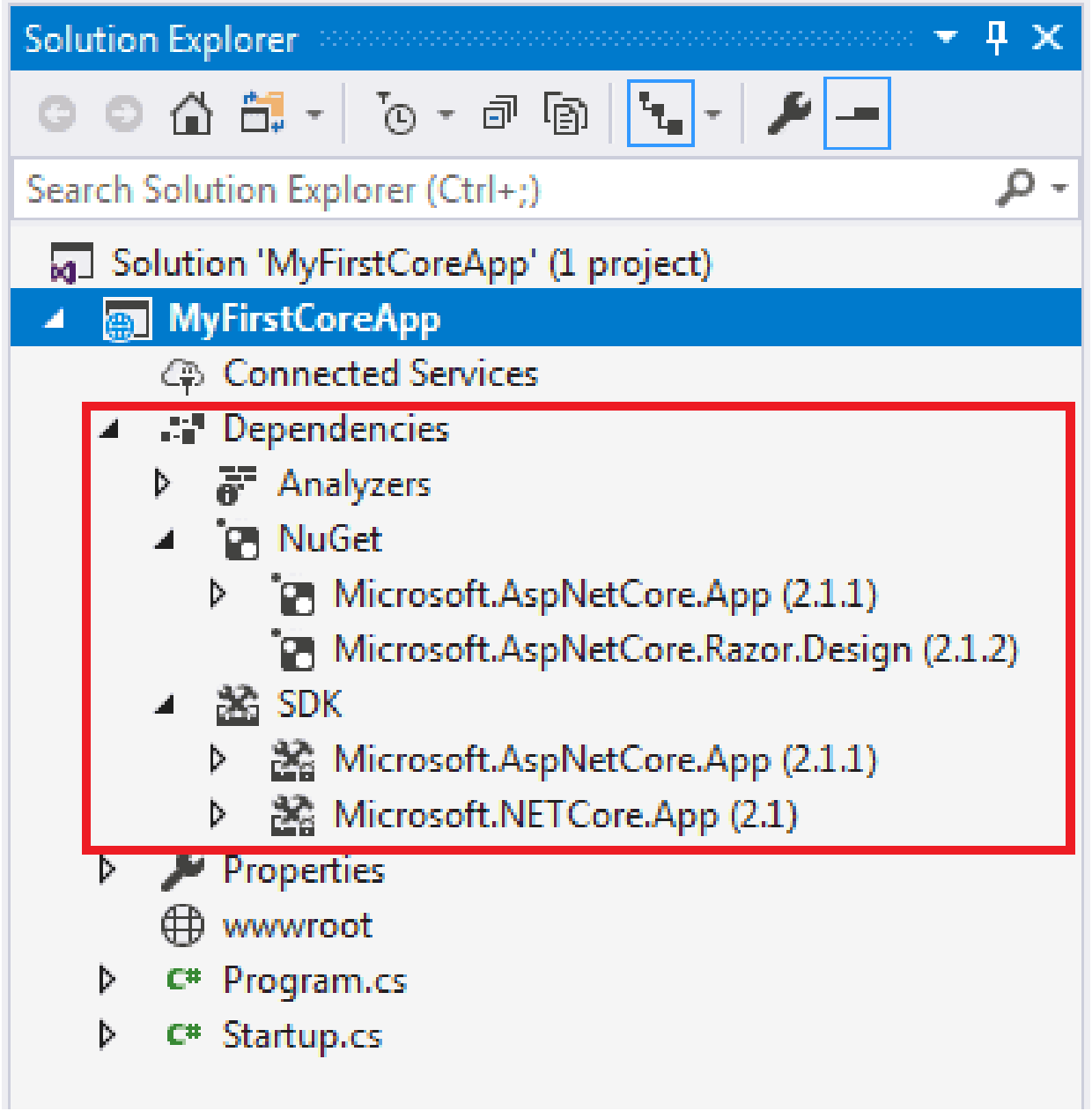
```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All" />
  </ItemGroup>

</Project>

```

- The .csproj for the project looks like above.
- The csproj file includes settings related to targeted .NET Frameworks, project folders, NuGet package references etc.

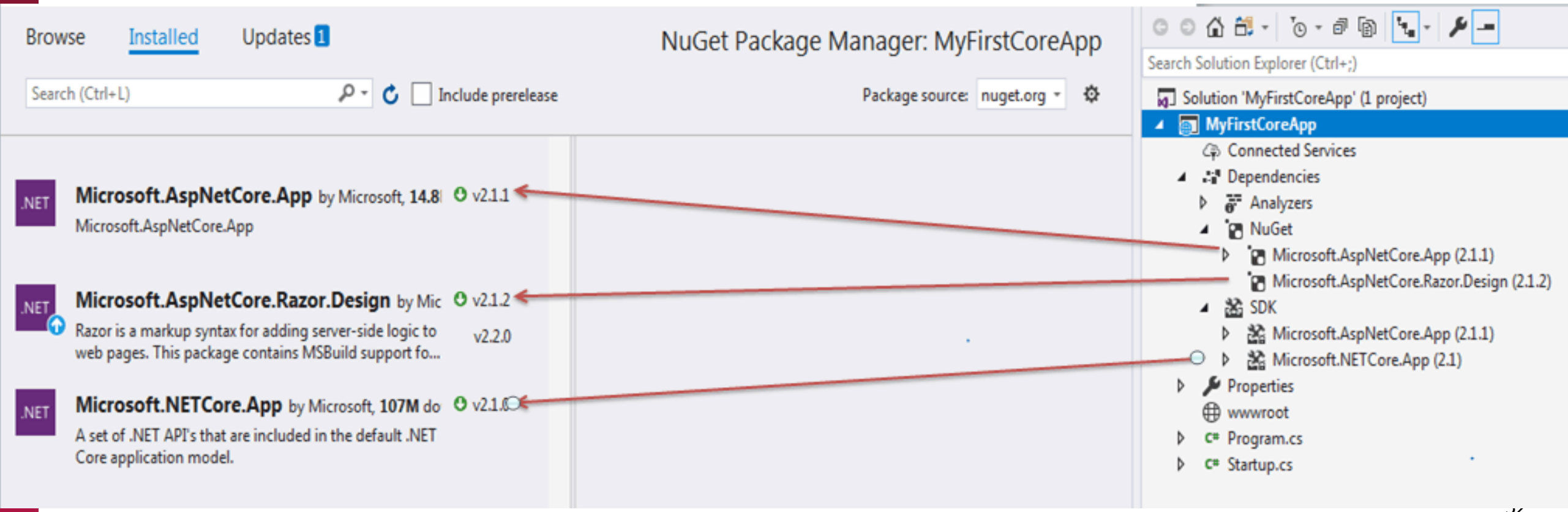
Projects and Conventions

- **Dependencies**

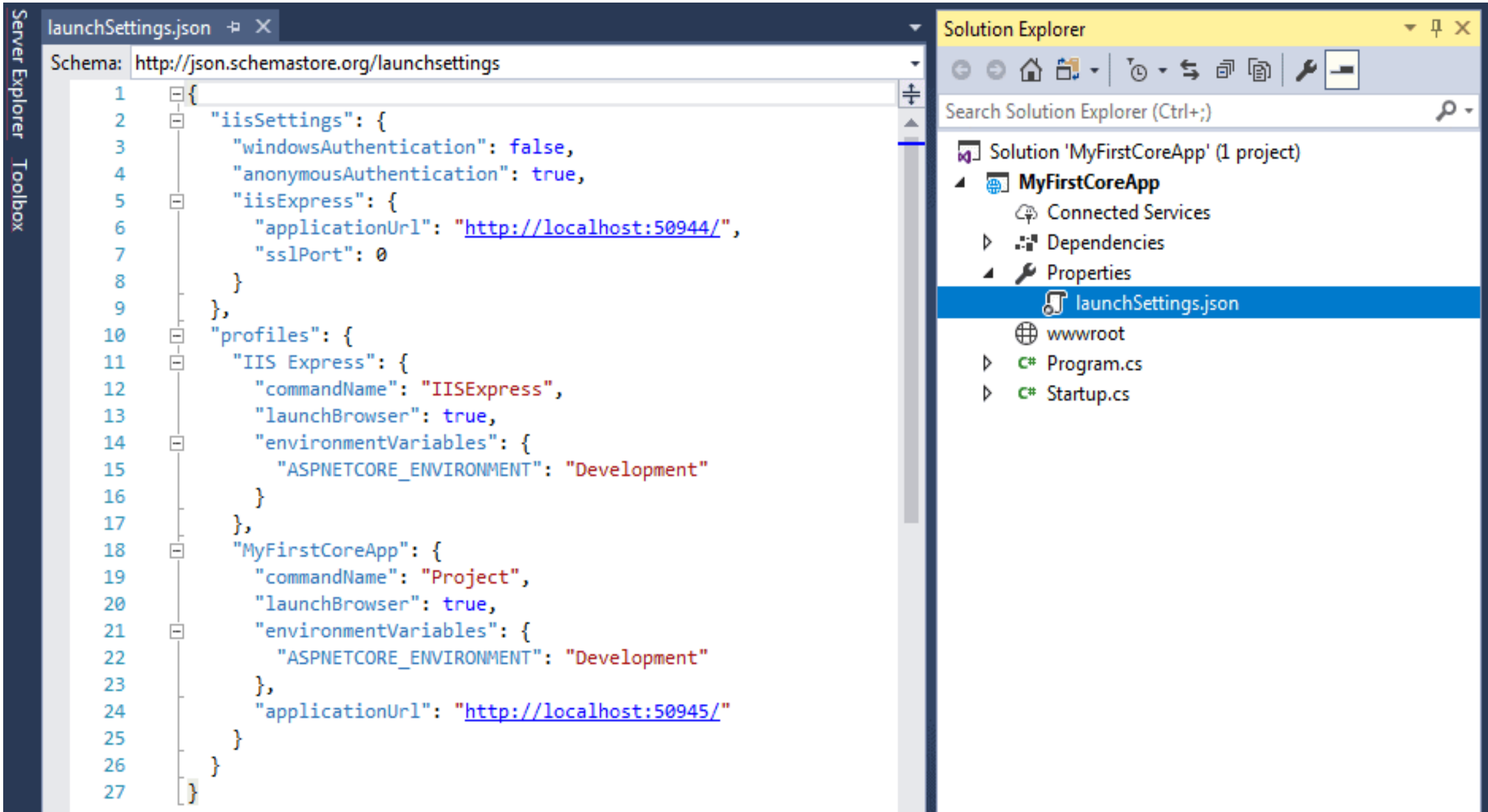
The Dependencies in the ASP.NET Core project contain all the installed server-side NuGet packages, as shown.



- Right click on "Dependencies" and then click "**Manage NuGet Packages..**" to see the installed NuGet packages, as shown below.
- It has installed three packages, Microsoft.AspNetCore.App package is for ASP.NET web application, Microsoft.AspNetCore.Razor.Design package is for Razor engine, and Microsoft.NETCore.App package is for .NET Core API.



Properties : The Properties node includes launchSettings.json file which includes Visual Studio profiles of debug settings. The following is a default launchSettings.json file.



Unit 4

Creating ASP.NET Core MVC Applications

Creating a Web App & Run

- From the Visual Studio, select Create a new project.
- Select ASP.NET Core Web Application and then select Next.
- Name the project as you like or WebApplicationCoreS1
- Choose the location path to save your project.
- Click Create
- Select Web Application(Model-View-Controller), and then select Create.
- Now, To run the App,
  - Select **Ctrl-F5** to run the app in non-debug mode, Or
  - Select IIS Express Button. 

Setting up the Environment

ASP.NET Core wwroot Folder

- By default, the wwwroot folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.
- In ASP.NET Core, 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.
- Generally, we find static files such as JavaScript, CSS, Images, library scripts etc. in the wwwroot folder
- You can access static files with base URL and file name. For example, for css folder, we can access via http://localhost:<port>/css/app.css

Setting up the Environment

ASP.NET Core – Program.cs Class

- ASP.NET Core web application project starts executing from the entry point - public static void Main() in Program class.

ASP.NET Core – Startup.cs Class

- It is like Global.asax in the traditional .NET application. As the name suggests, it is executed first when the application starts.
- The startup class can be configured at the time of configuring the host in the Main() method of Program class.

Add a controller

- In Solution Explorer, right-click Controllers > Add > Controller
- In the Add Scaffold dialog box, select Controller Class – Empty
- In the Add Empty MVC Controller dialog, enter HelloWorldController and select Add.
- Replace the contents of Controllers/HelloWorldController.cs

```
public string Index() {
        return "This is my default action...";
}
public string Welcome() {
        return "This is the Welcome action method...";
}
```

Add a controller

- MVC invokes controller classes (and the action methods within them) depending on the incoming URL.
- The default URL routing logic used by MVC uses a format like this:

  /[Controller]/[ActionName]/[Parameters]

- The routing format is set in the Configure method in Startup.cs file.

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

- Run your app & Check with these url in your browser
  - https://localhost:{PORT}/HelloWorld
  - https://localhost:{PORT}/HelloWorld/Index
  - https://localhost:{PORT}/HelloWorld/Welcome
- Make changes for Welcome Method like this:

```
// GET: /HelloWorld/Welcome/
// Requires using System.Text.Encodings.Web;
public string Welcome(string name, int numTimes = 1)
{
    return HtmlEncoder.Default.Encode($"Hello {name}, NumTimes is: {numTimes}");
}
```

- Check on your browser with these:
  - https://localhost:{PORT}/HelloWorld/Welcome?name=AAA&numtimes=4

- Make changes again for Welcome Method with following code

```
public string Welcome(string name, int ID = 1) {
    return HtmlEncoder.Default.Encode($"Hello {name},  ID is: {ID}");
}
```

- Check on your browser with these:
  - http://localhost:{PORT}/HelloWorld/Welcome/3?name=AAA
- Here, the third URL segment matched the route parameter id. The Welcome method contains a parameter id that matched the URL template in the MapControllerRoute method in Startup.cs file. The trailing ? (in id?) indicates the id parameter is optional.

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

Add a view

- In your Project, Right click on the *Views* folder, and then **Add > New Folder** and name the folder *HelloWorld*.
- Right click on the *Views/HelloWorld* folder, and then **Add > New Item**.
- In the **Add New Item** dialog
  - In the search box in the upper-right, enter *view*
  - Select **Razor View**
  - Keep the **Name** box value, *Index.cshtml*.
  - Select **Add**
- Replace the contents of the Views/HelloWorld/Index.cshtml Razor view file with the following

```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

Your Controller and View

```
public class HelloWorldController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }
```

Index.cshtml

```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

- Navigate to https://localhost:{PORT}/HelloWorld

Change views and layout pages

- In page, Select the menu links (WebApplicationCoreS1, Home, Privacy)
- Each page shows the same menu layout.
- The menu layout is implemented in the Views/Shared/_Layout.cshtml file.
- Open the Views/Shared/_Layout.cshtml file.
- Layout templates allow you to specify the HTML container layout of your site in one place and then apply it across multiple pages in your site.
- Find the @RenderBody() line. RenderBody is a placeholder where all the view-specific pages you create show up, wrapped in the layout page.
- For example, if you select the Privacy link, the Views/Home/Privacy.cshtml view is rendered inside the RenderBody method.

Change the title, footer, and menu link in the layout file

- Replace the content of the Views/Shared/_Layout.cshtml file with the following markup. The changes are highlighted:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplicationCoreS1</title>


<div class="container">
    <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">WebApplicationCoreS1</

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2020 - WebApplicationCoreS1 - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy<
```

Passing Data from the Controller to the View

- Controllers are responsible for providing the data required in order for a view template to render a response.
- In HelloWorldController.cs, change the Welcome method to add a Message and NumTimes value to the ViewData dictionary.
- The ViewData dictionary is a dynamic object, which means any type can be used; the ViewData object has no defined properties until you put something inside it. The MVC model binding system automatically maps the named parameters (name and numTimes) from the query string in the address bar to parameters in your method.
- Ex looks like :

Passing Data from the Controller to the View

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

    public IActionResult Welcome(string name, int numTimes = 1)
    {
        ViewData["Message"] = "Hello " + name;
        ViewData["NumTimes"] = numTimes;

        return View();
    }
}
```

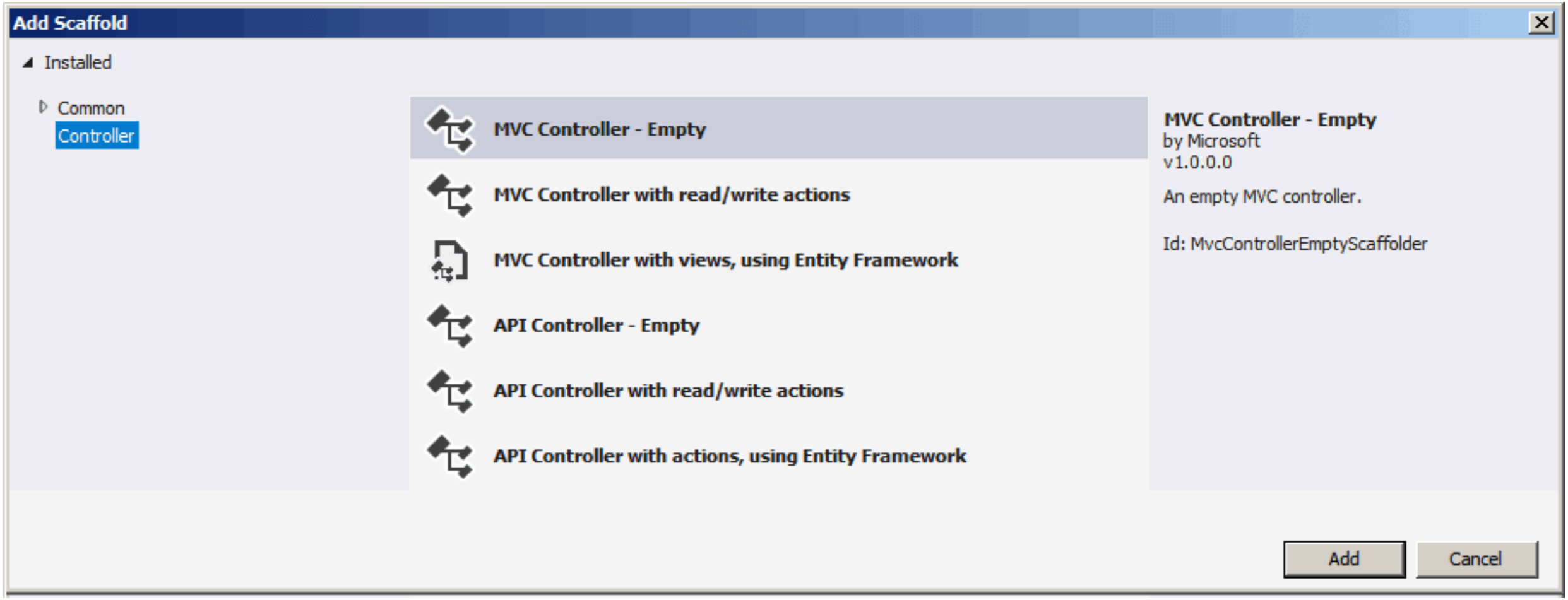
Passing Data from the Controller to the View

```
@{
    ViewData["Title"] = "Welcome";
}

<h2>Welcome</h2>

<ul>
    @for (int i = 0; i < (int)ViewData["NumTimes"]; i++)
    {
        <li>@ViewData["Message"]</li>
    }
</ul>
```

Controllers Responsibities

- Controllers are usually placed in a folder called "Controllers", directly in the root of your MVC project.
- They are usually named based on their purpose, with the word "Controller" as a suffix.
- The Controller has three major responsibilities
  - Handle the Request from the User
  - Build a Model – Controller Action method executes the application logic and builds a model
  - Send the Response – it returns the Result in HTML/File/JSON/XML or other format as requested by user.

Controller Scaffolding Option



Controller Scaffolding Option

- Both the MVC and API Controller inherits from the same Controller class and there is not much difference between them, except that API Controller is expected to return the data in serialized format to the client.
- Further, we have three options under both types of controllers.
  - Empty
  - With Read/Write Actions
  - With Views, using entity framework

Actions

- Controller is just a regular .NET class, it can have fields, properties and methods.
- Methods of a Controller class is referred to as **actions** - a method usually corresponds to an action in your application, which then returns something to the browser/user.
- All public methods on a Controller class is considered an Action.
- For instance, the browser might request a URL like /Products/Details/1 and then you want your ProductsController to handle this request with a method/action called Details.

When creating Action Method, points to remember

- Action methods Must be a public method
- The Action method cannot be a Static method or an Extension method.
- The Constructor, getter or setter cannot be used.
- Inherited methods cannot be used as the Action method.
- Action methods cannot contain ref or out parameters.
- Action Methods cannot contain the attribute [NonAction].
- Action methods cannot be overloaded

Actions Verbs

- To gain more control of how your actions are called, you can decorate them with the so-called Action Verbs.
- an action can be accessed using all possible HTTP methods (the most common ones are GET and POST)
- Edit action can be accessed with a GET request.

```
[HttpGet]
public IActionResult Edit()
{
  return View();
}
```

Actions Verbs

```
[HttpGet]
public IActionResult Edit()
{
  return Content("Edit");
}
[HttpPost]
public IActionResult Edit(Product product)
{
  product.Save();
  return Content("Product Updated!");
}
```

Actions Result Types

- When the Action (method) finishes it work, it will usually return something, as IActionResult interface
- Some list of Action Result are:
  - Content() - returns specified string as plain text
  - View() - returns a View to the client
  - PartialView() - returns a Partial View to the client
  - File() - returns the content of a specified file to the client
  - Json() - returns a JSON response to the client
  - Redirect() and RedirectPermanent() - returns a redirect response to the browser (temporary or permanent), redirecting the user to another URL
  - StatusCode() - returns a custom status code to the client

Actions Result - Ex

- A common use case for this is to return either a View or a piece of content if the requested content is found, or a 404 (Page not Found) error if its not found. It could look like this:

```
public IActionResult Details(int id)
{
    Product product = GetProduct(id);
    if (product != null)
        return View(product);
    return NotFound();
}
```

Rendering HTML with Views

- In MVC pattern, the view handles the app's data presentation and user interaction.
- A view is an HTML template with embedded Razor markup. Razor markup is code that interacts with HTML markup to produce a webpage that's sent to the client
- In ASP.NET Core MVC, views are .cshtml files that use the C# programming language in Razor markup. Usually, view files are grouped into folders named for each of the app's controllers. The folders are stored in a Views folder at the root of the app

Rendering HTML with Views

- The Home controller is represented by a Home folder inside the Views folder. The Home folder contains the views for the About, Contact, and Index (homepage) webpages. When a user requests one of these three webpages, controller actions in the Home controller determine which of the three views is used to build and return a webpage to the user.
- Use layouts to provide consistent webpage sections and reduce code repetition. Layouts often contain the header, navigation and menu elements, and the footer. The header and footer usually contain boilerplate markup for many metadata elements and links to script and style assets. Layouts help you avoid this boilerplate markup in your views.

Creating a View

- To create a view, add a new file and give it the same name as its associated controller action with the .cshtml file extension.
- For Ex - For About action in the Home controller, create an About.cshtml file in the Views/Home folder:

```
@{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"].</h2>
<h3>@ViewData["Message"]</h3>
<p>Use this area to provide additional information.</p>
```

Creating a View

- Razor markup starts with the @ symbol.
- Your can write C# code within Razor code blocks set off by curly braces ({ ... }).
- You can display values within HTML by simply referencing the value with the @ symbol. See the contents of the <h2> and <h3> elements above.

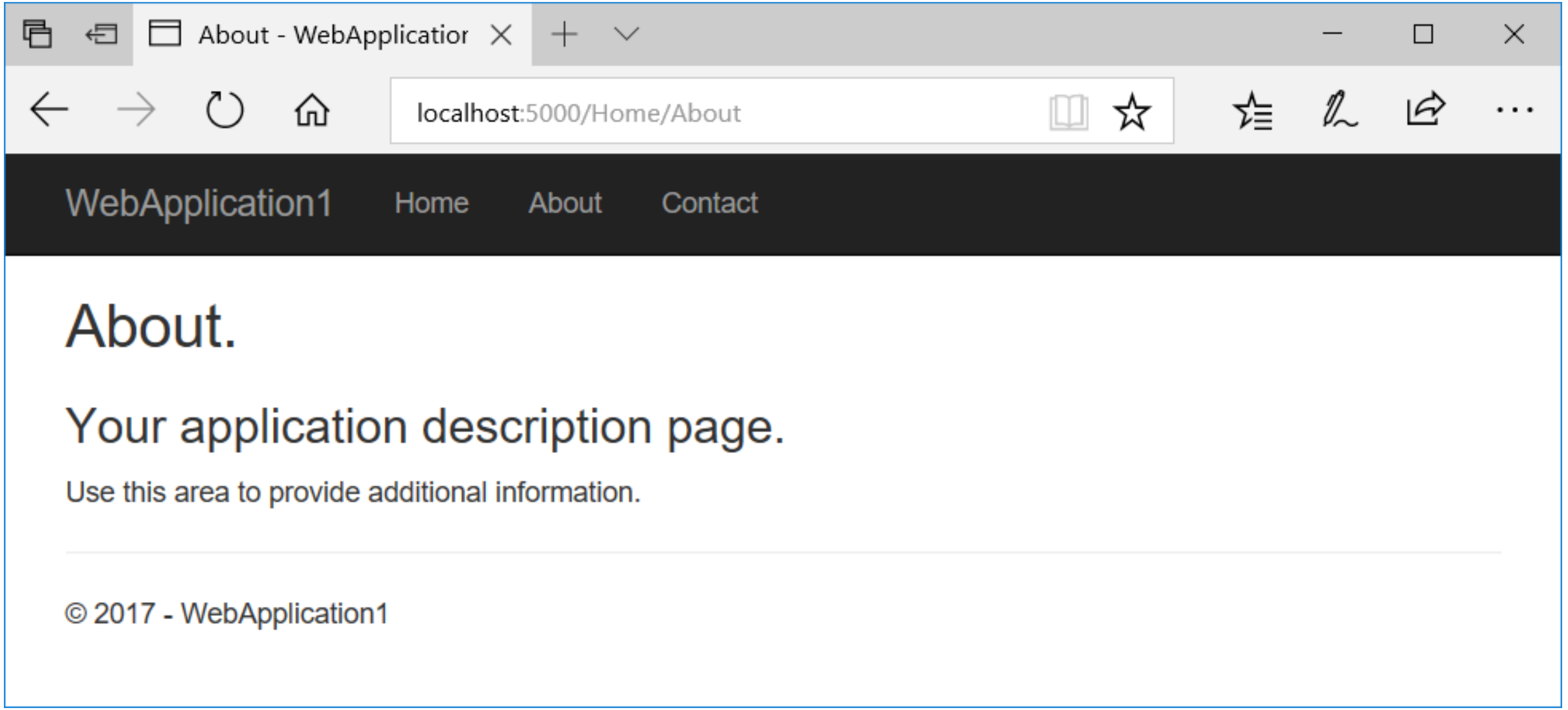
How Controllers Specify Views

- Views are typically returned from actions as a ViewResult, which is a type of ActionResult.

HomeController.cs

```
public IActionResult About()
{
        ViewData["Message"] = "Your application description page.";
        return View();
}
```

How Controllers Specify Views



Passing Data to Views: ViewData & ViewBag

- Views have access to a weakly typed(loosely typed) collection of data. You can use these for passing small amounts of data in and out of controllers and views
- The ViewData property is a dictionary of weakly typed objects.
- The ViewBag property is a wrapper around ViewData that provides dynamic properties for the underlying ViewData collection.
- ViewData and ViewBag are dynamically resolved at runtime.
- ViewData is a ViewDataDictionary object accessed through string keys

Ex - ViewData

```
// HomeController.cs
public class HomeController : Controller {
        public IActionResult About() {
                ViewData["Message"] = "Your application description page.";
                return View();
        }
}
// About.cshtml
    @{
        ViewData["Title"] = "About";
    }
    <h3>@ViewData["Message"]</h3>
```

Ex - ViewBag

```
// HomeController.cs
public class HomeController : Controller {
        public IActionResult SomeAction() {
                ViewBag.Greeting = “Hello";
                return View();
        }
}
//SomeAction.cshtml
    @{
        ViewData["Title"] = “My Title";
    }
    <h3>@ViewBag.Greeting</h3>
```

Razor Syntax

- The biggest advantage of the Razor is the fact that you can mix client-side markup (HTML) with server-side code (e.g C# or VB.NET), without having to explicitly jump in and out of the two syntax types.
- In Razor, you can reference server-side variables etc. by simply prefixing it with an at-character (@).

```
<p>Hello, the current date is: @DateTime.Now.ToString()</p>
```

Ex - Razor & HTML Encoding

```
@{
    var helloWorld = "<b>Hello, world!</b>";
}
<p>@helloWorld</p>
<p>@Html.Raw(helloWorld)</p>
```

Ex – Razor Explicit Expressionsc

```
@{
    var name = "John Doe";
}
Hello, @(name.Substring(0,4)).
Your age is: <b>@(37 + 5).</b>
```

Ex – Multi-statement Razor blocks

```
@{
    var sum = 32 + 10;
    var greeting = "Hello, world!";
    var text = "";
    for(int i = 0; i <3; i++)
    {
        text += greeting + " The result is: " + sum + "\n";
    }
}
<h2>CodeBlocks</h2>
Text: @text
```

Razor Server-side Comments

- Sometimes you may want to leave comments in your code, or comment out lines of code temporarily to test things.

```
@*
    Here's a Razor server-side comment
    It won't be rendered to the browser
*@
```

Razor Server-side Comments

- If you're inside a Razor code block, you can even use the regular C# style comments:

```
@{
    @*
        Here's a Razor server-side comment
    *@
    // C# style single-line comment
    /*
    C# style multiline comment
    It can span multiple lines
    */
}
```

Razor Syntax – Variables and Expressions

```
@{
    string helloWorldMsg = "Good day";
    if(DateTime.Now.Hour >17){
        helloWorldMsg = "Good evening";
    }
    helloWorldMsg += ", world!";
    helloWorldMsg = helloWorldMsg.ToUpper();
}
<div> @helloWorldMsg </div>
```

Razor Syntax – The if-else statement

```
@if(DateTime.Now.Year >= 2042)
{
    <span>The year 2042 has finally arrived!</span>
}
else
{
    <span>We're still waiting for the year of 2042...</span>
}
```

Razor Syntax – Loop

```
@{
    List<string> names = new List<string>() {
    "VB.NET", "C#", "Java"
    };
}
<ul>
   @for (int i = 0; i < names.Count; i++)
   {
       <li>@names[i]</li>
   }
</ul>
<ul>
@foreach (string name in names)
{
    <li>@name</li>
}
</ul>
```

Understanding Tag Helpers

- Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files. Tag helpers are a new feature and similar to HTML helpers, which help us render HTML.

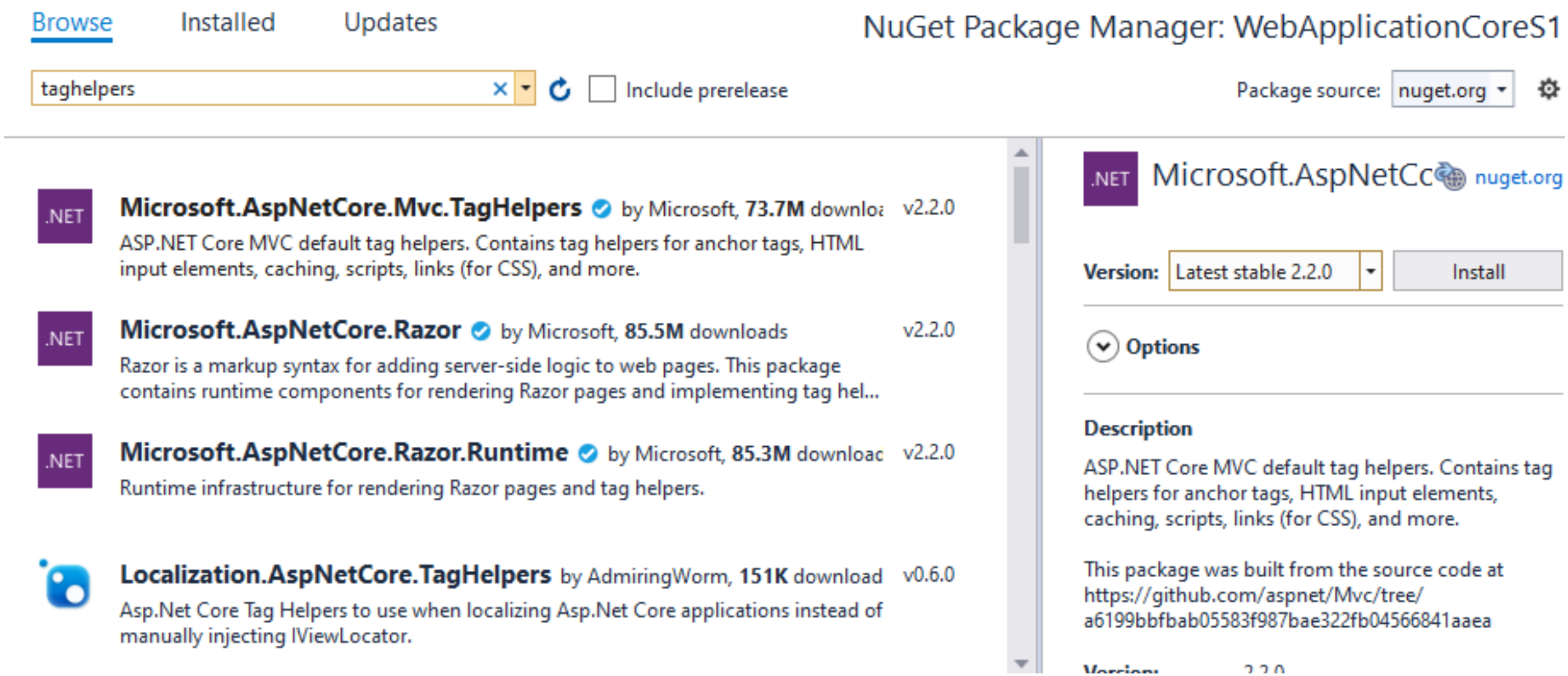
Understanding Tag Helpers

- Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files. Tag helpers are similar to HTML helpers, which help us render HTML.
- There are many built-in Tag Helpers for common tasks, such as creating forms, links, loading assets etc.
- Tag Helpers are authored in C#, and they target HTML elements based on the element name, attribute name, or the parent tag.
- For ex, LabelTagHelper can target the HTML <label> element.
- Tag Helpers reduce the explicit transitions between HTML and C# in Razor views.

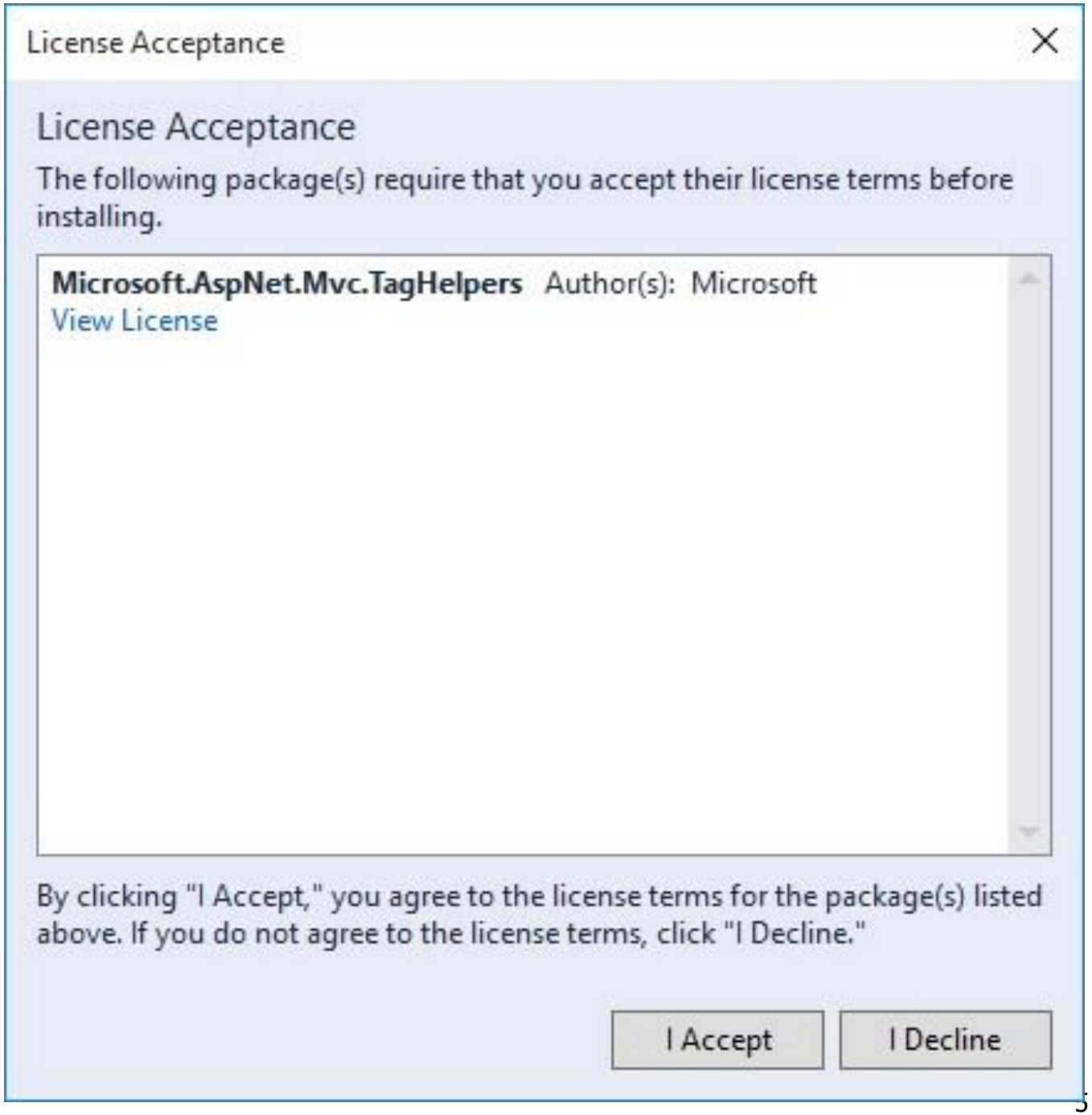
Understanding Tag Helpers

- In order to use Tag Helpers, we need to install a NuGet library and also add an addTagHelper directive to the view or views that use these tag helpers.
- Let us right-click on your project in the Solution Explorer and select Manage NuGet Packages....
- Search for Microsoft.AspNet.Mvc.TagHelpers and click the Install button.
- In the dependencies section, you will see "Microsoft.AspNet.Mvc.TagHelpers"

Understanding Tag Helpers



- You will receive the following dialog box.
- Click on I Accept



Writing your own Tag Helpers

- You can also write your own tag helper. You can place it right inside your application project, but you need to tell the Razor view engine about the tag helper. By default, they are not just rendered down to the client, even though these tag helpers look like they blend into the HTML.
- Razor will call into some code to process a tag helper; it can remove itself from the HTML and it can also add additional HTML.
- You need to register your tag helpers with Razor, even the Microsoft tag helpers, in order for Razor to be able to spot these tag helpers in the markup and to be able to call into the code that processes the tag helper.
- The directive to do that is addTagHelper, and you can place this into an individual view or ViewImports file.

Form Tag Helper

- The Form Tag Helper is bound to the HTML <form> element.
- provides several server-side attributes which help us to manipulate the generated HTML.
- Some of the available attributes are
  - **asp-controller:** The name of the MVC controller.to use
  - **asp-action:** The name of the MVC Controller action.method to use
  - **asp-area:** The name of the Controller Area to use

Form Tag Helper

EX

```
<form asp-controller="Home" asp-action="Create">
```

The above code translated into

```
<form method="post" action="/Home/Create">
  <input name="__RequestVerificationToken" type="hidden"
  value="CfDJ8PlIso5McDBOjgPkVg9O4mnNiAE8U0HkVlA9e-
  Mtc76u7fSjCnoy909Co49eGlbyJx
  pp-nYphF_XkOrPo0tTGdygc2H8nCtZCcGURMZ9Uf01fPOg5jRARxTHXnb8N6yYADtdQSn
  JItXtYsir8GCWqZM" />
</form>
```

Form Tag Helper

Label tag Helper :

```
<label asp-for="@Model.Name"></label>
```

Which translates into `<label for="Name">Name</label>`

- Using @Model keyword is optional here. You can directly use the model property name as shown below.

```
<label asp-for="Name"></label>
```

Input Tag Helper - Similarly, the Input tag Helper is applied to the input HTML element.

```
<input asp-for="Name"/>
```

Which translates into `<input type="text" id="Name" name="Name" value=""/>`

- The type, id & name attributes are automatically derived from the *model property type* & *data annotations* applied on the model property

Form Tag Helper

EX

```
<form asp-controller="Home" asp-action = "Create">
    <label asp-for = "Name"></label>
    <input asp-for = "Name"/>
    <label asp-for = "Rate"></label>
    <input asp-for = "Rate"/>
    <label asp-for = "Rating"></label>
    <input asp-for = "Rating"/>
    <input type="submit" name="submit"/>
</form>
```

List of Built-in Tag Helpers

TagHelper	Targets	Attributes
Form Tag Helper	<Form>	asp-action, asp-all-route-data, asp-area, asp-controller, asp-protocol, asp-route, asp-route-
Anchor Tag Helpers	<a>	asp-action, asp-all-route-data, asp-area, asp-controller, asp-Protocol, asp-route, asp-route-
Image Tag Helper	<img>	append-version
Input Tag Helper	<input>	for
Label Tag Helper	<label>	For
Link Tag Helper	<link>	href-include, href-exclude, fallback-href, fallback-test-value, append-version

List of Built-in Tag Helpers

TagHelper	Targets	Attributes
Options Tag Helper	<select>	asp-for, asp-items
Partial Tag Helper	<partial>	name, model, for, view-data
Script Tag Helper	<script>	src-include, src-exclude, fallback-src, fallback-src-include, fallback-src-exclude fallback-test, append-version
Select Tag Helper	<select>	for, items
Textarea Tag Helper	<textarea>	for
Validation Message Tag Helper		validation-for
Validation Summary Tag Helper		validation-summary

Model

- The Model in an MVC application should represent the current state of the application, as well as business logic and/or operations.
- A very important aspect of the MVC pattern is the Separation of Concerns (SoC). SoC is achieved by encapsulating information inside a section of code, making each section modular, and then having strict control over how each module communicates. For the MVC pattern, this means that both the View and the Controller can depend on the Model, but the Model doesn't depend on neither the View nor the Controller.
- As mentioned, the Model can be a class already defined in your project, or it can be a new class you add specifically to act as a Model. Therefore, Models in the ASP.NET MVC framework usually exists in a folder called "Models".

ViewModels

- There are, however, a lot of situations where you may want to create a specific ViewModel for a specific View. This can be to extend or simplify an existing Model, or because you want to represent something in a View that's not already covered by one of your models.
- ViewModels are often placed in their own directory in your project, called "ViewModels".
- Some people also prefer to postfix the name of the class with the word ViewModel, e.g. "AddressViewModel" or "EditUserViewModel".

When to use ViewModel?

- **To represent something in a View that's not already contained by an existing Model:** When you pass a Model to a View, you are free to pass e.g. a String or another simple type, but if you want to pass multiple values, it might make more sense to create a simple ViewModel to hold the data, like this one:

```
public class AddressViewModel
{
    public string StreetName { get; set; }
    public string ZipCode { get; set; }
}
```

When to use ViewModel?

- **To access the data of multiple Models from the same View:** This is relevant in a lot of situations, e.g. when you want to create a FORM where you can edit the data of multiple Models at the same time. You could then create a ViewModel like this:

```
public class EditItemsViewModel
{
    public Model1Type Model1 { get; set; }
    public Model2Type Model2 { get; set; }
}
```

When to use ViewModel?

- **To simplify an existing Model:** Imagine that you have a huge class with information about a user. Perhaps even sensitive information like passwords. When you want to expose this information to a View, it can be beneficiary to only expose the parts of it you actually need. For instance, you may have a small widget showing that the user is logged in, which username they have and for how long they have been logged in. So instead of passing your entire User Model, you can pass in a much leaner ViewModel, designed specifically for this purpose:

```
public class SimpleUserInfoViewModel   {
    public string Username { get; set; }
    public TimeSpan LoginDuration { get; set; }
}
```

When to use ViewModel?

- **To extend an existing Model with data only relevant to the View:** On the other hand, sometimes your Model contains less information than what you need in your View. An example of this could be that you want some convenience properties or methods which are only relevant to the View and not your Model in general, like in this example where we extend a user Model (called WebUser) with a LoginDuration property, calculated from the LastLogin DateTime property already found on the WebUser class:

```
public class WebUser
{
    public DateTime LastLogin { get; set; }
}
```

From there on there are two ways of doing things: You can either extend this class (inherit from it) or add a property for the WebUser instance on the ViewModel. Like this:

```
public class UserInfoViewModel {
    public WebUser User { get; set; }
    public TimeSpan LoginDuration
    {
        get { return DateTime.Now - this.User.LastLogin; }
    }
}
```

Or like this:

```
public class ExtendedUserInfoViewModel : WebUser {
    public TimeSpan LoginDuration
    {
        get { return DateTime.Now - this.LastLogin; }
    }
}
```

Model Binding in ASP.NET Core

- The Model Binding extracts the data from an HTTP request and provides them to the controller action method parameters. The action method parameters may be simple types like integers, strings, etc. or complex types such as Student, Order, Product, etc.
- The controller action method handle the incoming HTTP Request.
- Our application default route template ({controller=Home}/{action=Index}/{Id?})
- When you load this url - http://localhost:52191/home/details/101, shows the Details action method.

```
public ViewResult Details(int Id)
{
    var studentDetails = listStudents.FirstOrDefault(std => std.StudentId == Id);
    return View(studentDetails);
}
```

Using Model Binding

- In Model Folder, create a class *WebUser.cs*

```
public class WebUser
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

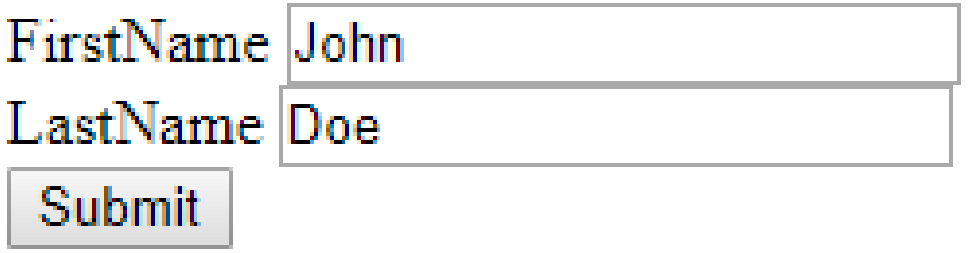
- In Controller Folder, create a new Controller as *UserController.cs* and add action method as follows:

```
[HttpGet]
public IActionResult SimpleBinding()
{
    return View(new WebUser() { FirstName = "John", LastName = "Doe" });
}
```

- By letting our View know what kind of Model it can expect, with the @model directive, we can now use various helper methods (more about those later) to help with the generation of a FORM:
- In View Folder, create a file SimpleBinding.*cshtml*

```
@using(var form = Html.BeginForm())
{
    <div>
        @Html.LabelFor(m => m.FirstName)
        @Html.TextBoxFor(m => m.FirstName)
    </div>

    <div>
        @Html.LabelFor(m => m.LastName)
        @Html.TextBoxFor(m => m.LastName)
    </div>
    <input type="submit" value="Submit" />
}
```

- The result will be a very generic-looking FORM, but with labels and textboxes designated to host the properties of your Model:



- By default, the FORM will be posted back to the same URL that delivered it, so to handle that, we need a POST-accepting Controller method to handle the FORM submission:

```
[HttpPost]
public IActionResult SimpleBinding(WebUser webUser)
{
    //TODO: Update in DB here...
    return Content($"User {webUser.FirstName} updated!");
}
```

Data Annotations

- Data Annotations (sometimes referred to as Model Attributes), which basically allows you to add meta data to a property.
- The cool thing about DataAnnotations is that they don't disturb the use of your Models outside of the MVC framework.
- When generating the label, the name of the property is used, but property names are generally not nice to look at for humans. As an example of that, we might want to change the display-version of the FirstName property to "First Name".

```
public class WebUser
{
    [Display(Name="First Name")]
    public string FirstName { get; set; }
}
```

Model Validation

- They will allow you to enforce various kinds of rules for your properties, which will be used in your Views and in your Controllers, where you will be able to check whether a certain Model is valid in its current state or not (e.g. after a FORM submission).
- Let's add just a couple of basic validation to the WebUser

```
public class WebUser  {
    [Required]
    [StringLength(25)]
    public string FirstName { get; set; }
    [Required]
    [StringLength(50, MinLength(3)]
    public string LastName { get; set; }
    [Required]
    [EmailAddress]
    public string MailAddress { get; set; }
}
```

Model Validation

- Notice how the three properties have all been decorated with DataAnnotations.
- First of all, all properties have been marked with the [Required] attribute, meaning that a value is required - it can't be NULL.
- [StringLength] attribute make requirements about the maximum, and in one case minimum, length of the strings. These are of course particularly relevant if your Model corresponds to a database table, where strings are often defined with a maximum length.
- For the last property, [EmailAddress] attribute ensure that the value provided looks like an e-mail adress.

```
@model HelloMVCWorld.Models.WebUser
@using(var form = Html.BeginForm()) {
    <div>
        @Html.LabelFor(m => m.FirstName)
        @Html.TextBoxFor(m => m.FirstName)
    </div>
    <div>
        @Html.LabelFor(m => m.LastName)
        @Html.TextBoxFor(m => m.LastName)
    </div>
    <div>
        @Html.LabelFor(m => m.MailAddress)
        @Html.TextBoxFor(m => m.MailAddress)
    </div>
    <input type="submit" value="Submit" />
}
```

In your View Page

```
public class ValidationController : Controller
{
    [HttpGet]
    public IActionResult SimpleValidation()
    {
         return View();
    }
    [HttpPost]
    public IActionResult SimpleValidation(WebUser webUser)
    {
        if(ModelState.IsValid)
             return Content("Thank you!");
        else
             return Content("Model could not be validated!");
    }
}
```

In your Controller

- In POST action, we check the IsValid property of the ModelState object. Depending on the data submitted in the FORM, it will be either true or false, based on the validation rules we defined for the Model (WebUser).
- With this in place, you can now prevent a Model from being saved, e.g. to a database, unless it's completely valid.

```
@model HelloMVCWorld.Models.WebUser
@using(var form = Html.BeginForm()) {
    <div>
       @Html.LabelFor(m => m.FirstName)
       @Html.TextBoxFor(m => m.FirstName)
       @Html.ValidationMessageFor(m => m.FirstName)
    </div>
    <div>
       @Html.LabelFor(m => m.LastName)
       @Html.TextBoxFor(m => m.LastName)
       @Html.ValidationMessageFor(m => m.LastName
    </div>
<div>
@Html.LabelFor(m => m.MailAddress)
@Html.TextBoxFor(m => m.MailAddress)
@Html.ValidationMessageFor(m => m.MailAddress)
</div>
<input type="submit" value="Submit" />
}
```

Displaying validation errors

- Let's extend our FORM so that it can display error messages to the user. We can use helper method - *ValidationMessageFor()*.
- It will simply output the error message related to the field, if there is one - otherwise, nothing will be outputted. Here's the extended version of our FORM:

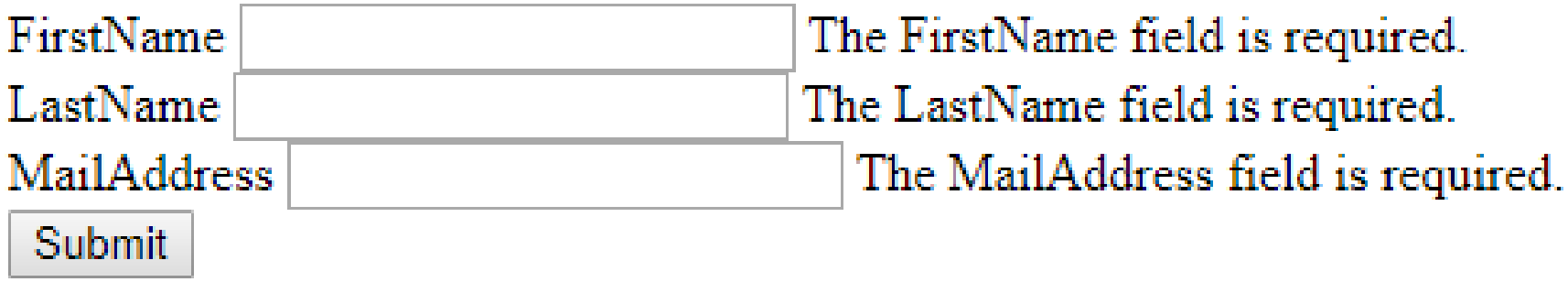
```
public class ValidationController : Controller
{
    [HttpGet]
    public IActionResult SimpleValidation()
    {
        return View();
    }
    [HttpPost]
  public IActionResult SimpleValidation(WebUser webUser)
  {
      if(ModelState.IsValid)
          return Content("Thank you!");
      else
          return View(webUser);
  }
}
```

In your Controller

- make sure that once the FORM is submitted, and if there are validation errors, we return the FORM to the user, so that they can see and fix these errors.
- We do that in our Controller, simply by returning the View and the current Model state, if there are any validation errors:

Try Submitting the Form

- Try submitting the FORM with empty fields, you should be immediately returned to the FORM, but with validation messages next to each of the fields.



- If you try submitting the FORM with a value that doesn't meet the StringLength requirements, you will notice that there are even automatically generated error messages for these as well. For instance, if you submit the FORM with a LastName that's either too long or too short, you will get this message:
- The field LastName must be a string with a minimum length of 3 and a maximum length of 50.

What if you want more control of these messages?

- But No problem, they can be overridden directly in the DataAnnotations of the Model. Here's a version of our Model where we have applied custom error messages:

```
public class WebUser  {
    [Required(ErrorMessage = "You must enter a value for the First Name field!")]
    [StringLength(25, ErrorMessage = "The First Name must be no longer than 25 characters!")]
    public string FirstName { get; set; }

    [Required(ErrorMessage = "You must enter a value for the Last Name field!")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "The Last Name must be between 3 and 50 characters long!")]
    public string LastName { get; set; }

    [Required(ErrorMessage = "You must enter a value for the Mail Address field!")]
    [EmailAddress(ErrorMessage = "Please enter a valid e-mail address!")]
    public string MailAddress { get; set; }
}
```

```
@model HelloMVCWorld.Models.WebUser
@using(var form = Html.BeginForm())
{
    @Html.ValidationSummary()
    <div>
        @Html.LabelFor(m => m.FirstName)
        @Html.TextBoxFor(m => m.FirstName)
    </div>
    <div>
       @Html.LabelFor(m => m.LastName)
       @Html.TextBoxFor(m => m.LastName)
    </div>
    <div>
       @Html.LabelFor(m => m.MailAddress)
       @Html.TextBoxFor(m => m.MailAddress)
    </div>

    <input type="submit" value="Submit" />
}
```

Displaying a validation summary

- You must enter a value for the First Name field!
- The Last Name must be between 3 and 50 characters long!
- Please enter a valid e-mail address!

FirstName
LastName aa
MailAddress test
Submit

- Use ValidationSummary() method found on the Html helper object:
- Now, When the FORM is submitted, It will be returned with validation errors

Types of Model Validation DataAnnotations

- **[Required] -** Specifies that a value needs to be provided for this property
- **[StringLength] -** Allows you to specify at least a maximum amount of characters. We can also add Minumum Length as well.

```
[StringLength(50, MinimumLength = 3)]
```

- **[Range] -** specify a minimum and a maximum value for a numeric property (int, float, double etc.)

```
[Range(1, 100)]
```

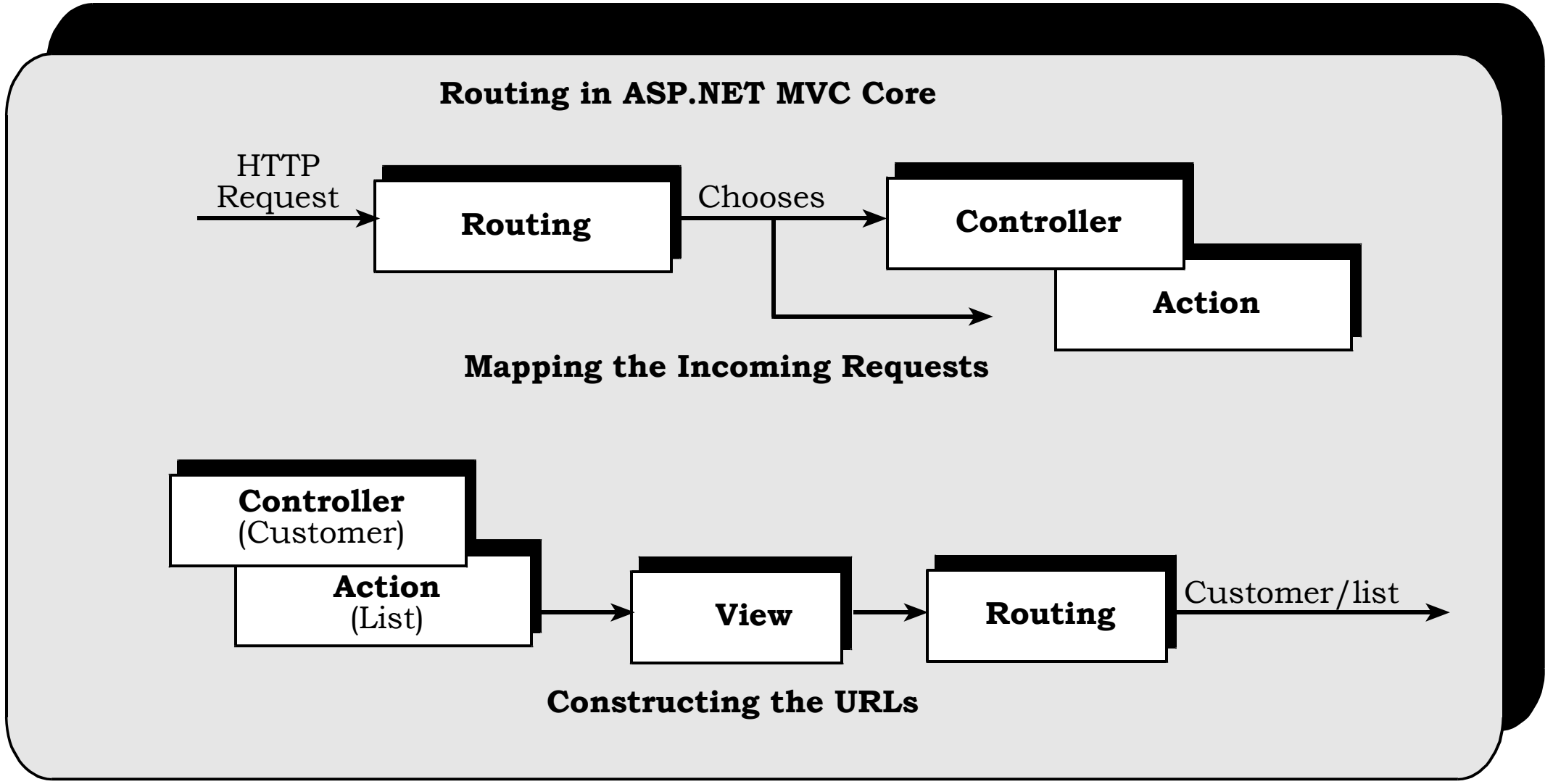
- **[Compare] -** allows you to set up a comparison between the property

```
[Compare("MailAddressRepeated")]
public string MailAddress { get; set; }
public string MailAddressRepeated { get; set; }
```

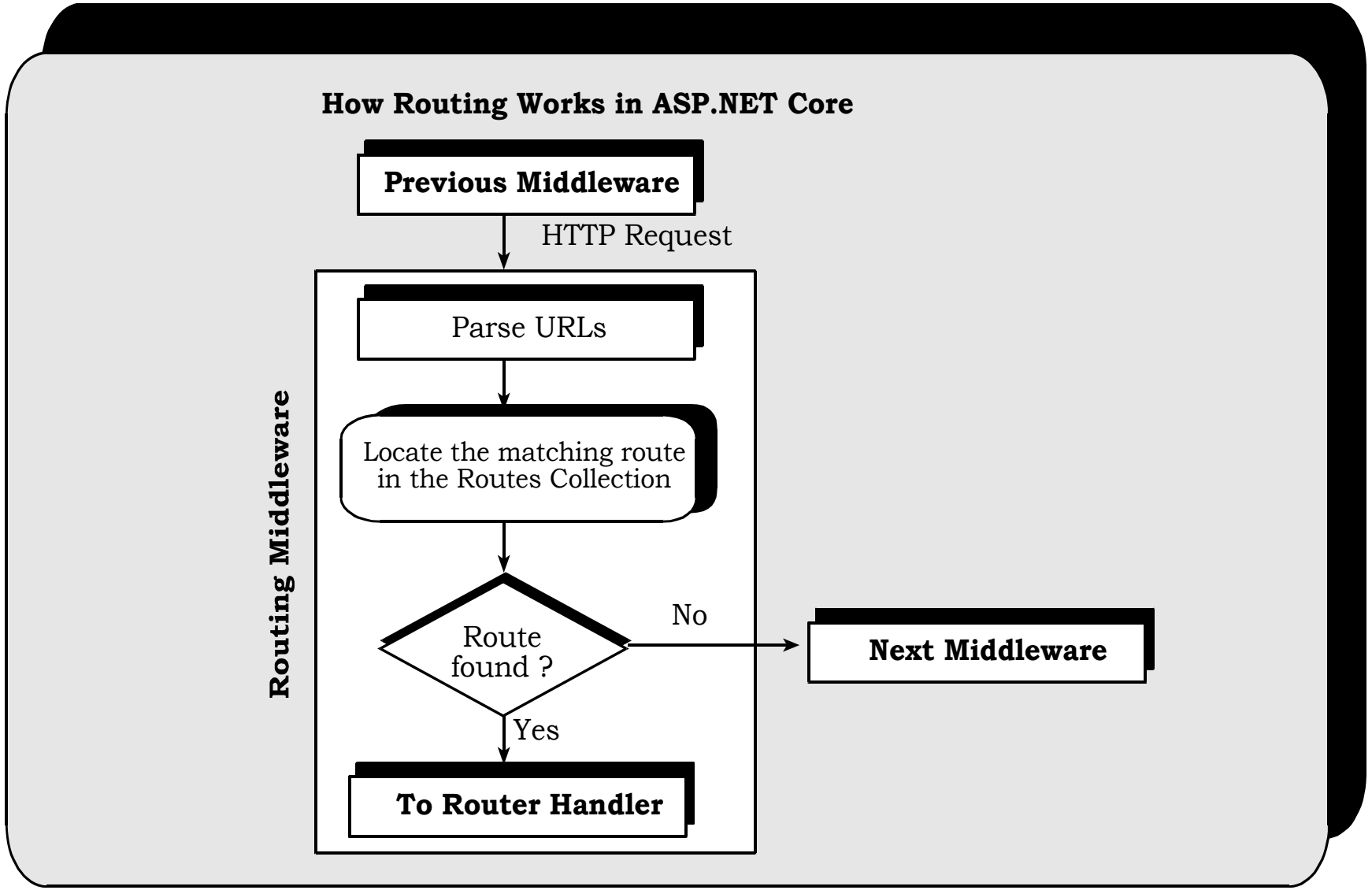
URL Routing and Features

- The Routing is the Process by which ASP.NET Core inspects the incoming URLs and maps them to Controller Actions.
- It also used to generate the outgoing URLs.
- This process is handled by the Routing Middleware. The Routing Middleware is available in Microsoft.AspNetCore.Routing Namespace .
- The Routing has two main responsibilities:
    1. It maps the incoming requests to the Controller Action
    2. Generate an outgoing URLs that correspond to Controller actions.

Routing in ASP.NET MVC Core



How Routing works in ASP.NET MVC Core



How Routing works in ASP.NET MVC Core

- When the Request arrives at the Routing Middleware it does the following.

1. It Parses the URL.
2. Searches for the Matching Route in the RouteCollection.
3. If the Route found then it passes the control to RouteHandler.
4. If Route not found, it gives up and invokes the next Middleware.

How Routing works in ASP.NET MVC Core

What is a Route

- The Route is similar to a roadmap. We use a roadmap to go to our destination. Similarly, the ASP.NET Core Apps uses the Route to go to the controller action.
- The Each Route contains a Name, URL Pattern (Template), Defaults and Constraints. The URL Pattern is compared to the incoming URLs for a match. An example of URL Pattern is {controller=Home}/{action=Index}/{id?}
- The Route is defined in the Microsoft.AspNetCore.routing namespace .

How Routing works in ASP.NET MVC Core

What is a Route Collection

- The Route Collection is the collection of all the Routes in the Application.
- An app maintains a single in-memory collection of Routes. The Routes are added to this collection when the application starts.
- The Routing Module looks for a Route that matches the incoming request URL on each available Route in the Route collection.
- The Route Collection is defined in the namespace Microsoft.AspNetcore.routing.

How Routing works in ASP.NET MVC Core

What is a Route Handler

- The Route Handler is the Component that decides what to do with the route.
- When the routing Engine locates the Route for an incoming request, it invokes the associated RouteHandler and passes the Route for further processing. The Route handler is the class which implements the IRouteHandler interface.
- In the ASP.NET Core, the Routes are handled by the MvcRouteHandler.

How Routing works in ASP.NET MVC Core

MVCRouteHandler

- Default Route Handler for the ASP.NET Core MVC Middleware. The MVCRouteHandler is registered when we register the MVC Middleware in the Request Pipeline. You can override this and create your own implementation of the Route Handler.
- defined in the namespace Microsoft.AspnetCore.Mvc.
- The MVCRouteHandler is responsible for invoking the Controller Factory, which in turn creates the instance of the Controller associated the Route.
- The Controller then takes over and invokes the Action method to generate the View and Complete the Request.

How to setup Routes

- There are two different ways by which we can set up routes.
    1. Convention-based routing
    2. Attribute routing

Convention-based routing

- The Convention based Routing creates routes based on a series of conventions, defined in the ASP.NET Core Startup.cs file.

Attribute routing

- Creates routes based on attributes placed on controller actions.

The two routing systems can co-exist in the same system.

How to setup Routes

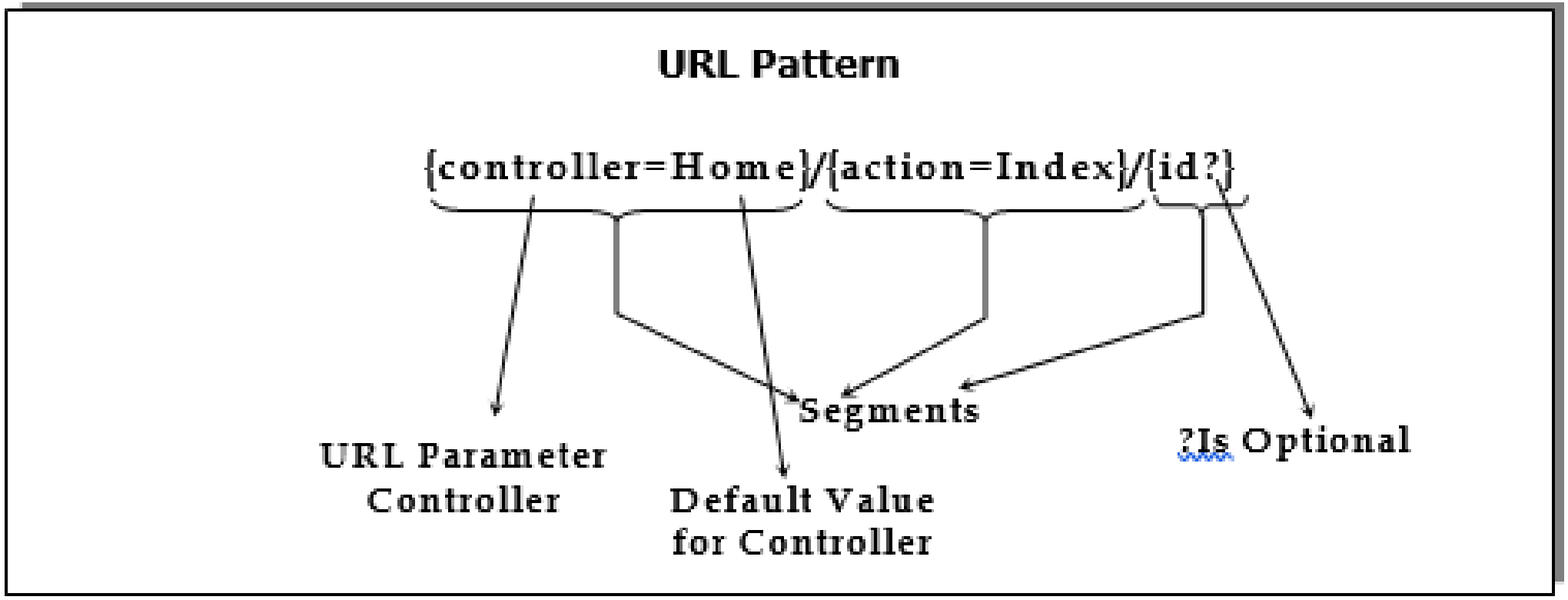
- The Convention based Routes are configured in the Configure method of the Startup class. The Routing is handled by the Router Middleware. ASP.NET MVC adds the routing Middleware to the Middleware pipeline when using the app.UseMVC or app.UseMvcWithDefaultRoute.

MapRoute Api

```
app.UseMvc(routes=>
    {
      routes.MapRoute(
         "default",  ──────────► Name of the Route
         "{controller=Home}/{action=Index}/{id?}"), ──► URL
      Pattern
```

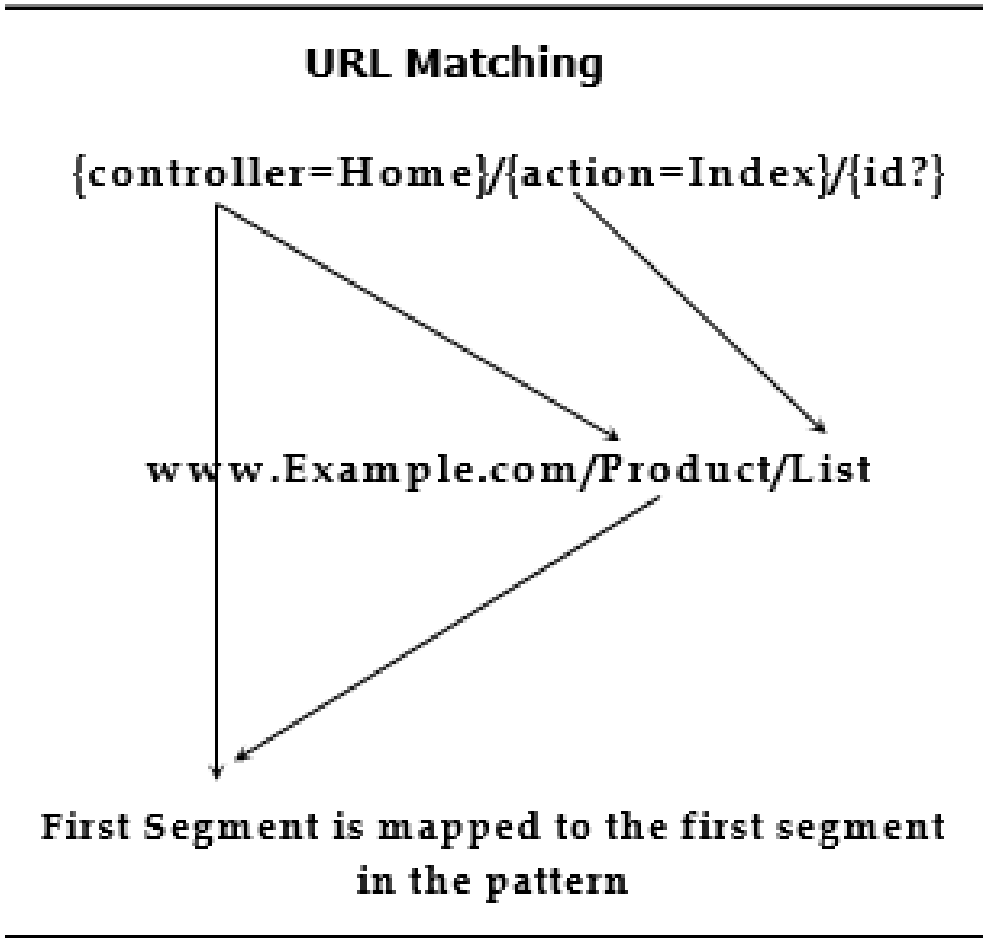
URL Patterns

- The Each route must contain a URL pattern. This Pattern is compared to an incoming URL. If the pattern matches the URL, then it is used by the routing system to process that URL.



- The URL Pattern {controller=Home}/{action=Index}/{id?} Registers route where the first part of the URL is Controller, the second part is the action method to invoke on the controller. The third parameter is an additional data in the name of id.

- The Each segment in the incoming URL is matched to the corresponding segment in the URL Pattern.
- {controller=Home}/{action=Index}/{id?} has three segments. The last one is optional.

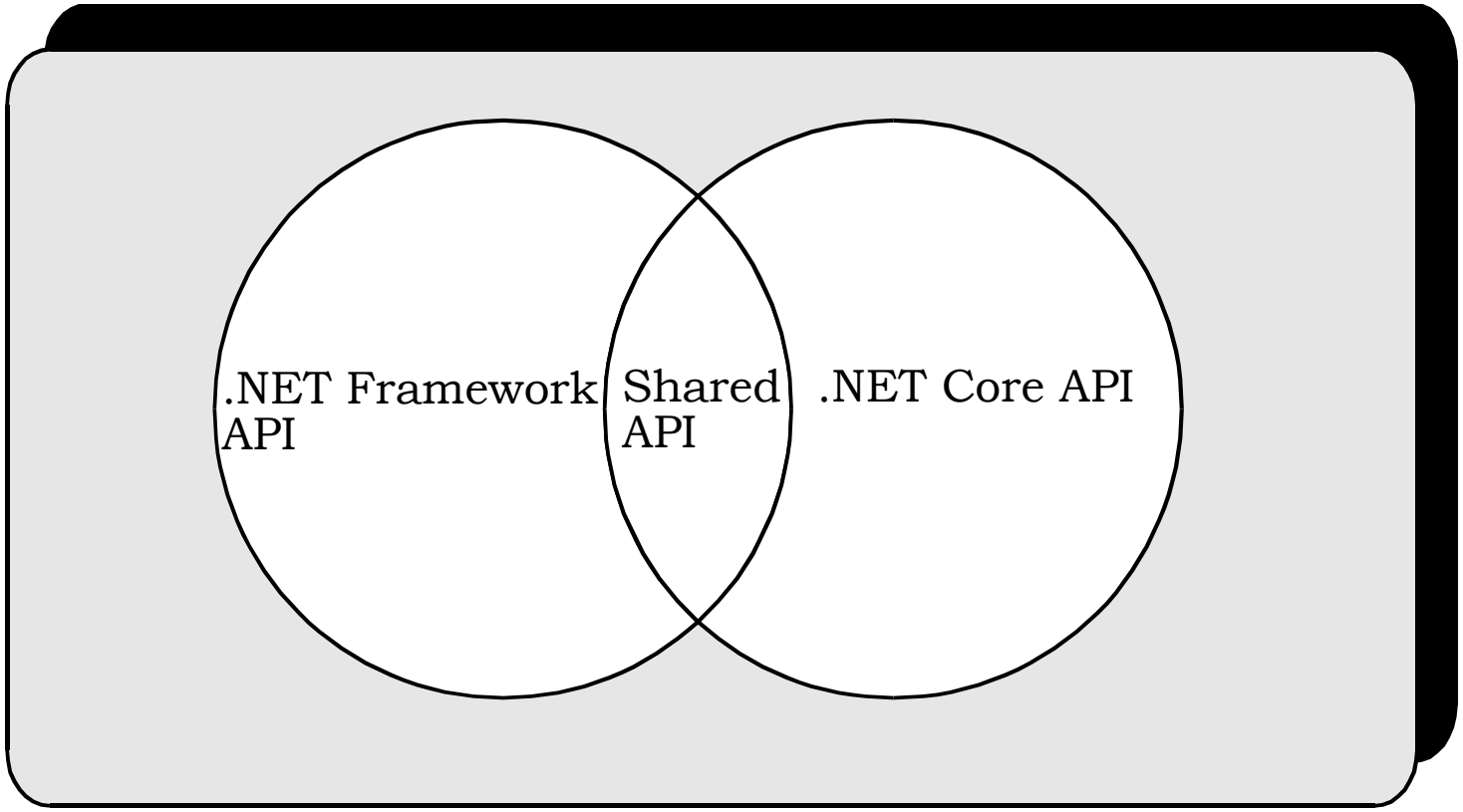


Web API Applications

- Before ASP.NET Web API core, the two-different framework MVC and Web API were pretty much similar.
- Both used to support Controller and action methods. In earlier version, the main purpose of Web API was to make REST API calls and there were view engine like Razor.
- On the other hand, MVC was designed for HTML front ends to communicate to backend in a standard a web application. However, when ASP.NET Web API core was released, the main target was to support JSON based REST API. It combines the key feature of both MVC and old Web API framework.

ASP.NET Core Web API Architecture

- ASP.NET Web API is mainly based on the MVC architecture. The .NET framework and .NET Core also share a number of APIs.



New Features in ASP.NET Core Web API

- **Cross Platform -** ASP.NET Web API Core is cross-platform; therefore, it is suitable for running on any platform like Windows, Mac, or Linux. Earlier ASP.NET applications were not able to run on Linux and Mac operating system.
- **Backward Compatibility -** For existing application, ASP.NET Web API Core supports two framework.
- **Faster** - ASP.NET Web API Core is much faster than previous versions
- **Static Content -** wwwroot folder contain all the static content e.g. js, css, images.

Creating Web API in ASP.NET Core

- Create the controller that have 3 things:
  - should have [ApiController] attribute on them. This attribute tells that the controller will serve HTTP API Responses.
  - derive from ControllerBase class instead of Controller class.
  - should have attribute routing applied on them like [Route("someUrl/[controller]")].
  - The controller of a Web API looks like:

```
[ApiController]
[Route("someURL/[controller]")]
public class ExampleController : ControllerBase
```

API Controllers

- API Controller is just a normal Controller, that allows data in the model to be retrieved or modified, and then deliver it to the client. It does this without having to use the actions provided by the regular controllers.
- The data delivery is done by following a pattern known by name as REST. REST Stands for REpresentational State Transfer pattern, which contains 2 things:
  - Action Methods which do specific operations and then deliver some data to the client. These methods are decorated with attributes that makes them to be invoked only by HTTP requests.
  - URLs which defines operational tasks. These operations can be – sending full or part of a data, adding, deleting or updating records. In fact it can be anything.

API Controller

- MVC and API controllers both derive from the Controller class, which derives from ControllerBase:

```
public class MyMvc20Controller : Controller {}
[Route("api/[controller]")]
public class MyApi20Controller : Controller {}
```

- As of Core 2.1 (and 2.2), the template-generated classes look a little different, where a Web controller is a child of the Controller class and an API controller is a child of ControllerBase.

```
public class MyMvc21Controller : Controller {}
[Route("api/[controller]")]
public class MyApi21Controller : ControllerBase {}
```

API Controller

- This can be expressed in the table below:

Namespace	Microsoft.AspNetCore.Mvc
Common parent	ControllerBase (Abstract Class)
MVC Controller parent	Controller: ControllerBase
MVC Controller	MyMvcController: Controller

JSON

- The new built-in JSON support, System.Text.Json, is high-performance, low allocation, and based on Span<byte>.
- The System.Text.Json namespace provides high-performance, low-allocating, and standards-compliant capabilities to process JavaScript Object Notation (JSON), which includes serializing objects to JSON text and deserializing JSON text to objects, with UTF-8 support built-in.
- It also provides types to read and write JSON text encoded as UTF-8, and to create an in-memory document object model (DOM) for random access of the JSON elements within a structured view of the data.

Adding JSON Patch To Your ASP.Net Core Project

- Run Package Manager and install JSON Patch Library with command:
  - `Install-PackageMicrosoft.AspNetCore.JsonPatch`
- Write in your controller

```
public class Person
{
    public string FirstName{get;set;}
    public string LastName{get;set;}
}
```

```
[Route("api/[controller]")]
public class PersonController : Controller {
private readonly Person _defaultPerson =newPerson
{
    FirstName="Jim",
  LastName="Smith"
};
[HttpPatch("update")]
      public Person Patch([FromBody]JsonPatchDocument<Person> personPatch) {
                personPatch.ApplyTo(_defaultPerson);
       return _defaultPerson;
    }
}
```

Adding JSON Patch To Your ASP.net Core Project

- In above example we are just using a simple object stored on the controller and updating that, but in a real API we will be pulling the data from a datasource, applying the patch, then saving it back.
- When we call this endpoint with the following payload :

```
[{"op":"replace","path":"FirstName","value":"Bob"}]
```

- We get the response of :

```
{"firstName":"Bob","lastName":"Smith"}
```

first name got changed to Bob!

DEPENDENCY INJECTION AND IOC CONTAINERS

- ASP.NET Core is designed from scratch to support Dependency Injection.
- ASP.NET Core injects objects of dependency classes through constructor or method by using built-in IoC container.
- ASP.NET Core framework contains simple out-of-the-box IoC container which does not have as many features as other third party IoC containers. If you want more features such as auto-registration, scanning, interceptors, or decorators then you may replace built-in IoC container with a third party container.

BUILT-IN IOC CONTAINER

- The built-in container is represented by IServiceProvider implementation that supports constructor injection by default. The types (classes) managed by built-in IoC container are called services.
- There are basically two types of services in ASP.NET Core:
  1. Framework Services: Services which are a part of ASP.NET Core framework such as IApplicationBuilder, IHostingEnvironment, ILoggerFactory etc.
  2. Application Services: The services (custom types or classes) which you as a programmer create for your application.
- In order to let the IoC container automatically inject our application services, we first need to register them with IoC container.

Registering Application Service

- Consider the following example of simple ILog interface and its implementation class. We will see how to register it with built-in IoC container and use it in our application.

```
public interface ILog {
    void info(string str);
}
class MyConsoleLogger : ILog {
    public void info(string str)
    {
            Console.WriteLine(str);
    }
}
```

Registering Application Service

- ASP.NET Core allows us to register our application services with IoC container, in the ConfigureServices method of the Startup class. The ConfigureServices method includes a parameter of IServiceCollection type which is used to register application services.
- Let's register above ILog with IoC container in ConfigureServices() method as shown below. Example: Register Service

```
public class Startup {
        public void ConfigureServices(IServiceCollection services) {
                services.Add(new ServiceDescriptor(typeof(ILog),
                                        new MyConsoleLogger()));
   } // other code removed for clarity..
}
```

Registering Application Service

- In above ex:
- Add() method of IServiceCollection instance is used to register a service with an IoC container.
- ServiceDescriptor is used to specify a service type and its instance. We have specified ILog as service type and MyConsoleLogger as its instance. This will register ILog service as a singleton by default.
- Now, an IoC container will create a singleton object of MyConsoleLogger class and inject it in the constructor of classes wherever we include ILog as a constructor or method parameter throughout the application.
- Thus, we can register our custom application services with an IoC container in ASP.NET Core application. There are other extension methods available for quick and easy registration of services.

Understanding Service Lifetime for Registered Service

- Built-in IoC container manages the lifetime of a registered service type. It automatically disposes a service instance based on the specified lifetime.
- The built-in IoC container supports three kinds of lifetimes:

1. Singleton: IoC container will create and share a single instance of a service throughout the application's lifetime.
2. Transient: The IoC container will create a new instance of the specified service type every time you ask for it.
3. Scoped: IoC container will create an instance of the specified service type once per request and will be shared in a single request.

Understanding Service Lifetime for Registered Service

- The following example shows how to register a service with different lifetimes.
- Example: Register a Service with Lifetime

```
public void ConfigureServices(IServiceCollection services)
{
    // singleton
    services.Add(new ServiceDescriptor(typeof(ILog), new MyConsoleLogger()));
    services.Add(new ServiceDescriptor(typeof(ILog), typeof(MyConsoleLogger),
                                        ServiceLifetime.Transient)); // Transient
    services.Add(new serviceDescriptor(typeof(ILog), typeof(MyConsoleLogger),
                                        ServiceLifetime.Scoped));    // Scoped
}
```

IOC Containers

- ASP.NET Core framework includes built-in IoC container for automatic dependency injection. The built-in IoC container is a simple yet effective container.
- The followings are important interfaces and classes for built-in IoC container:

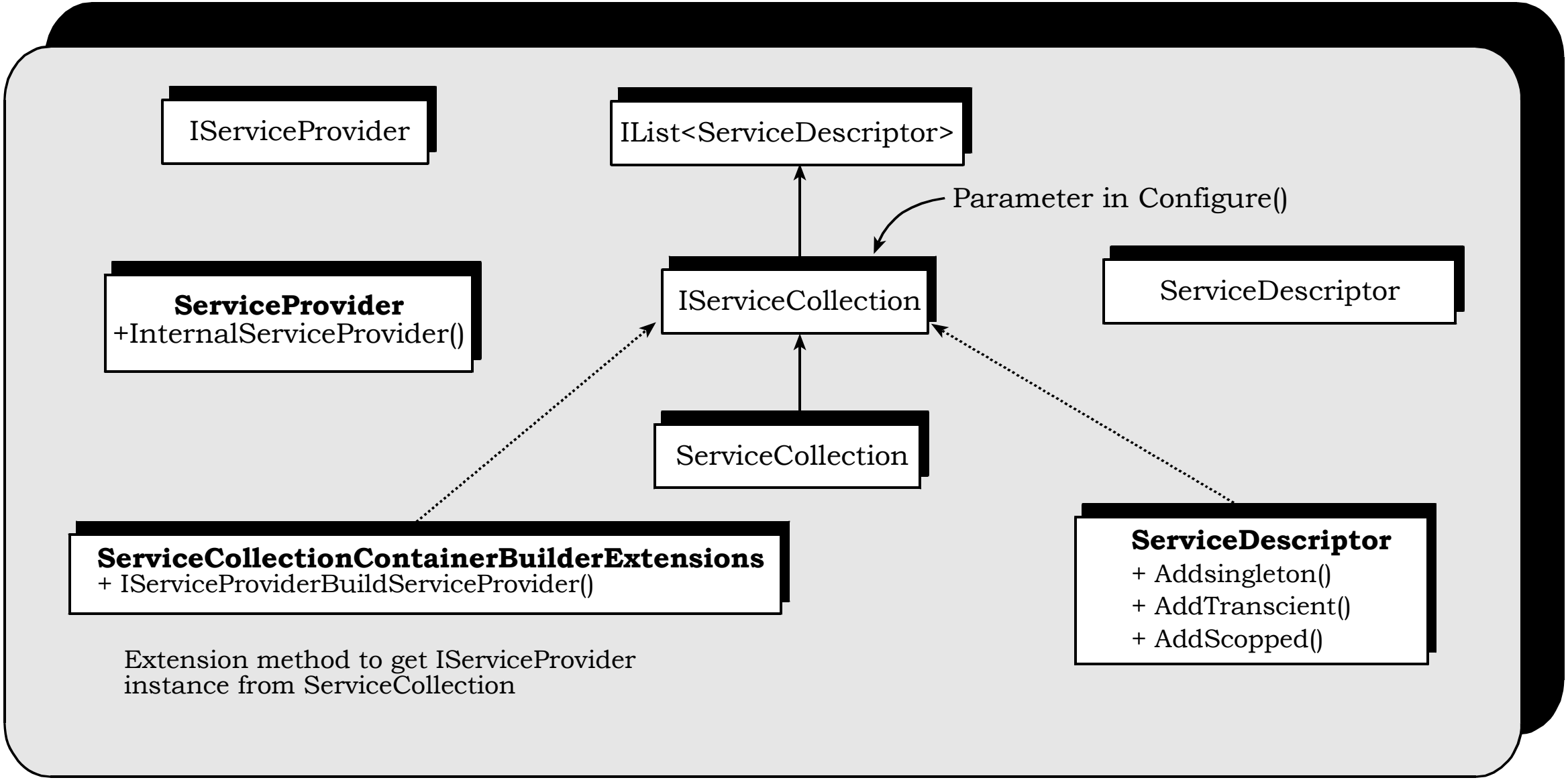
Interfaces

1. IServiceProvider
2. IServiceCollection

Classes

1. ServiceProvider
2. ServiceCollection
3. ServiceDescription
4. ServiceCollectionServiceExtensions
5. ServiceCollectionContainerBuilderExtensions

IOC Containers



IOC Containers

IServiceCollection

- we can register application services with built-in IoC container in the Configure method of Startup class by using IServiceCollection. IServiceCollection interface is an empty interface. It just inherits IList<servicedescriptor>.
- The ServiceCollection class implements IServiceCollection interface.
- So, the services you add in the IServiceCollection type instance, it actually creates an instance of ServiceDescriptor and adds it to the list.

IServiceProvider

- IServiceProvider includes GetService method.
- The ServiceProvider class implements IServiceProvider interface which returns registered services with the container. We cannot instantiate ServiceProvider class because its constructors are marked with internal access modifier.

IOC Containers

ServiceCollectionServiceExtensions

- The ServiceCollectionServiceExtensions class includes extension methods related to service registrations which can be used to add services with lifetime. AddSingleton, AddTransient, AddScoped extension methods defined in this class.

ServiceCollectionContainerBuilderExtensions

- ServiceCollectionContainerBuilderExtensions class includes BuildServiceProvider extension method which creates and returns an instance of ServiceProvider.
- There are three ways to get an instance of IServiceProvider:
  - Using IApplicationBuilder
  - Using HttpContext
  - Using IServiceCollection

IOC Containers

Using IApplicationBuilder

- We can get the services in Configure method using IApplicationBuilder's ApplicationServices property as shown below.

```
public void Configure(IServiceProvider pro, IApplicationBuilder app,
      IHostingEnvironment env)
{
      var services = app.ApplicationServices;
      var logger = services.GetService<ILog>() }
      //other code removed for clarity
}
```

IOC Containers

Using HttpContext

```
var services = HttpContext.RequestServices;
var log = (ILog)services.GetService(typeof(ILog));
```

Using IServiceCollection

```
public void ConfigureServices(IServiceCollection services)
{
      var serviceProvider = services.BuildServiceProvider();
}
```

Unit 5

Working with Database

Database

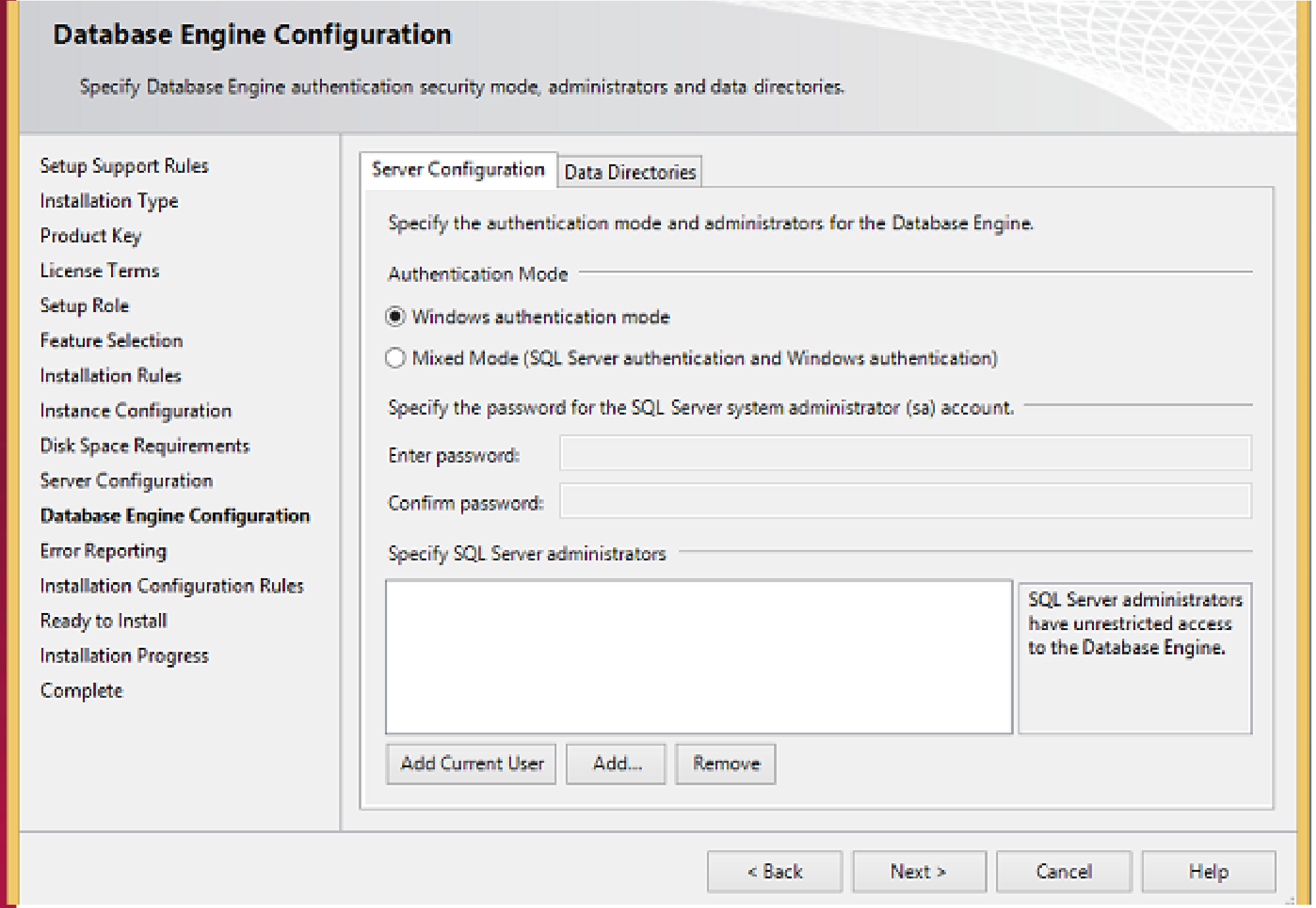
- A database is an organized collection of structured information, or data, typically stored electronically in a computer system.
- A database is usually controlled by a database management system (DBMS).
- The main purpose of the database is to operate a large amount of information by storing, retrieving, and managing data.
- Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient.
- Most databases use Structured Query Language (SQL) for writing and querying data
- There are many databases available like SQL Server, Oracle, MySQL, MongoDB, PostgreSQL, Sybase, Informix, etc.

SQL Server

- SQL Server is a relational database management system, or RDBMS, developed and marketed by Microsoft.
- Similar to other RDBMS software, SQL Server is built on top of SQL, a standard programming language for interacting with the relational databases.
- SQL server is tied to Transact-SQL, or T-SQL, the Microsoft's implementation of SQL that adds a set of proprietary programming constructs.

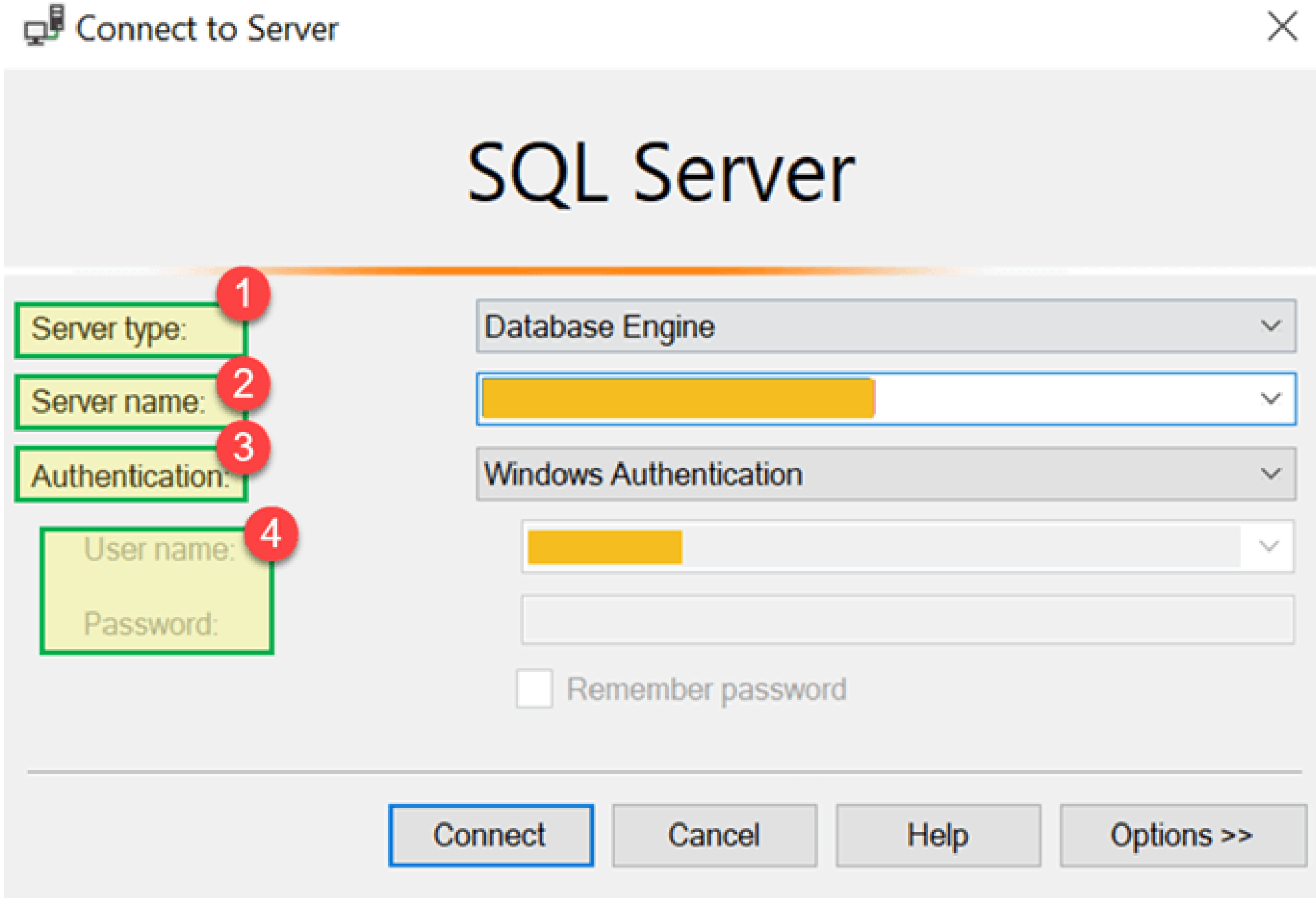
Download and Setup SQL Server

- Go to URL: https://www.microsoft.com/en-in/sql-server/sql-server-downloads
- Download Free Edition of MS SQL Server. Either Developer or Express Edition
- During installation, remember these:
  - In Instance Configuration Screen, choose **Default Instance**
  - In Database Engine Configuration Screen, choose **Mixed Mode(SQL Server authentication and Windows Authentication)** and Enter Password
  - Then follow Next Button.


Database Engine Configuration
Specify Database Engine authentication security mode, administrators and data directories.
Setup Support Rules
Installation Type
Product Key
License Terms
Setup Role
Feature Selection
Installation Rules
Instance Configuration
Disk Space Requirements
Server Configuration
Database Engine Configuration
Error Reporting
Installation Configuration Rules
Ready to Install
Installation Progress
Complete
Server Configuration
Data Directories
Specify the authentication mode and administrators for the Database Engine.
Authentication Mode
Windows authentication mode
Mixed Mode (SQL Server authentication and Windows authentication)
Specify the password for the SQL Server system administrator (sa) account.
Enter password:
Confirm password:
Specify SQL Server administrators
SQL Server administrators have unrestricted access to the Database Engine.
Add Current User
Add...
Remove
< Back
Next >
Cancel
Help

Install SQL Sever Management Studio

- Get SSMS from this url - https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017
- Install SSMS
- After install, search Microsoft SQL Server Management Studio and run
- You will see the screen as shown.
- On Authentication, Select SQL Server Authentication. For User name enter sa and for password, enter the one that use provide during installation.


Connect to Server
SQL Server
1
Server type:
Database Engine
2
Server name:
3
Authentication:
Windows Authentication
4
User name:
Password:
Remember password
Connect
Cancel
Help
Options >>

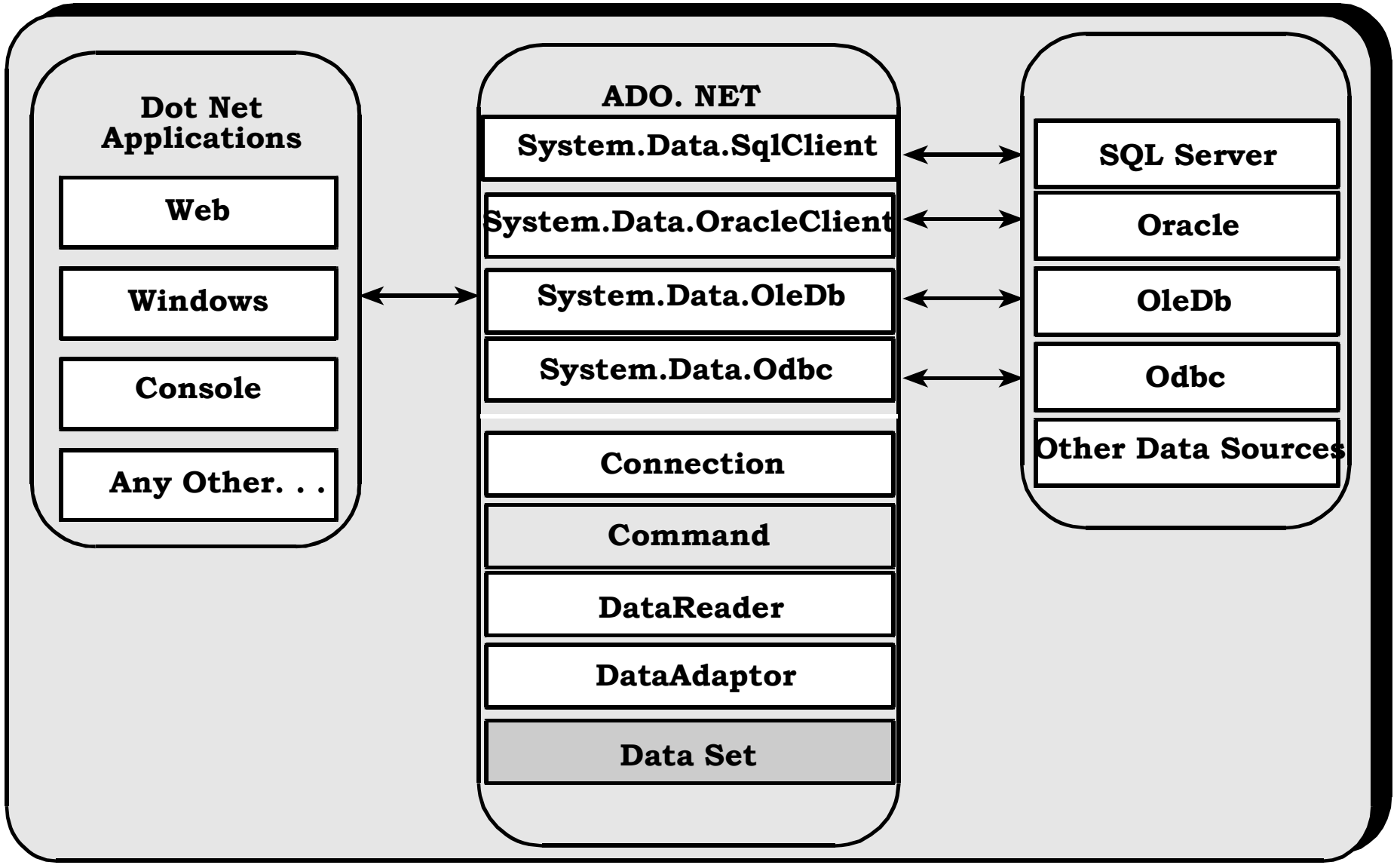
ADO.NET Basics

- ADO stands for Microsoft ActiveX Data Objects.
- The ADO.NET is one of the Microsoft's data access technology which is used to communicate between the .NET Application (Console, WCF, WPF, Windows, MVC, Web Form, etc.) and data sources such as SQL Server, Oracle, MySQL, XML document, etc.
- It has classes and methods to retrieve and manipulate data.
- The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.
  - ASP.NET Web Applications
  - Console Applications
  - Windows Applications.

2 Types of Connection Architectures

1. **Connected architecture**:
   - the application remains connected with the database throughout the processing.
2. **Disconnected architecture:**
   - the application automatically connects/disconnects during the processing.
   - The application uses temporary data on the application side called a DataSet.

Understanding ADO.NET and its class library



Important Classes in ADO.NET

1. Connection Class
2. Command Class
3. DataReader Class
4. DataAdaptor Class
5. DataSet Class

Connection Class

- In ADO.NET, we use connection classes to connect to the database.
- These connection classes also manage transactions and connection pooling.

Important Classes in ADO.NET

Command Class

provides methods for storing and executing SQL statements and Stored Procedures. Various commands that are executed by the Command Class:

a. ExecuteReader:

- Returns data to the client as rows.
- This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. T
- this method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.

Important Classes in ADO.NET

DataReader Class

- The DataReader is used to retrieve data.
- It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.

DataAdapter Class

- The DataAdapter is used to connect DataSets to databases.
- The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views and Stored Procedures.

Important Classes in ADO.NET

DataSet Class

- The DataSet is essentially a collection of DataTable objects.
- In turn each object contains a collection of DataColumn and DataRow objects.
- The DataSet also contains a Relations collection that can be used to define relations among Data Table Objects.

Connect to a Database using ADO.NET

- To create a connection, we have to use the connection strings.
- A connection string is required as a parameter to SQLConnection.
- A ConnectionString is a string variable (not case sensitive).
- This contains key and value pairs:Provider, Server, Database, User Id and Password as in the following:

```
Server="name of the server or IP Address of the server"
Database="name of the database"
UserId="user name who has permission to work with database"
Password="the password of User Id"
```

- Example - SQL Authentication

```
string constr="server=.;database=db1;user id=sa;password=yourpassword";
```

How to connect, retrieve and display data from a database

1. Create a SqlConnection object using a connection string.
2. Handle exceptions.
3. Open the connection.
4. Create a SQLCommand. To represent a SQLCommand like (select * from studentdetails) and attach the existing connection to it. Specify the type of SQLCommand (Text/StoredProcedure).
5. Execute the command (use ExecuteReader).
6. Get the Result (use SqlDataReader). This is a forwardonly/readonly data object.
7. Process the result
8. Display the result
9. Close the connection

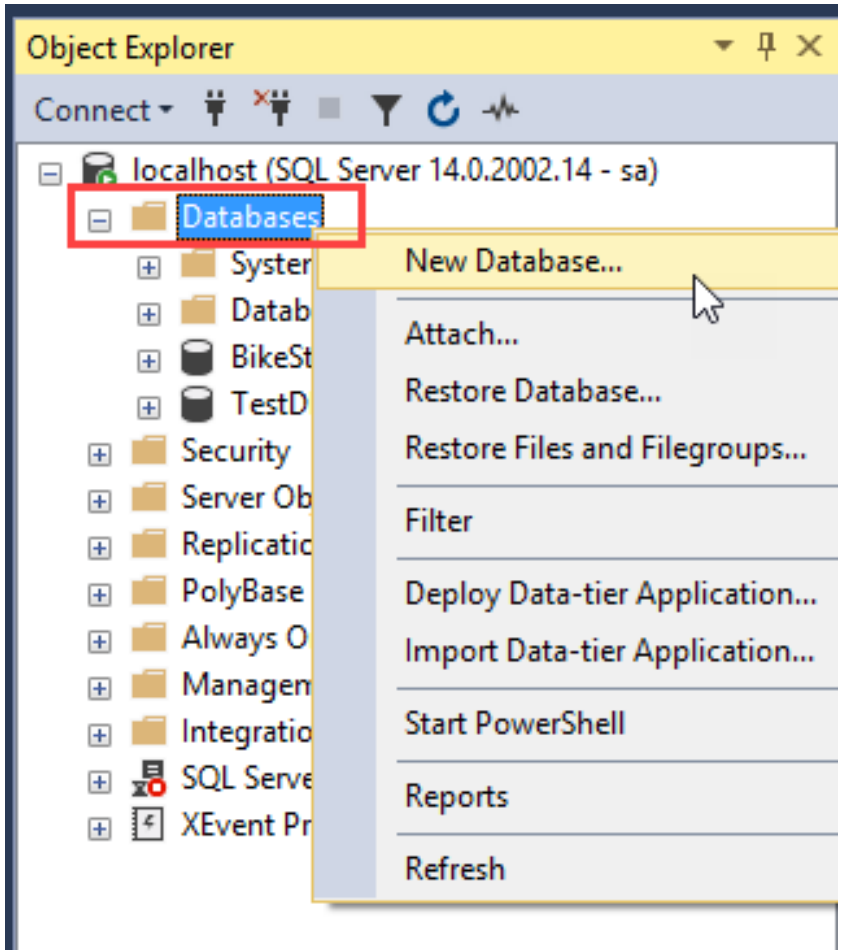
Create a Database in SQL Server

- **Create Database using CREATE DATABASE** statement

  Syntax: CREATE DATABASE database_name;

  Ex: CREATE DATABASE TestDb;

- **Create Database using Object Explorer**
  - Right Click the Database, choose New Database
  - Enter name for the database as TestDb
  - Then OK



Create a Table in SQL Server

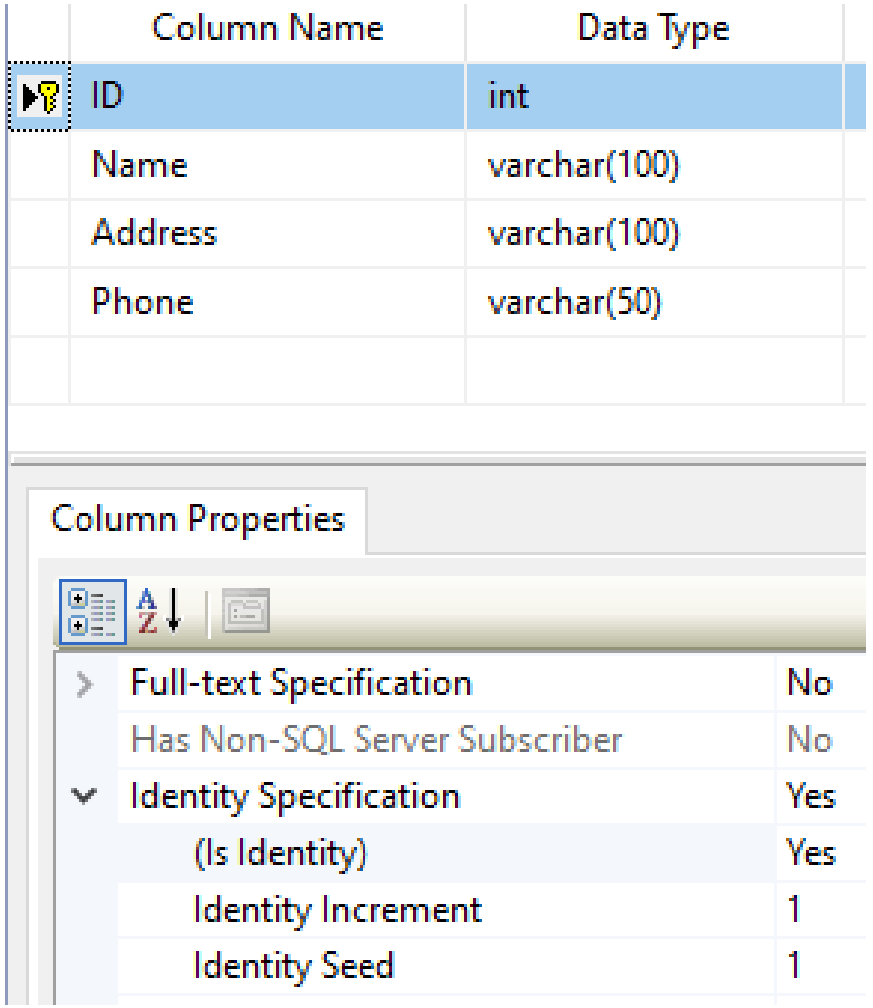
- **Create Table using CREATE TABLE** statement

  Syntax: CREATE TABLE <Table_Name>

  Ex:

```
CREATE TABLE AddressBook (
        ID int PRIMARY KEY IDENTITY (1, 1),
        Name varchar(100),
        Address varchar(100),
        Phone varchar(50)
)
```

- **Create Table using Object Explorer**
  - Right Click Table, choose New > Table
  - Enter Column Name and DataType
  - Set Primary Key and Auto Increment for ID Column
  - Save Table with name as AddressBook



EX Showing Connection, Command

```
string name = "Name 1";
string address = "Address 1";
string phone = "9801000000";

string connStr = "Data Source=AM;Initial Catalog=TestDb;User ID=sa;Password=12345";
SqlConnection conn = new SqlConnection(connStr);

string sql = "Insert into AddressBook values ('" + name + "','"
    + address + "','" + phone + "')";
SqlCommand cmd = new SqlCommand(sql, conn);
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();
```

EX – Reading Data with SqlDataAdapter & DataSet

```
string connStr = "Data Source=.;Initial Catalog=TestDb;User ID=sa;Password=123456";
SqlConnection conn = new SqlConnection(connStr);
string sql = "Select * from AddressBook";
SqlDataAdapter da = new SqlDataAdapter(sql, conn);

DataSet ds = new DataSet();
da.Fill(ds);

GridView1.DataSource = ds;
GridView1.DataBind();
```

EX Read Data Using SqlDataReader

```
string connStr = "Data Source=.;Initial Catalog=TestDb;User ID=sa;Password=123456";
SqlConnection conn = new SqlConnection(connStr);
string sql = "Select * from AddressBook";
SqlCommand cmd = new SqlCommand(sql, conn);

List<MyAddressBook> books = new List<MyAddressBook>();
conn.Open();
SqlDataReader dr = cmd.ExecuteReader();
while(dr.Read())
{
    MyAddressBook b = new MyAddressBook();
    b.ID = (int)dr[0];
    b.Name = dr["Name"].ToString();
    b.Address = dr["Address"].ToString();
    b.Phone = dr["Phone"].ToString();
    books.Add(b);
}
conn.Close();
GridView2.DataSource = books;
GridView2.DataBind();
```

```
class MyAddressBook
{
    1 reference
    public int ID { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string Address { get; set; }
    1 reference
    public string Phone { get; set; }
}
```

ASP.Net core 3.1 Crud Pperation with ADO.Net

- Ref Link

https://tutorialshelper.com/asp-net-core-3-1-crud-operation-with-ado-net/

Entity Framework(EF) Core

- Is a new version of Entity Framework after EF 6.x.
- It is open-source, lightweight, extensible and a cross-platform version of Entity Framework data access technology.
- Entity Framework is an Object/Relational Mapping (O/RM) framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.
- EF Core is intended to be used with .NET Core applications. However, it can also be used with standard .NET 4.5+ framework based applications.

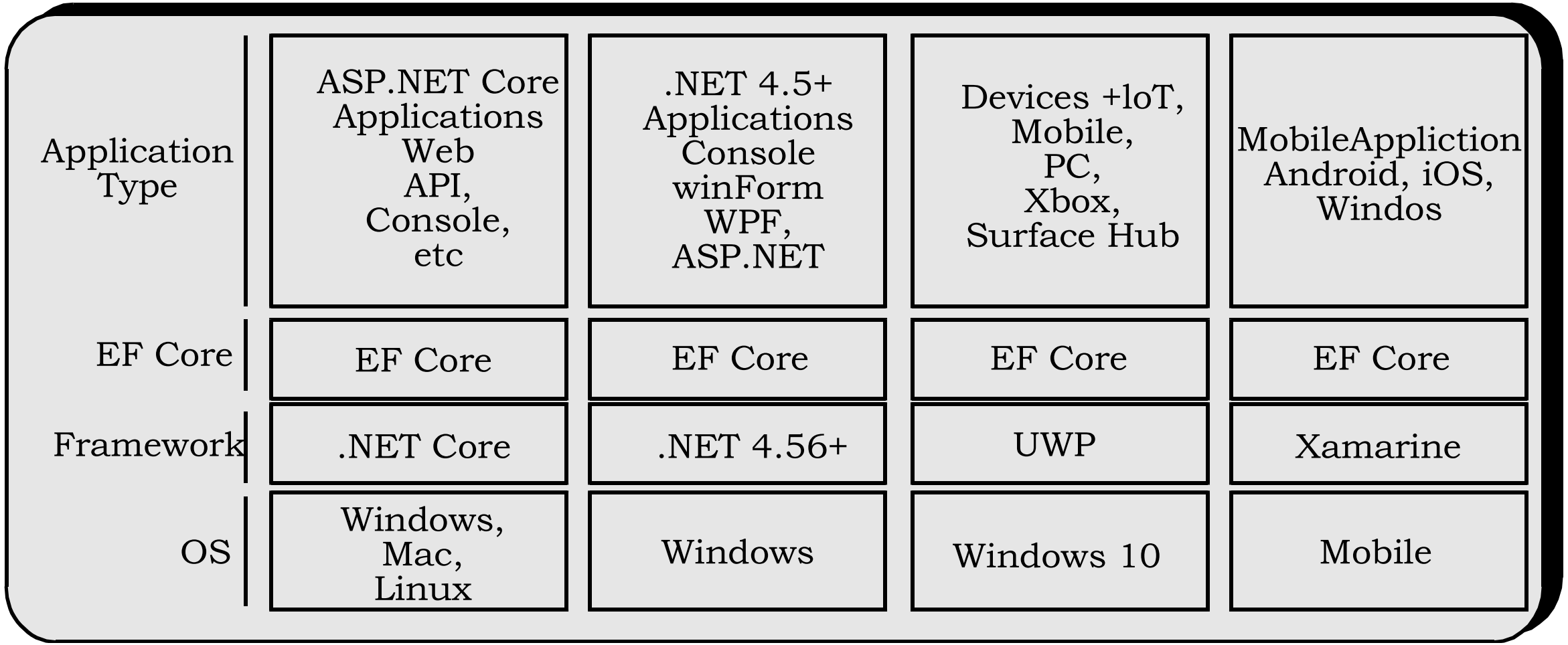


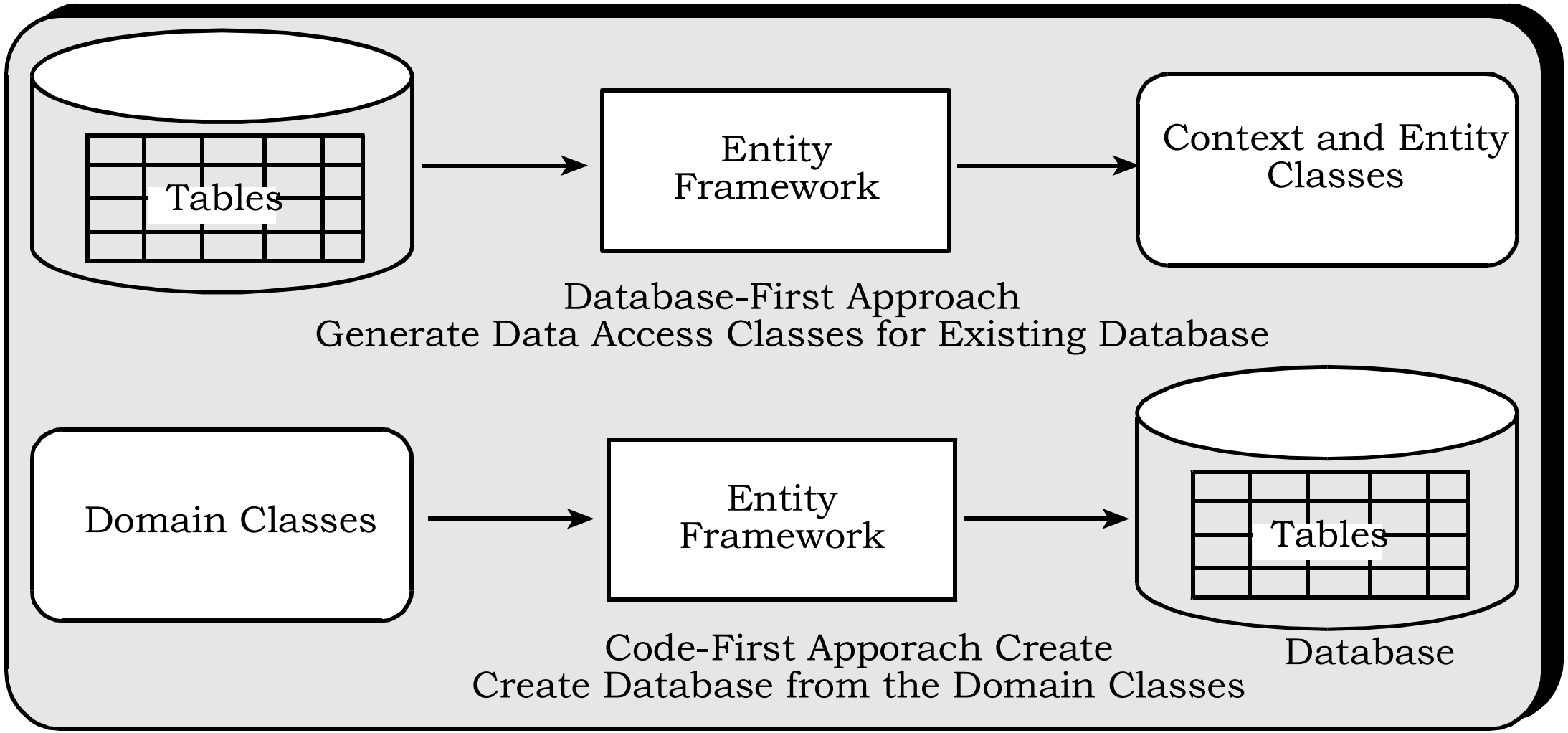
figure showing supported application types, .NET Frameworks and OSs.

EF Core Development Approaches

- EF Core supports two development approaches:

  (1) Code-First (2) Database-First.
- EF Core mainly targets the code-first approach and provides some support for the database-first.
- In the code-first approach, EF Core API creates the database and tables using migration based on the conventions and configuration provided in your domain classes. This approach is useful in Domain Driven Design (DDD).
- In the database-first approach, EF Core API creates the domain and context classes based on your existing database using EF Core commands. This has limited support in EF Core as it does not support visual designer or wizard.

EF Core Development Approaches



EF Core vs EF 6

- Entity Framework Core is the new and improved version of Entity Framework for .NET Core applications.
- EF Core continues to support the following features and concepts, same as EF 6.
  - DbContext & DbSet
  - Data Model
  - Querying using Linq-to-Entities
  - Change Tracking
  - SaveChanges
  - Migrations

EF Core vs EF 6

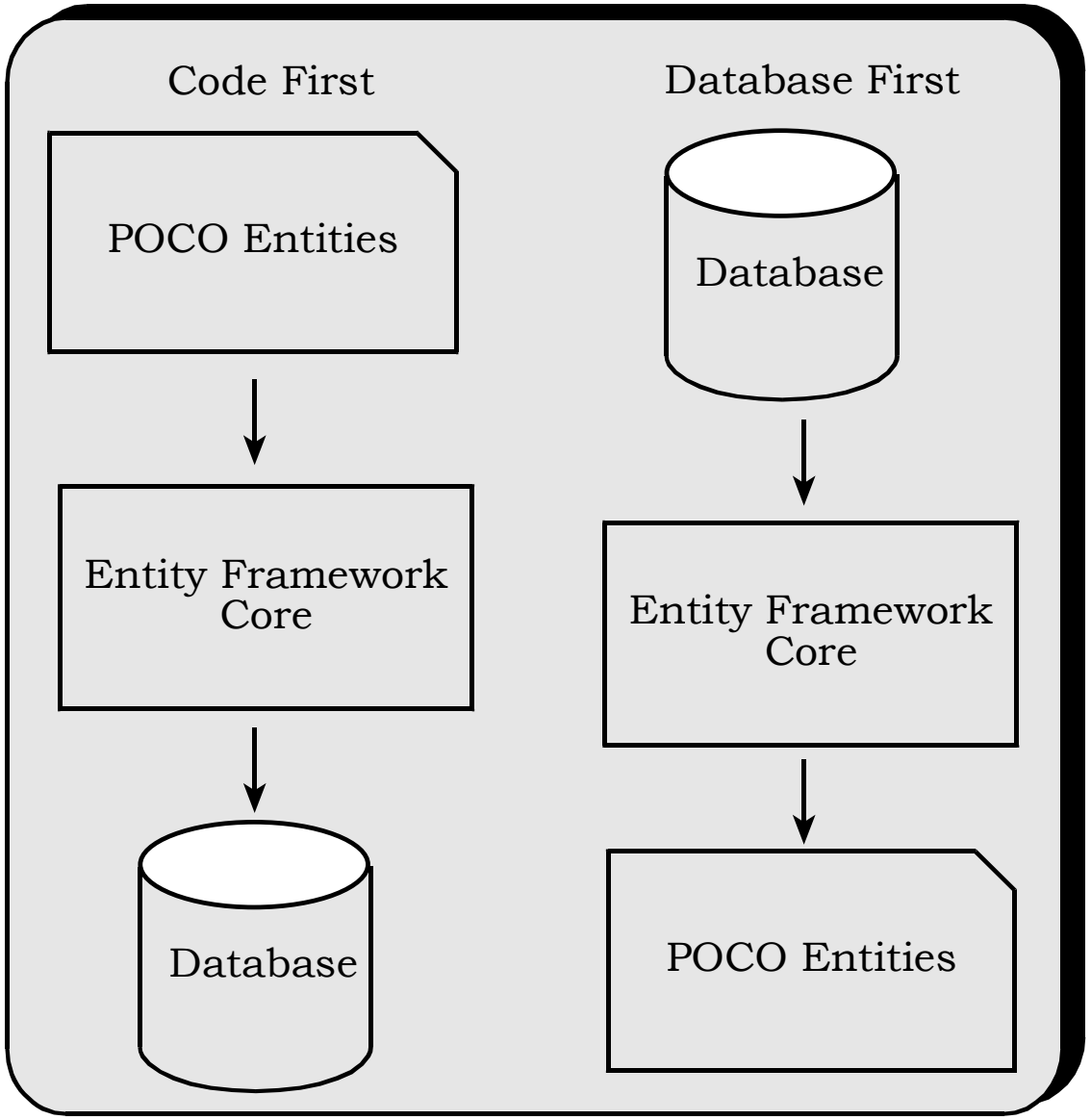
- EF Core includes the following new features which are not supported in EF 6.x:
  1. Easy relationship configuration
  2. Batch INSERT, UPDATE, and DELETE operations
  3. In-memory provider for testing
  4. Support for IoC (Inversion of Control)
  5. Unique constraints
  6. Shadow properties
  7. Alternate keys
  8. Global query filter
  9. Field mapping
  10. DbContext pooling
  11. Better patterns for handling disconnected entity graphs

EF Core Database Providers

- EF Core uses a provider model to access many different databases.
- EF Core includes providers as NuGet packages which you need to install.
- Below table lists database providers and NuGet packages for EF Core.

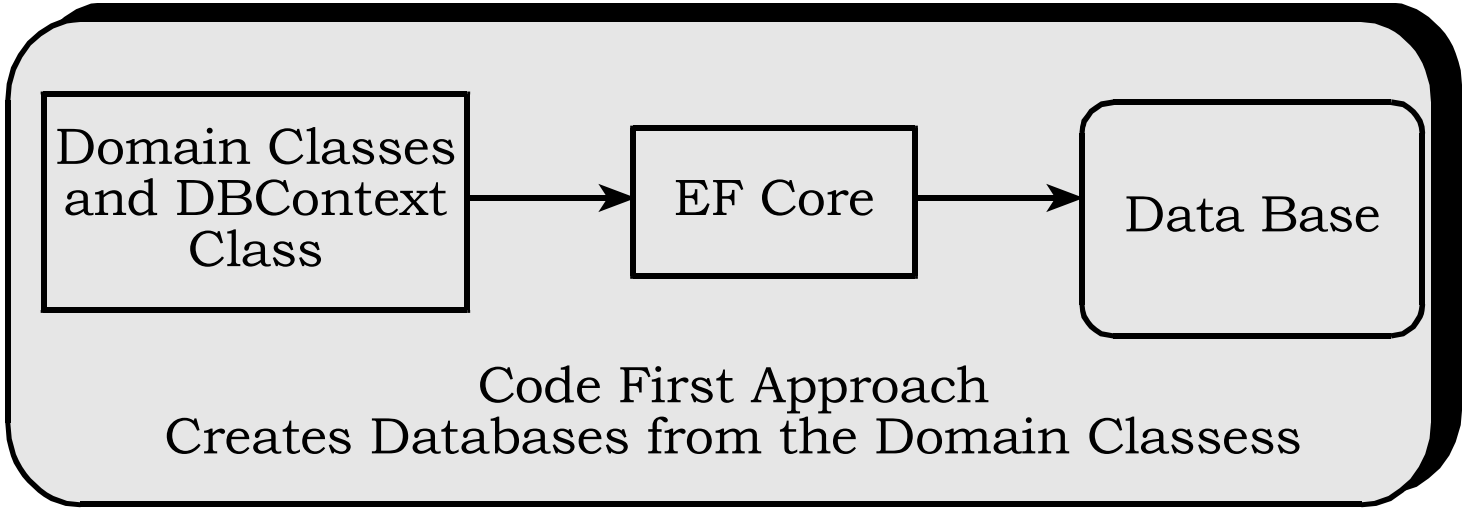
Database	NuGet Package
SQL Server	Microsoft.EntityFrameworkCore.SqlServer
MySQL	MySql.Data.EntityFrameworkCore
PostgreSQL	Npgsql.EntityFrameworkCore.PostgreSQL
SQLite	Microsoft.EntityFrameworkCore.SQLite
SQL Compact	EntityFrameworkCore.SqlServerCompact40
In-memory	Microsoft.EntityFrameworkCore.InMemory

EF Core Development Approaches



EF Core Code First Approach

- In the EF Core Code First Approach, first, we need to create our application domain classes such as Student, Branch, Address, etc. and a special class that derives from Entity Framework DbContext class.
- Then based on the application domain classes and DBContext class, the EF Core creates the database and related tables.



EF Core Code First Approach

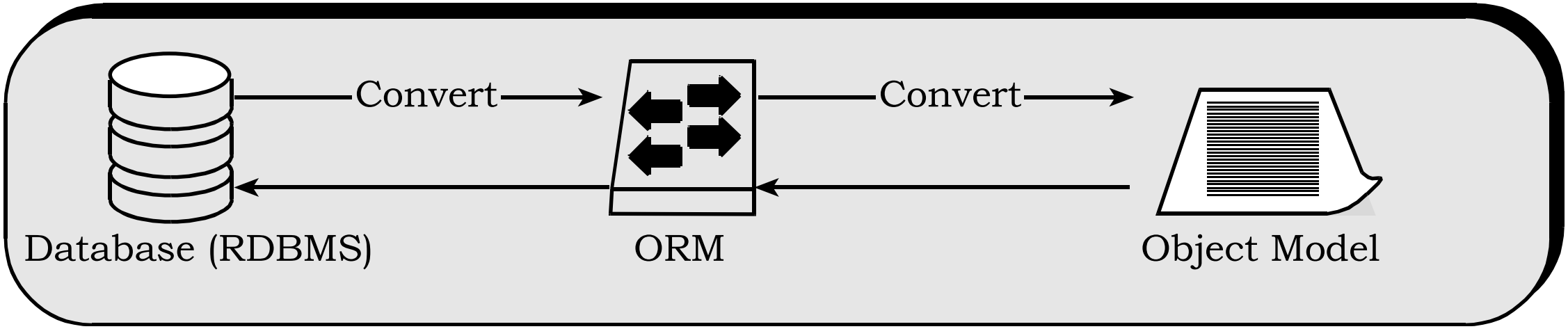
- In the code-first approach, the EF Core creates the database and tables using migration based on the default conventions and configuration. This approach is useful in Domain-Driven Design (DDD).
- Good option if you don't know the whole picture of your database as you can just update your Plain Old Class Object (POCO) entities and let EF sync changes to the database. In other words, you can easily add or remove features defined in your class without worrying about syncing your database using Migrations.
- You don't have to worry about your database as EF will handle the creation for you. In essence, database is just a storage medium with no logic.
- You will have full control over the code. You simply define and create POCO entities and let EF generate the corresponding Database for you. The downside is if you change something in your database manually, you will probably lose them because your code defines the database.
- It's easy to modify and maintain as there will be no auto-generated code.

Object Relational Mappers

- Essential parts of an ASP.NET MVC application is the architectural design. It's the Model-View-Controller (MVC) pattern. It show us the view of the application and the business logic within the application.
  - Model : designed to manage the business logic.
  - View : view that user can see.
  - Controller : manages the interaction between Model and View.
- A one of basic end point of project is the Database. We can prepare the database following many methods. The thing is, we have to access the DB from the next layer (Controller). In that point, object relational mapper(ORM) will come to the battle.

Object Relational Mappers

- An ORM is an application or system that support in the conversion of data within a relational database management system (RDBMS) and the object model that is necessary for use within object-oriented programming.



ADDING EF CORE TO AN APPLICATION

Install Entity Framework Core

- Entity Framework Core can be used with .NET Core or .NET 4.6 based applications. Here, you will learn to install and use Entity Framework Core .NET Core applications
- EF Core is not a part of .NET Core and standard .NET framework. It is available as a NuGet package.
- You need to install NuGet packages for the following two things to use EF Core in your application:
- 1. EF Core DB provider
- 2. EF Core tools

ADDING EF CORE TO AN APPLICATION

Install EF Core DB Provider

- EF Core allows us to access databases via the provider model. There are different EF Core DB providers available for the different databases. These providers are available as NuGet packages.
- First, install the NuGet package for the provider of database you want to access.
- For, MS SQL Server database,
  install Microsoft.EntityFrameworkCore.SqlServer NuGet package.
- To Install DB provider NuGet package:
  - Right click on the project in the Solution Explorer in Visual Studio
  - select Manage NuGet Packages.. (or select on the menu: Tools -> NuGet Package Manager -> Manage NuGet Packages For Solution).
  - search for Microsoft.EntityFrameworkCore.SqlServer and install

ADDING EF CORE TO AN APPLICATION

Install EF Core Tools

- Along with the DB provider package, you also need to install EF tools to execute EF Core commands. These make it easier to perform several EF Core-related tasks in your project at design time, such as migrations, scaffolding, etc.
- EF Tools are available as NuGet packages.
- To Install EF Core Tools:
  - Right click on the project in the Solution Explorer in Visual Studio
  - select Manage NuGet Packages.. (or select on the menu: Tools -> NuGet Package Manager -> Manage NuGet Packages For Solution).
  - search for Microsoft.EntityFrameworkCore.Tools and install

Data Models

- Entity Framework needs to have a model (Entity Data Model) to communicate with the underlying database. It builds a model based on the shape of your domain classes, the Data Annotations and Fluent API configurations.
- The EF model includes three parts: conceptual model, storage model, and mapping between the conceptual and storage models.
- In the code-first approach, EF builds the conceptual model based on your domain classes (entity classes), the context class and configurations.
- EF Core builds the storage model and mappings based on the provider you use. EF uses this model for CRUD (Create, Read, Update, Delete) operations to the underlying database.

Data Context

- The DbContext class is an integral part of Entity Framework. An instance of DbContext represents a session with the database which can be used to query and save instances of your entities to a database.
- DbContext is a combination of the Unit Of Work and Repository patterns.
- DbContext in EF Core allows us to perform following tasks:
  - Manage database connection
  - Configure model & relationship
  - Querying database
  - Saving data to the database
  - Configure change tracking
  - Caching
  - Transaction management

Create Database From Model Using Entity Framework Core And ASP.NET Core

1. Add these two NuGet packages to the project:

- EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools

2. In Models Folder Create a Class with Name as WebUser and add these lines of Codes

```
public class WebUser
{
    [Column("UserID")]
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public int UserID { get; set; }
    [Display(Name = "First Name")]
    [Required]
    [StringLength(25)]
    2 references
    public string FirstName { get; set; }
    [Required]
    [StringLength(25), MinLength(3)]
    [Display(Name = "Last Name")]
    1 reference
    public string LastName { get; set; }
    [EmailAddress]
    1 reference
    public string Email { get; set; }
}
```

Create Database From Model Using Entity Framework Core And ASP.NET Core

3. In Models Folder Create a custom DbContext class named AppDbContext and write the following code.

```
using Microsoft.EntityFrameworkCore;
namespace WebApplicationCoreS1.Models
{
    4 references
    public class AppDbContext : DbContext
    {
        0 references
        public AppDbContext(DbContextOptions <AppDbContext> options) : base(options)
        {

        }
        0 references
        public DbSet<WebUser> WebUsers { get; set; }
    }
}
```

Create Database From Model Using Entity Framework Core And ASP.NET Core

3. Build your project

4. Open the appsettings.json file and Add Database Connection string:

```
 "ConnectionStrings": {
    "DBConnectionString": "Data Source=.; Initial Catalog=DotNetCoreDBS; User Id=sa;
Password =123456"
  }
```

Create Database From Model Using Entity Framework Core And ASP.NET Core

5. open the Startup class and add this code to the ConfigureServices() method.

```
public void ConfigureServices(IServiceCollection services)
{
            services.AddControllersWithViews();
            services.AddDbContext<AppDbContext>(o =>
o.UseSqlServer(Configuration.GetConnectionString("DBConnectionString")));
}
```

The above code uses AddDbContext() method to register AppDbContext. Notice that the database connection string stored in the appsettings.json file is supplied to the UseSqlServer() method.

Create Database From Model Using Entity Framework Core And ASP.NET Core

6. Create Database using EnsureCreated() method

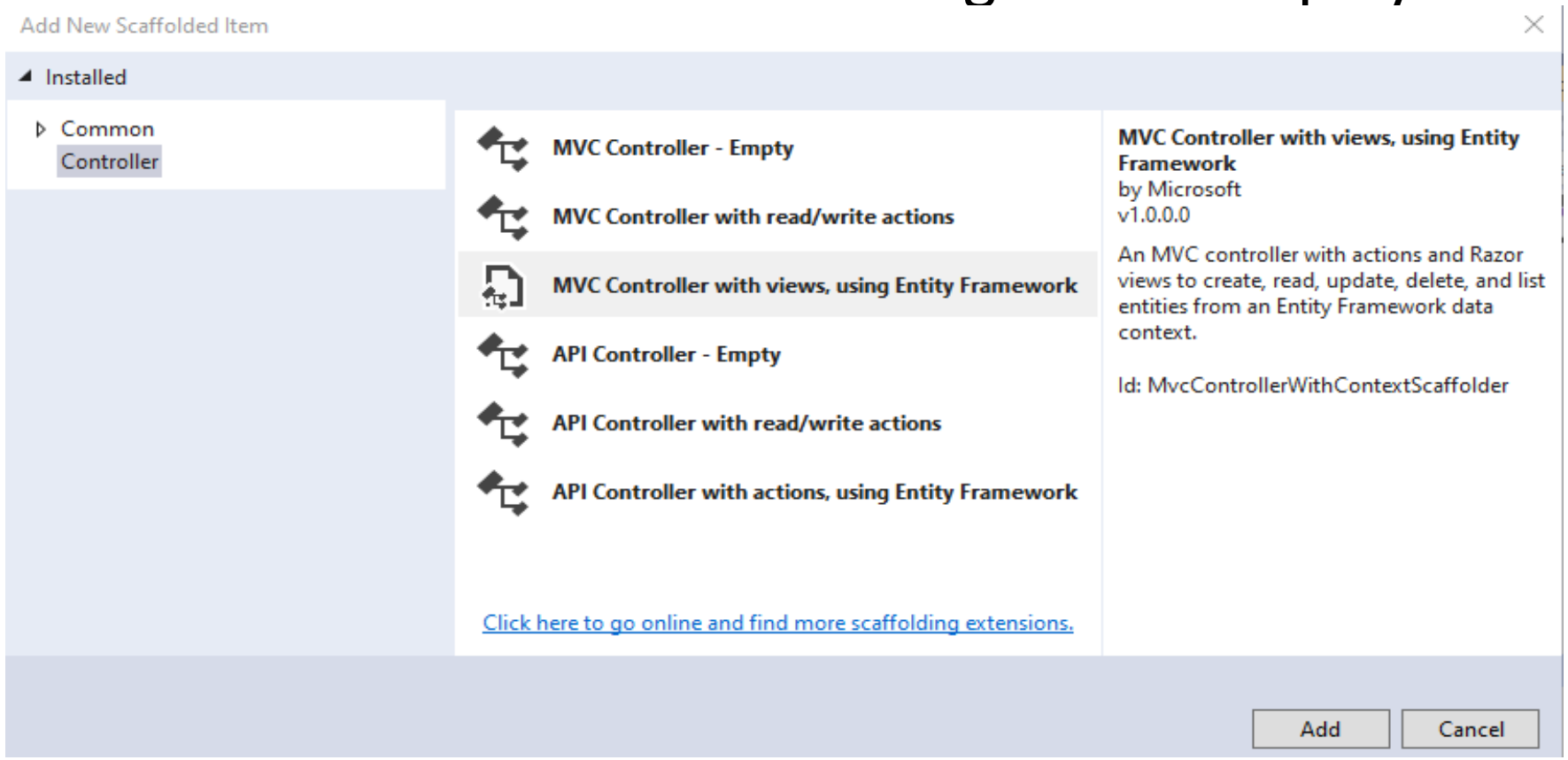
- EF Core model is ready, let's try to create the required database using EnsureCreated() method. This technique is a code based technique and works great for quick and simple database creation scenarios. If database is already exists, then no action is taken, otherwise database is created.
- Add following marked as bold in Configure Method

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, AppDbContext db)
{
        .
        db.Database.EnsureCreated();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
}
```

CRUD Operation Using Entity Framework Core

Create MVC Controller with views, using Entity Framework

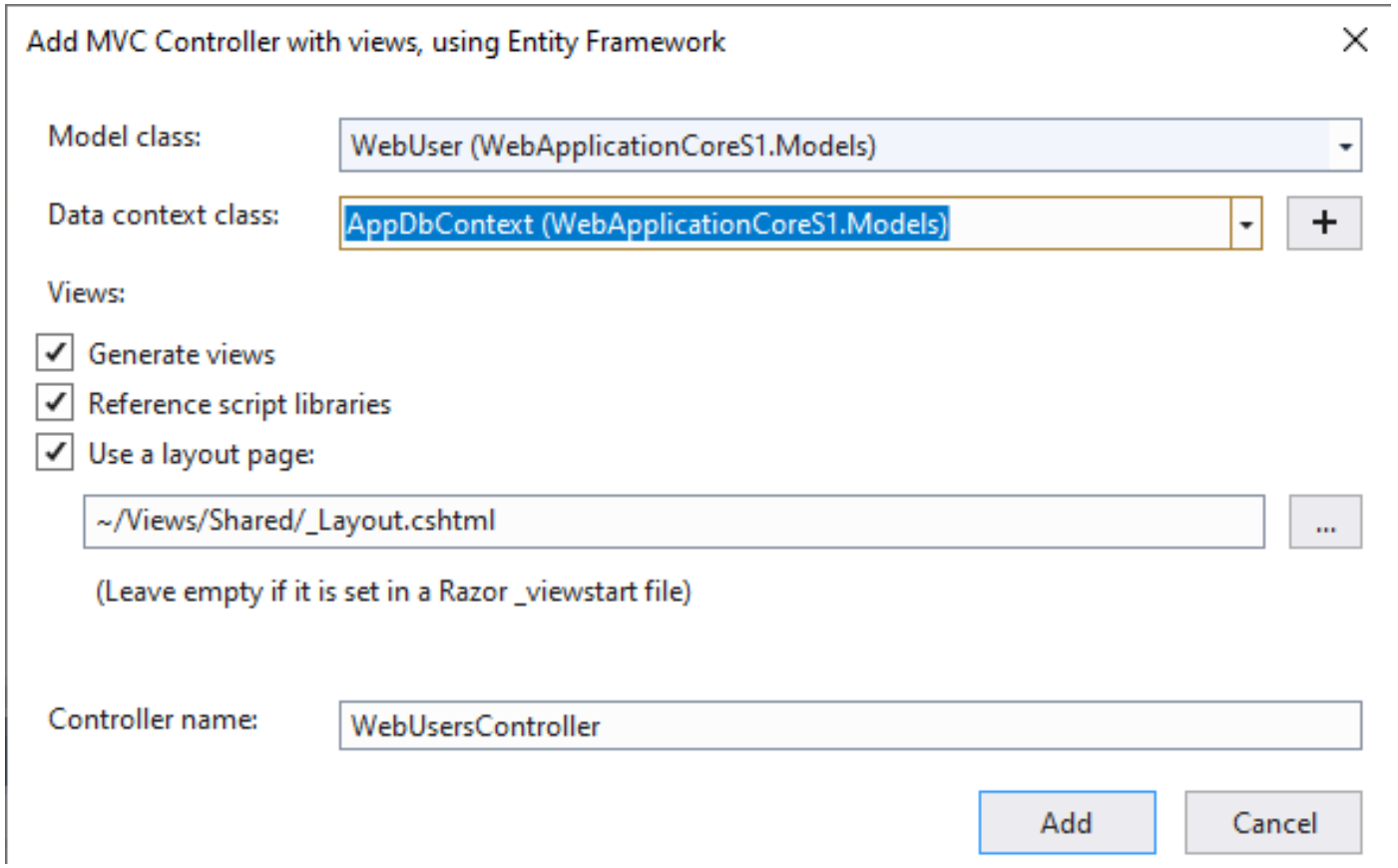
- Right-click on the controller folder, select add new item, and then select controller. Then this dialog will be displayed.



CRUD Operation Using Entity Framework Core

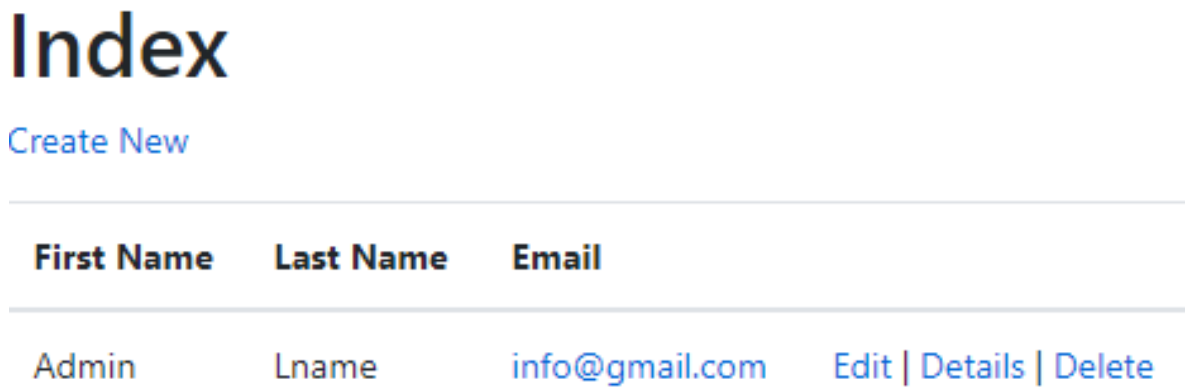
Create MVC Controller with views, using Entity Framework

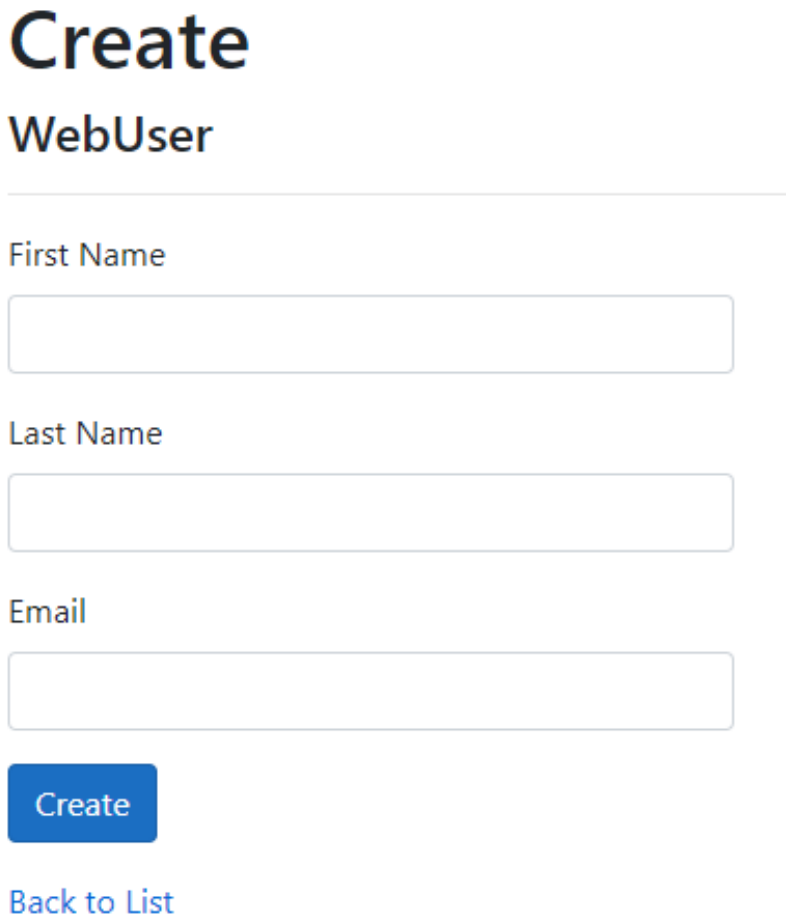
- Enter for Model Class, Data context class, Controller name as shown
- Tick Views as shown



CRUD Operation Using Entity Framework Core

- Review your generated code in controller and view pages
- Load your controller in your browser
  - https://localhost:44347/WebUsers
  - https://localhost:44347/WebUsers/Create
- Click on Edit, Details and Delete





Unit 6

State Management on ASP.NET Core Application

STATE MANAGEMENT ON STATELESS HTTP

- HTTP is a stateless protocol. So, HTTP requests are independent messages that don't retain user values or app states. We need to take additional steps to manage state between the requests.
- State can be managed in our application using several approaches.

Storage Approach	Description
Cookies	HTTP cookies. May include data stored using server-side app code.
Session state	HTTP cookies and server-side app code
TempData	HTTP cookies or session state
Query strings	HTTP query strings
Hidden fields	HTTP form fields
HttpContext	Server-side app code
Cache	Cache Server-side app code

SERVER-SIDE STRATEGIES: SESSION STATE, TEMPDATA, USING HTTPCONTEXT

Session State

- Session state is an ASP.NET Core mechanism to store user data while the user browses the application.
- It uses a store maintained by the application to persist data across requests from a client. We should store critical application data in the user's database and we should cache it in a session only as a performance optimization if required.
- ASP.NET Core maintains the session state by providing a cookie to the client that contains a session ID. The browser sends this cookie to the application with each request. The application uses the session ID to fetch the session data.

SESSION STATE

While working with the Session state, we should keep the following things in mind:

- A Session cookie is specific to the browser session
- When a browser session ends, it deletes the session cookie
- If the application receives a cookie for an expired session, it creates a new session that uses the same session cookie
- An Application doesn’t retain empty sessions
- The application retains a session for a limited time after the last request. The app either sets the session timeout or uses the default value of 20 minutes
- Session state is ideal for storing user data that are specific to a particular session but doesn’t require permanent storage across sessions

A Session State Example

- We need to configure the session state before using it in our application. This can be done in the ConfigureServices() method in the Startup.cs class:

```
services.AddSession();
```

- The order of configuration is important and we should invoke the UseSession() before invoking UseMVC().
- Let's create a controller with endpoints to set and read a value from the session:

```
public class WelcomeController : Controller {
    public IActionResult Index()
    {
        HttpContext.Session.SetString("Name", "John");
        HttpContext.Session.SetInt32("Age", 32);
        return View();
    }
    public IActionResult Get() {
        User u = new User()
        {
            Name = HttpContext.Session.GetString("Name"),
            Age = HttpContext.Session.GetInt32("Age").Value
        };
        return View(u);
    }
}
```

A Session State Example

- The Index() method sets the values into session and Get() method reads the values from the session and passes them into the view.
- Let’s auto-generate a view to display the model values by right-clicking on the Get() method and using the “Add View” option.
- Now let’s run the application and navigate to /welcome.
- This will set the session values.
- Now let’s navigate to /welcome/get:

TempData

- TempData property which can be used to store data until it is read.
- TempData is particularly useful when we require the data for more than a single request. We can access them from controllers and views.
- TempData is implemented by TempData providers using either cookies or session state.
- Let’s create a controller with three endpoints. In the First() method, let’s set a value into TempData. Then let’s try to read it in Second() and Third() methods:

```
public class TempDataController : Controller {
    public IActionResult First() {
        TempData["UserId"] = 101;
        return View();
    }
    public IActionResult Second() {
        var userId = TempData["UserId"] ?? null;
        return View();
    }
    public IActionResult Third() {
        var userId = TempData["UserId"] ?? null;
        return View();
    }
}
```

TempData

- Now let’s run the application by placing breakpoints in the Second() and Third() methods.
- We can see that the TempData is set in the First() request and when we try to access it in the Second() method, it is available. But when we try to access it in the Third() method, it is unavailable as is retains its value only till its read.
- Now let’s move the code to access TempData from the controller methods to the views.

Let’s create a view for the Second() action method:

```
@{
        ViewData["Title"] = "Second";
        var userId = TempData["UserId"].ToString();
}
<h1>Second</h1>
User Id : @userId
```

Similarly, let’s create a view for the Third() action method:

```
@{
        ViewData["Title"] = "Third";
        var userId= TempData["UserId"].ToString();
}
<h1>Third</h1>
User Id : @userId
```

Let’s run the application and navigate to /first, /second and /third

- We can see that TempData is available when we read it for the first time and then it loses its value. Now, what if we need to persist the value of TempData even after we read it?
- We have two ways to do that:
  - **TempData.Keep()/TempData.Keep(string key):** This method retains the value corresponding to the key passed in TempData. If no key is passed, it retains all values in TempData.
  - **TempData.Peek(string key):** This method gets the value of the passed key from TempData and retains it for the next request.
- Let's slightly modify our second view with one of these methods:

```
var userId = TempData["UserId"].ToString();
TempData.Keep();
// OR
var userId = TempData.Peek("UserId").ToString();
```

- Now let's run the application and navigate to /first, /second and /third.
- We can see that the TempData value persists in the third page even after its read on the second page.

Using HttpContext

- A HttpContext object holds information about the current HTTP request. The important point is, whenever we make a new HTTP request or response then the Httpcontext object is created. Each time it is created it creates a server current state of a HTTP request and response.
- It can hold information like: Request, Response, Server, Session, Item, Cache, User's information like authentication and authorization and much more.
- As the request is created in each HTTP request, it ends too after the finish of each HTTP request or response.

Example to Check request processing time using HttpContext class

- This example check the uses of the HttpContext class. In the global.aspx page we know that a BeginRequest() and EndRequest() is executed every time before any Http request. In those events we will set a value to the context object and will detect the request processing time.

```
protected void Application_BeginRequest(object sender, EventArgs e) {
        HttpContext.Current.Items.Add("Begintime", DateTime.Now.ToLongTimeString());
}
protected void Application_EndRequest(object sender, EventArgs e) {
     TimeSpan diff = Convert.ToDateTime(DateTime.Now.ToLongTimeString()) -
        Convert.ToDateTime(HttpContext.Current.Items["Begintime"].ToString());
}
```

Example to access current information using HttpContext class

```
protected void Page_Load(object sender, EventArgs e){
      Response.Write("Request URL"+  HttpContext.Current.Request.Url)
     Response.Write("Number of Session variable" +
                         HttpContext.Current.Session.Count);
      Response.Write("current Timestamp" + HttpContext.Current.Timestamp);
      Response.Write("Object in Application level " +
                         HttpContext.Current.Application.Count);
      Response.Write("Is Debug Enable in current Mode?" +
                         HttpContext.Current.IsDebuggingEnabled);
 }
```

CACHE CLIENT-SIDE STRATEGIES

- COOKIES,
- QUERY STRINGS,
- HIDDEN FIELDS

Cookies

Reading Cookie

```
//read cookie from IHttpContext Accessor
string cookieValueFromContext =
httpContextAccessor.HttpContext.Request.Cookies["key"];

//read cookie from Request object
string cookieValueFromReq = Request.Cookies[“key"];
```

Remove Cookie

```
Response.Cookies.Delete(key);
```

Cookies

Writing cookie

- In this example, SetCookie method show how to write cookies.
- CookieOption is available to extend the cookie behavior.

```
public void SetCookie(string key, string value, int? expireTime)  {
  CookieOptions option = new CookieOptions();
  if (expireTime.HasValue)
    option.Expires = DateTime.Now.AddMinutes(expireTime.Value);
  else
       option.Expires = DateTime.Now.AddMilliseconds(10);
  Response.Cookies.Append(key, value, option);
}
```

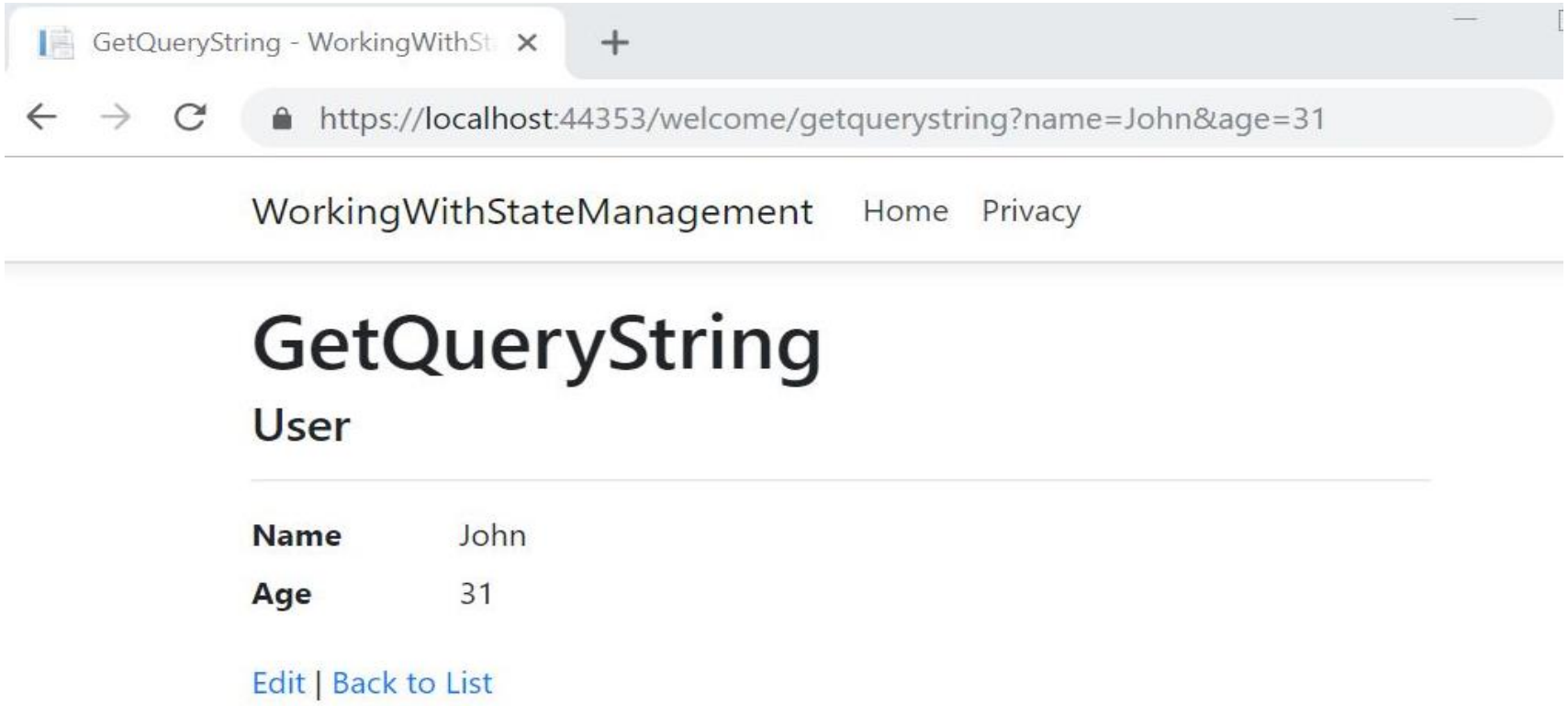
Query strings

- We can pass a limited amount of data from one request to another by adding it to the query string of the new request. This is useful for capturing the state in a persistent manner and allows the sharing of links with the embedded state.

```
public IActionResult GetQueryString(string name, int age) {
    User newUser = new User()
    {
        Name = name,
        Age = age
    };
    return View(newUser);
}
```

Query strings

- Now let’s invoke this method by passing query string parameters:
- /welcome/getquerystring?name=John&age=31



Query strings

- We can retrieve both the name and age values from the query string and display it on the page.
- As URL query strings are public, we should never use query strings for sensitive data.
- In addition to unintended sharing, including data in query strings will make our application vulnerable to Cross-Site Request Forgery (CSRF) attacks, which can trick users into visiting malicious sites while authenticated. Attackers can then steal user data or take malicious actions on behalf of the user.

Hidden Fields

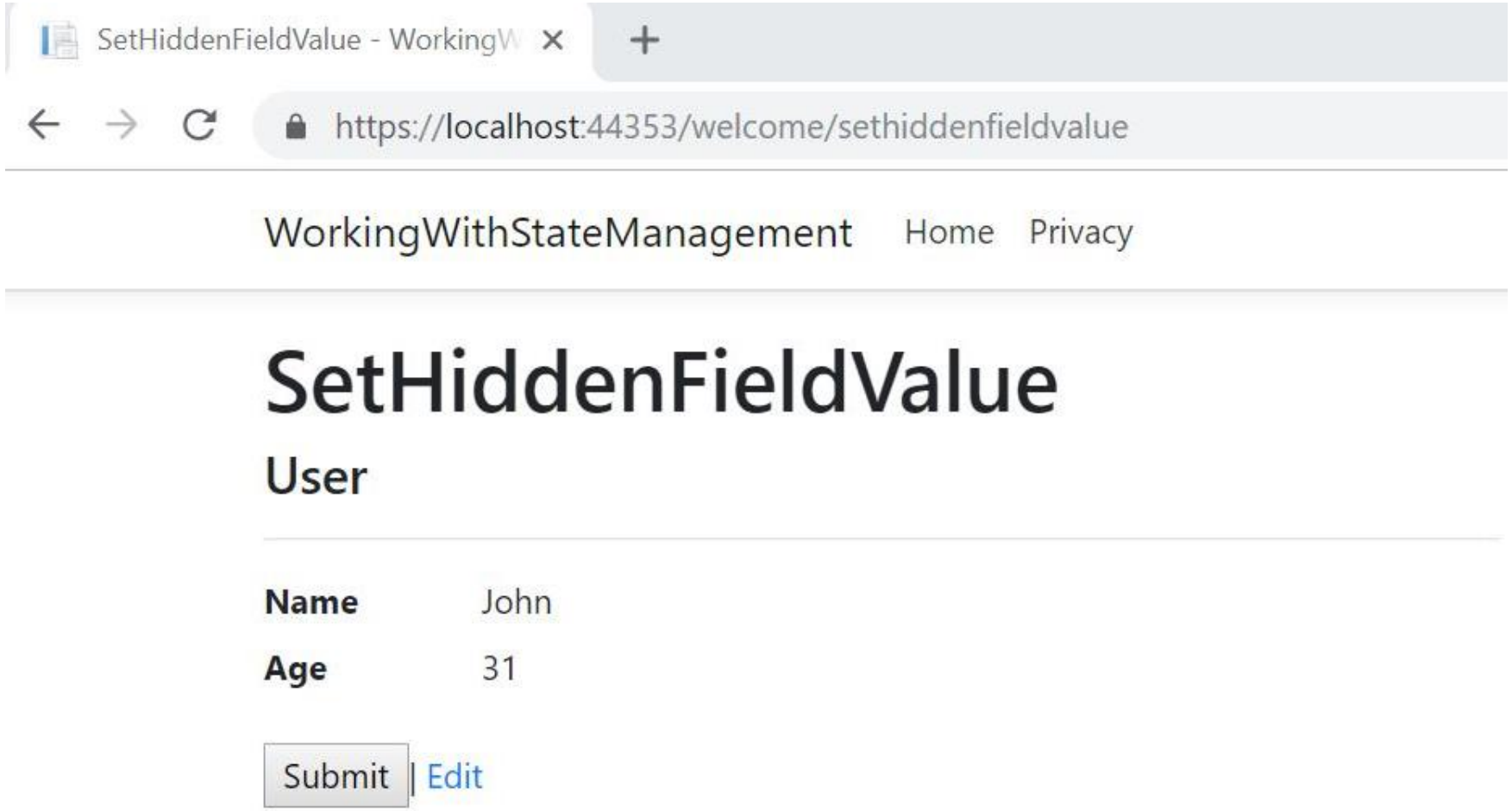
- We can save data in hidden form fields and send back in the next request.
- Sometimes we require some data to be stored on the client side without displaying it on the page. Later when the user takes some action, we'll need that data to be passed on to the server side. This is a common scenario in many applications and hidden fields provide a good solution for this.
- Let's add two methods in our WelcomeController:

```
[HttpGet]
public IActionResult SetHiddenFieldValue() {
    User newUser = new User() {
        Id = 101,  Name = "John",  Age = 31
    };
    return View(newUser);
}
[HttpPost]
public IActionResult SetHiddenFieldValue(IFormCollection keyValues) {
    var id = keyValues["Id"];
    return View();
}
```

Hidden Fields

- The GET version of theSetHiddenValue() method creates a user object and passes that into the view.
- We use the POST version of the SetHiddenValue() method to read the value of a hidden field Id from FormCollection.
- In the View, we can create a hidden field and bind the Id value from Model:
  - @Html.HiddenFor(model =>model.Id)
- Then we can use a submit button to submit the form:
  - <input type="submit" value="Submit" />
- Now let’s run the application and navigate to /Welcome/SetHiddenFieldValue

Hidden Fields



Hidden Fields

- On inspecting the page source, we can see that a hidden field is generated on the page with the Id as the value: <input id="Id" name="Id" type="hidden" value="101">
- Now click the submit button after putting a breakpoint in the POST method. We can retrieve the Id value from the FormCollection

- Since the client can potentially tamper with the data, our application must always revalidate the data stored in hidden fields.

Discussion Exercise

1. Write about the State Management Strategies.
2. What is Session State? Show with an example to manage session state in ASP.NET Core.
3. Show the difference between TempData and Using HttpContext with suitable example.
4. How do you manage to handle state with client side strategies?

Unit 7

Client-Side Development in ASP.NET Core

COMMON CLIENT-SIDE WEB TECHNOLOGIES

- ASP.NET Core applications are web applications and they typically rely on client-side web technologies like HTML, CSS, and JavaScript.
- By separating the content of the page (the HTML) from its layout and styling (the CSS), and its behavior (via JavaScript), complex web apps can leverage the Separation of Concerns principle.
- While HTML and CSS are relatively stable, JavaScript, by means of the application frameworks and utilities developers work with to build web-based applications.
- We will discuss on JavaScript, jQuery, Angular SPA, React, Vue.

Javascript

- JavaScript is a dynamic, interpreted programming language of the web.
- Just like CSS, it's recommended to organize JavaScript into separate files, keeping it separated as much as possible from the HTML found on individual web pages or application views.
- With Javascript, we can perform following:
  - Selecting an HTML element and retrieving and/or updating its value.
  - Decision Making, complex calculations, Validate Data, Animate and Add Effects
  - Interaction with properties of page object
  - React to events
  - Querying a Web API for data.
  - Sending a command to a Web API (and responding to a callback with its result).
  - Performing validation.

Quick Example Review on Javascript

Example

```
<HTML>
      <TITLE> Displaying Text </TITLE>
      <BODY>
            <script>
                  document.write("<h1> Hello Good Day </H1>");
                  document.write("<H3> Best of Luck. </H3>");
                 alert("Hello");
            </script>
      </BODY>
</HTML>
```

Example2

```
<script type="text/javascript">
 s1=12;
 s2=28;
 sum=s1+s2;
 diff=s1-s2;
 mult=s1*s2;
 div=s1/s2;
 document.write("<br>Sum: "+sum);
 document.write("<br>Difference: "+diff);
 document.write("<br>Multiply: "+mult);
 document.write("<br>Division: "+div);
</script >
```

Example3 – JavaScript Array

```
<script>
var sports = new Array( "Football", "Tennis", "Cycling”);
document.write(sports[0]);
document.write(sports[1]);
document.write(sports[2]);
var count = sports.length;
// loop through array elements
for(i=0; i< count; i++)
{
     document.write("<br>Index " + i + " is " +  sports[i]);
}
</script>
```

Example 4 – JavaScript String

- Used for storing and manipulating text
- Zero or more characters within quotes.

```
/* String : J  a  v  a  s  c   r   i    p   t
Index  : 0  1  2  3  4  5   6   7   8    9    */
var myText = "Javascript";
document.write("<br>" + myText.length);
document.write("<br>" + myText.charAt(4));
document.write("<br>" + myText.indexOf("va"));
document.write("<br>" + myText.substr(0,4));
document.write("<br>" + myText.toUpperCase());
document.write("<br>" + myText.toLowerCase());
```

Example 5 – JavaScript Function

```
<script>
	// function defination
	function callme() {
		alert("Hello there");
	}
	function f3(n1, n2) {
		var sum = n1 + n2;
		return sum;
	}
	callme();	// calling a function
	callme();
	var returned_sum = f3(10, 20);
	document.write(returned_sum);
	document.write(f3(20, 30));
 </script>
```

Example 5 – JavaScript Date

```
<html>
<body>
	<h1>Demo: Current Date</h1>
	<p id="p1"></p>
	<p id="p2"></p>
	<script>
		document.getElementById("p1").innerHTML = Date();
		var currentDate = new Date();
		document.getElementById("p2").innerHTML = currentDate;
	</script>
</body>
</html>
```

Javascript Events

- Interaction with HTML page and HTML elements is handled through events. Events can be page loads, button click, pressing a key, select data in form controls, focus on control, mouse over and mouse out on any element, etc.
- Events are a part of the Document Object Model (DOM) and every HTML element contains a set of events which can trigger JavaScript Code.
- We can categories Javascript events on:
  - Document Level Events - onload, onunload
  - Form Level Events - Onsubmit, Onreset,Onchange, onselect, onblur, onfocus
  - Keyboard Events - Onkeydown, onkeypress, onkeyup
  - Mouse Events - Onclick, ondblclick, onmouseover, onmouseout

Events –Example

```
<html>
<head>
<script>
        function callme() {
             alert("Hello");
             document.write("Hello");
        }
</script>
</head>
<body>
    <form>
       <input type = "button" onclick = "callme()" value = "Click Me">
    </form>
</body>
</html>
```

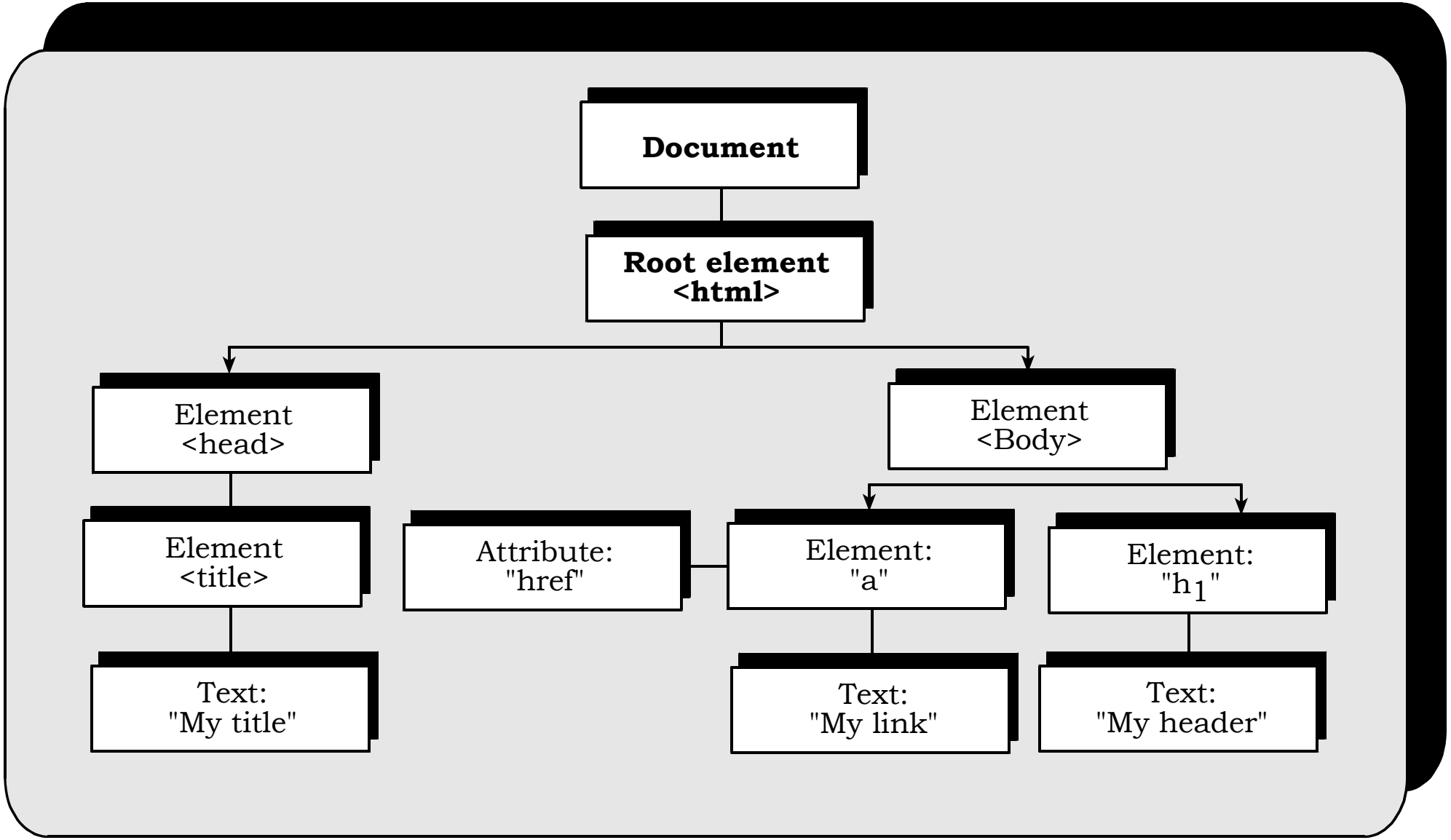
Events –Example

```
<html>
<head>
  <script type="text/javascript">
      function over() {
        alert("Mouse Over");
      }
      function out() {
        alert("Mouse Out");
      }
  </script>
</head>
<body>
  <div onmouseover="over()" onmouseout="out()">
    <h2> This is inside the division </h2>
  </div>
</body>
</html>
```

HTML DOM

- When a web page is loaded, browser creates a Document Object Model of the page. With the object model, JavaScript can do following:
  - modify all the HTML elements and attributes in the
  - change all the CSS styles in the page
  - Add remove existing HTML elements and attributes
  - add new HTML elements and attributes
  - react to all existing HTML events in the page
  - create new HTML events in the page

The HTML DOM model is constructed as a tree of Objects:



- In the DOM, all HTML elements are defined as objects. Below example changes the content (the innerHTML) of the <p> element with id="demo“ and getElementById is a method and innerHTML is a property

```
<body>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = "Hello World!";
  </script>
</body>
```

Changing HTML Elements

Property	Description
element.innerHTML = new htm content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style an HTML element
Method	**Description**
element.setAttribute(attribute, value)	Change the attribute of an HTML element

Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByclassName(name)	Find elements by class name

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

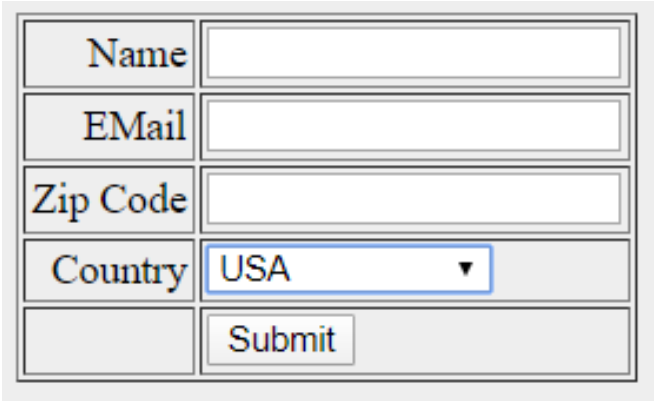
EXAMPLE

```
<html>
<head>
<script>
	var btn = document.querySelector('button');
	function random(number) {
		return Math.floor(Math.random() * (number+1));
	}
	function changeBgColor() {
		var rCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ') ';
	document.body.style.backgroundColor = rCol;
	}
</script>
</head>
<body>
	<button onclick= "changeBgColor()">Change color</button>
</body>
</html>
```

Form Validation

- JavaScript provides a way to validate form's data on the client's computer before sending it to the web server.
- Form validation generally performs two functions.
  - Basic Validation – check all the mandatory fields are filled in.
  - Data Format Validation – data entered checked for correct form and value with appropriate logic to test correctness of data.

```
<html>  <head> <title>Form Validation</title>
    <script type = "text/javascript">
      <!--         // Form validation code will come here.        //-->
    </script></head>
  <body>
    <form action = "next_page" name = "myForm" onsubmit = "return(validate());">
      <table cellspacing = "2" cellpadding = "2" border = "1">
        <tr>  <td align = "right">Name</td>  <td><input type = "text" name = "Name" /></td>  </tr>
        <tr>  <td align = "right">EMail</td>  <td><input type = "text" name = "EMail" /></td>    </tr>
        <tr>  <td align = "right">Zip Code</td>   <td><input type = "text" name = "Zip" /></td>   </tr>
        <tr>   <td align = "right">Country</td>  <td>
            <select name = "Country">
           <option value = "1">USA</option>
              <option value = "2">UK</option>
              <option value = "3">Nepal</option>
            </select>
          </td>   </tr>
        <tr>  <td align = "right"></td>  <td><input type = "submit" value = "Submit" /></td>   </tr>
      </table>
    </form> </body> </html>
```



```
<script type = "text/javascript">
   function validate() {
      if( document.myForm.Name.value == "" ) {
        alert( "Please provide your name!" );   document.myForm.Name.focus() ;    return false;
      }
      if( document.myForm.EMail.value == "" ) {
        alert( "Please provide your Email!" );   document.myForm.EMail.focus() ;   return false;
      }
      if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5 ) {
        alert( "Please provide a zip in the format #####." );   document.myForm.Zip.focus() ;   return false;
      }
      if( document.myForm.Country.value == "-1" ) {
        alert( "Please provide your country!" );     return false;
      }
      return( true );
    }
</script>
```

jQuery

- jQuery is a fast, small and feature-rich JavaScript library included in a single .js file.
- It provides many built-in functions using which developers can accomplish various tasks easily and quickly.
- Some of the jQuery important features are:
  - DOM Selection: jQuery provides Selectors to retrieve DOM element based on different criteria like tag name, id, css class name, attribute name, value, nth child in hierarchy etc.
  - DOM Manipulation: You can manipulate DOM elements using various built-in jQuery functions. For example, adding or removing elements, modifying html content, css class etc.

jQuery

- Some of the jQuery important features are:
  - Special Effects: You can apply special effects to DOM elements like show or hide elements, fade-in or fade-out of visibility, sliding effect, animation etc.
  - Events: jQuery library includes functions which are equivalent to DOM events like click, dblclick, mouseenter, mouseleave, blur, keyup, keydown etc. These functions automatically handle cross-browser issues.
  - Ajax: jQuery also includes easy to use AJAX functions to load data from servers without reloading whole page.
  - Cross-browser support: jQuery library automatically handles cross-browser issues, so the user does not have to worry about it.

Advantages of jQuery

- Easy to learn: jQuery is easy to learn because it supports same JavaScript style coding.
- Write less do more: jQuery provides a rich set of features that increase developers' productivity by writing less and readable code.
- Excellent API Documentation: jQuery provides excellent online API documentation.
- Cross-browser support: jQuery provides excellent cross-browser support without writing extra code.
- Unobtrusive: jQuery is unobtrusive which allows separation of concerns by separating html and jQuery code.

Getting Started with jQuery

- You can start writing jQuery code on any of the editor like notepad, SublimeText, Visual Studio. it's time to use jQuery.
- There are several ways to start using jQuery on your web site. You can:
  - Download the jQuery library from jQuery.com
  - Include jQuery from a CDN, like Google
- There are two versions of jQuery available for downloading:
  - Production version - this is for your live website because it has been minified and compressed
  - Development version - this is for testing and development (uncompressed and readable code)
- The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section)

Getting Started with jQuery

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
```

jQuery Syntax

- The jQuery syntax is used to select HTML elements and perform some **action** on those element(s). Basic syntax is: $(*selector*).*action*()
  - A $ sign to define/access jQuery
  - A (selector) to "query (or find)" HTML elements
  - A jQuery action() to be performed on the element(s)

Examples:

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test".

The Document Ready Event

All jQuery methods in are inside a document ready event.

```
$(document).ready(function(){
  // jQuery methods go here...
});
```

Example: jQuery Element Selector to hide all paragraphs

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
 $(document).ready(function(){
   $("button").click(function(){
     $("p").hide();
   });
 });
</script>
</head>
<body>
 <h2>This is a heading</h2>
 <p>This is a paragraph.</p>
 <p>This is another paragraph.</p>
 <button>Click me to hide paragraphs</button>
</body>
</html>
```

Example: Using id Selector - with id=test will be hidden on button click

```
$(document).ready(function(){
  $("button").click(function(){
    $("#test").hide();
  });
});
```

Example: Using .class Selector with class=test will be hidden on button click

```
$(document).ready(function(){
  $("button").click(function(){
    $(".test").hide();
  });
});
```

More Examples of jQuery Selectors

Syntax	Description
$(this)	Current HTML element
$("p")	All <p> elements
$("p.intro")	All <p> elements with class="intro"
$(".intro")	All elements with class="intro"
$("#intro")	The first element with id="intro"
$("ul li:first")	The first element of each
$("[href$='.jpg']")	All elements with an href attribute that ends with ".jpg"
$("div#intro .head")	All elements with class="head" inside a element with id="intro"

jQuery Events

- An event represents the precise moment when something happens.
- An event can be moving a mouse over an element, selecting a radio button, clicking on an element, etc
- The term "fires/fired" is often used with events. Example: "The keypress event is fired, the moment you press a key".
- Here are some common DOM events:

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mousenter	keyup	focus	scroll
mouseleave		blur	unload

jQuery Syntax for Events

Event Method	Description
$(selector).click(function)	Invokes a function when the selected elements are clicked
$(selector).dblclick(function)	Invokes a function when the selected elements are double-clicked
$(selector).focus(function)	Invokes a function when the selected elements receive the focus
$(selector).mouseover(function)	Invokes a function when the mouse is over the selected elements
$(selector).keypress(function)	Invokes a function when a key is pressed inside the selected elements

jQuery Event Methods

Example: jQuery Event Methods

```
$(document).ready(function(){

  $("p").click(function(){
    $(this).hide();
  });

  $("p").dblclick(function(){
    $(this).hide();
  });

  $("#id1").hover(function(){
alert("You hover on id1").
  });

});
```

jQuery Effects

Event Method	Description
$(selector).hide()	Hide selected elements
$(selector).show()	Show selected elements
$(selector).toggle()	Toggle (between hide and show) selected elements
$(selector).slideDown()	Slide-down (show) selected elements
$(selector).slideUp()	Slide-up (hide) selected elements
$(selector).slideToggle()	Toggle slide-up and slide-down of selected elements
$(selector).fadeIn()	Fade in selected elements
$(selector).fadeOut()	Fade out selected elements
$(selector).fadeTo()	Fade out selected elements to a given opacity
$(selector).fadeToggle()	Toggle between fade in and fade out

jQuery Effects

EX – Fade Paragraph id1 with 50% opacity when btnFade is clicked

```
<script>
      $("#btnFade").click(function(){
            $("#id1").fadeTo("slow", 0.5);
       });
</script>
```

LEGACY WEB APPS WITH JQUERY

- Although ancient by JavaScript framework standards, jQuery continues to be a commonly used library for working with HTML/CSS and building applications that make AJAX calls to web APIs.
- However, jQuery operates at the level of the browser document object model (DOM), and by default offers only an imperative, rather than declarative, model.
- For example, imagine that if a textbox's value exceeds 10, an element on the page should be made visible. In jQuery, this would typically be implemented by writing an event handler with code that would inspect the textbox's value and set the visibility of the target element.
- This is an imperative, code-based approach. Another framework might instead use databinding to bind the visibility of the element to the value of the textbox declaratively.
- As client-side behaviors grow more complex, data binding approaches frequently result in simpler solutions with less code and conditional complexity

jQuery vs a SPA Framework

Factor	jQuery	Angular
Abstracts the DOM	Yes	Yes
AJAX Support	Yes	Yes
Declarative Data Binding	No	Yes
MVC-style Routing	No	Yes
Templating	No	Yes
Deep-Link Routing	No	Yes

JQuery Vs a SPA Framework

- Most of the features jQuery lacks intrinsically can be added with the addition of other libraries. SPA framework like Angular provides these features in a more integrated fashion, since it's been designed with all of them in mind from the start.
- Also, jQuery is an imperative library, meaning that you need to call jQuery functions in order to do anything with jQuery. Much of the work and functionality that SPA frameworks provide can be done declaratively, requiring no actual code to be written.
- Data binding is a great example of this. In jQuery, it usually only takes one line of code to get the value of a DOM element or to set an element's value. However, you have to write this code anytime you need to change the value of the element, and sometimes this will occur in multiple functions on a page.
- Another common example is element visibility. In jQuery, there might be many different places where you'd write code to control whether certain elements were visible. In each of these cases, when using data binding, no code would need to be written. You'd simply bind the value or visibility of the elements in question to a viewmodel on the page, and changes to that viewmodel would automatically be reflected in the bound elements.

Angular SPAs

- Angular remains one of the world's most popular JavaScript frameworks. The redesigned Angular continues to be a robust framework for building Single Page Applications.
- Angular applications are built from components. Components combine HTML templates with special objects and control a portion of the page. A simple component from Angular's docs is shown here:

```
import{ Component } from '@angular/core';

@Component({

      selector: 'my-app',

      template: '<h1>Hello {{name}}</h1>'

})

export class AppComponent{ name = 'Angular'; }
```

Angular SPAs

- Components are defined using the @Component decorator function, which takes in metadata about the component. The selector property identifies the ID of the element on the page where this component will be displayed.
- The template property is a simple HTML template that includes a placeholder that corresponds to the component's name property, defined on the last line.
  - `Import { Component } from '@angular/core’;`
- By working with components and templates, instead of DOM elements, Angular apps can operate at a higher level of abstraction and with less overall code than apps written using just JavaScript (also called "vanilla JS") or with jQuery.
- Angular also imposes some order on how you organize your client-side script files.

Getting Started with Angular

- AngularJS is a client side JavaScript MVC framework to develop a dynamic web application. AngularJS was originally started as a project in Google but now, it is open source framework. AngularJS is entirely based on HTML and JavaScript, so there is no need to learn another syntax or language.
- AngularJS changes static HTML to dynamic HTML. It extends the ability of HTML by adding built-in attributes and components and also provides an ability to create custom attributes using simple JavaScript.
- We need the following tools to setup a development environment for AngularJS:
    1. AngularJS Library – download from angularjs.org
    2. Editor/IDE – notepad++, SublimeText, Visual Studio & others
    3. Web server – IIS, Apache, etc
    4. Browser

Advantages of AngularJS

- Open source JavaScript MVC framework.
- Supported by Google
- No need to learn another scripting language. It's just pure JavaScript and HTML.
- Supports separation of concerns by using MVC design pattern.
- Built-in attributes (directives) makes HTML dynamic.
- Easy to extend and customize.
- Supports Single Page Application.
- Uses Dependency Injection.
- Easy to Unit test.
- REST friendly.

See this example with jQuery

```
<!DOCTYPE html>
<html>
<head>
    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
</head>
<body>
    Enter Your Name: <input type="text" id="txtName" /> <br />
    Hello <label id="lblName"></label>

    <script>
        $(document).ready( function () {
            $('#txtName').keyup(function () {
                $('#lblName').text($('#txtName').val());
            });
        });
    </script>
</body>
</html>
```

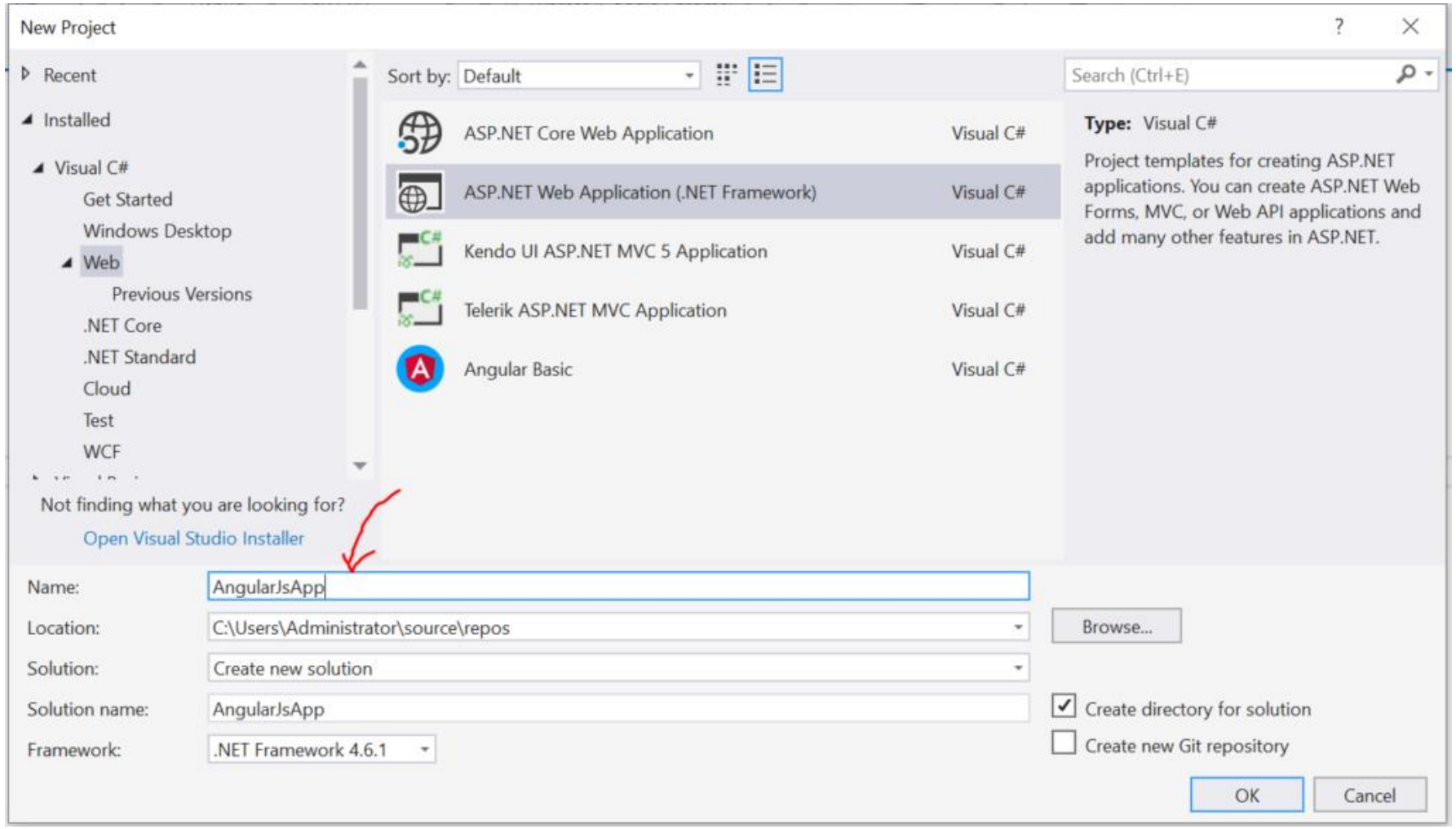
Example

- Conversion of above jQuery program to Angular Code to shows plain HTML code with couple of AngularJS directives (attributes) such as ng-app, ng-model, and ng-bind.

```
<!DOCTYPE html>
<html>
<head>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app>
    Enter Your Name: <input type="text" ng-model="name" /> <br />
    Hello <label ng-bind="name"></label>
</body>
</html>
```

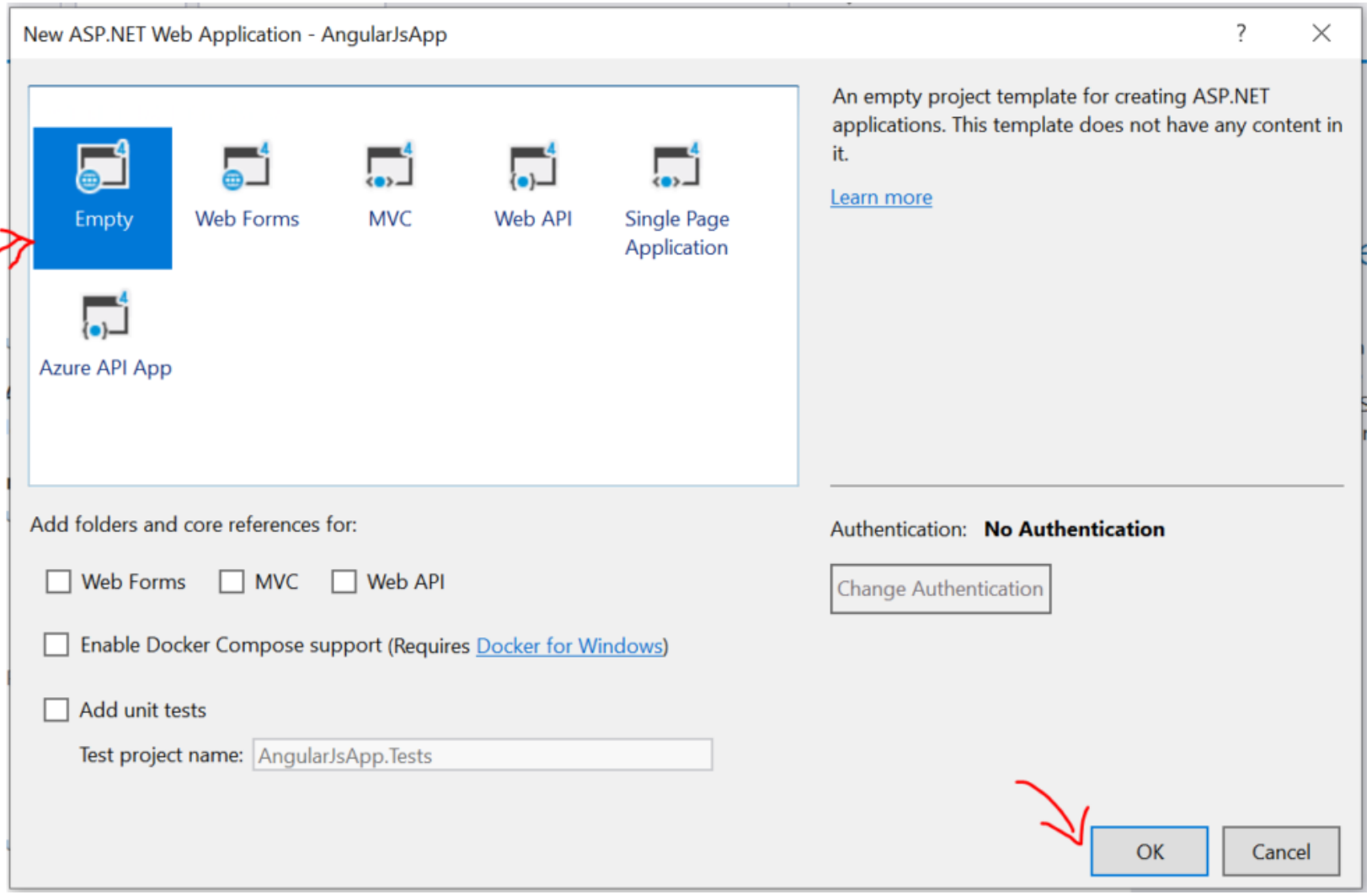
Setup angularjs application in Visual Studio 2019

- Open visual studio create angularjs project name like “angularJsApp”



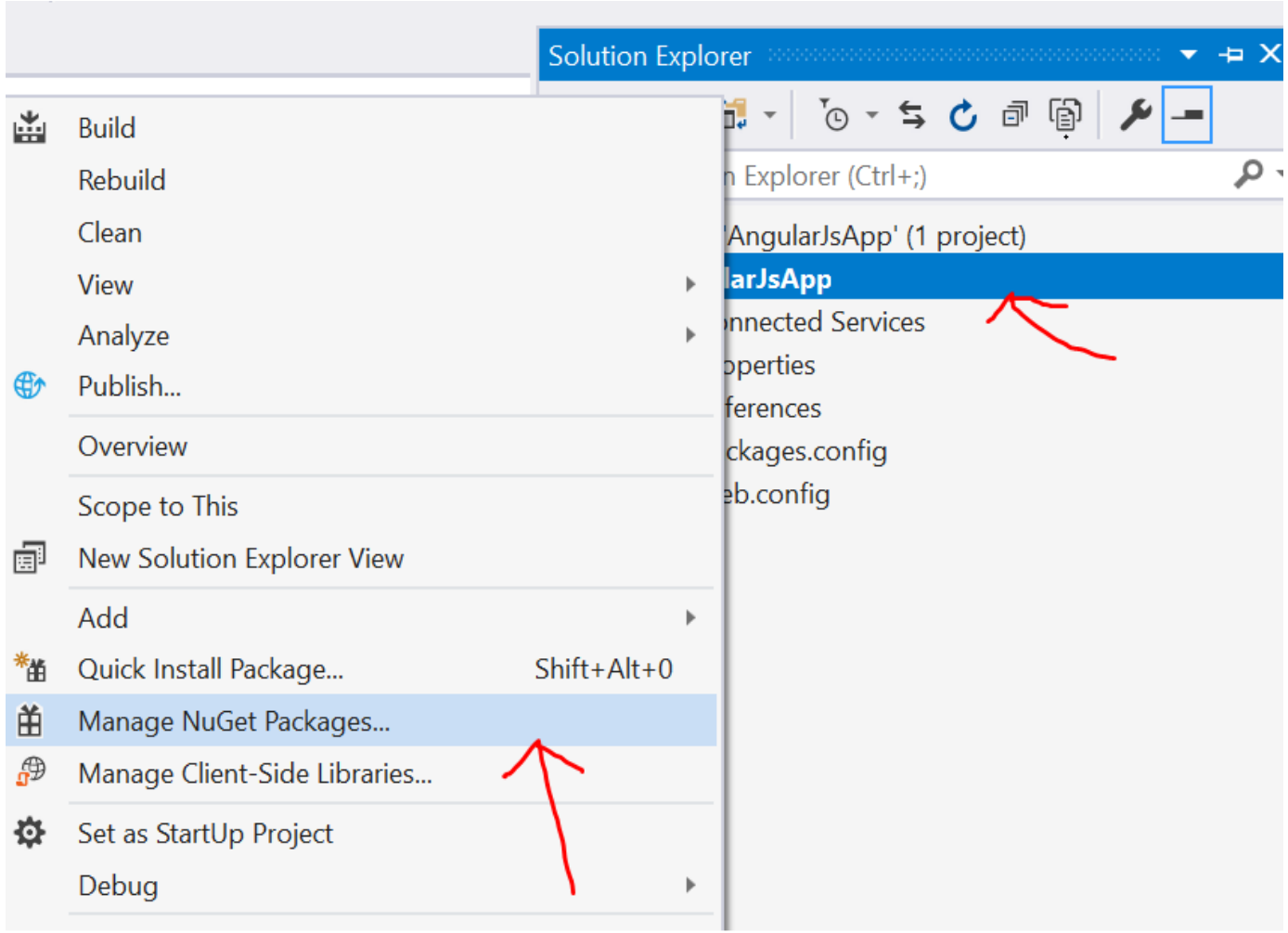
Setup angularjs application in Visual Studio 2019

- Select an empty project and then click on ok button



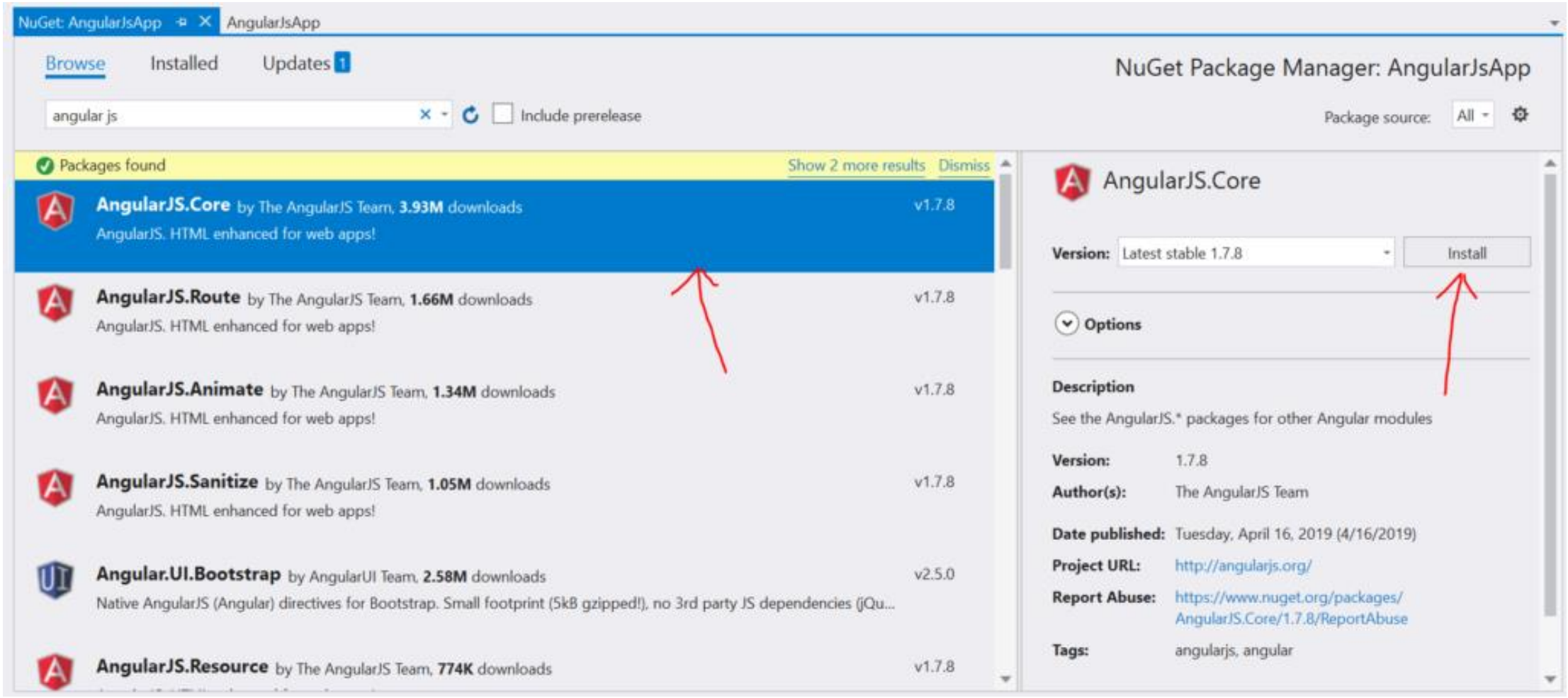
Setup angularjs application in Visual Studio 2019

- Right-click on your project select Manage NuGet packages



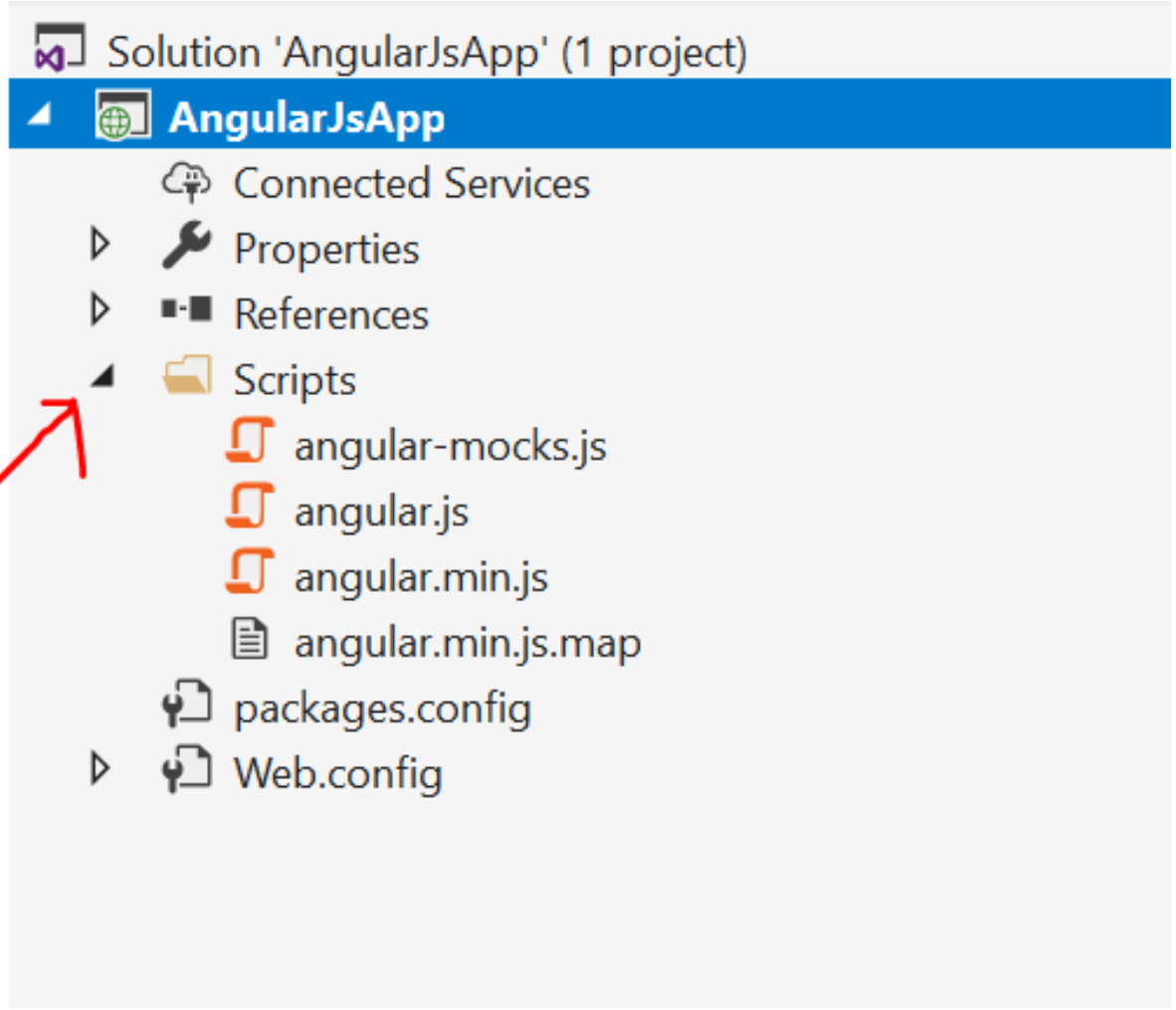
Setup angularjs application in Visual Studio 2019

- Browse the angularjs.core and install



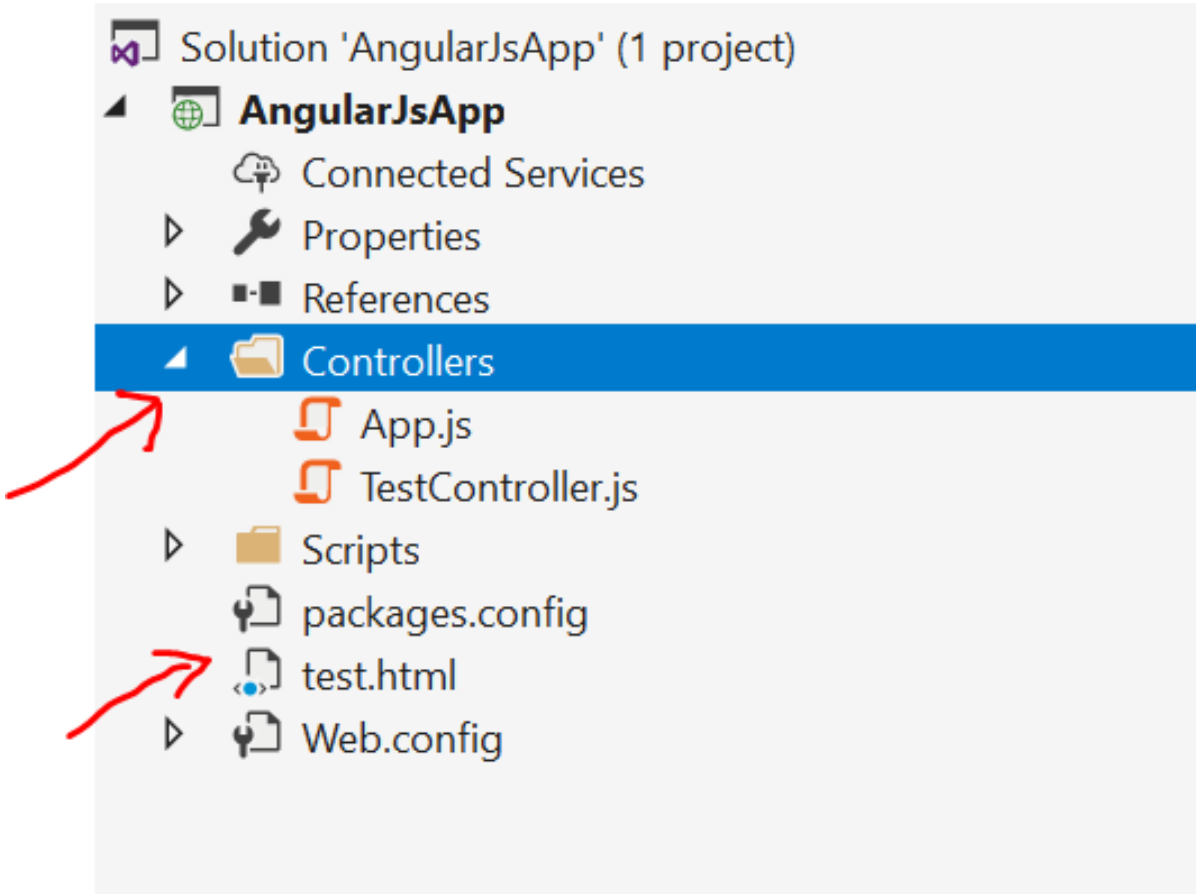
Setup angularjs application in Visual Studio 2019

- Once installed the angularjs you will have js file in a script folder



Setup angularjs application in Visual Studio 2019

- Setup is done. Now let's test by using one sample example. Create a directory structure for angularJs application following the below image.



App.js

```
var app = angular.module('myapp', []);
```

TestController.js

```
app.controller('TestController', function ($scope) {
    $scope.testmessage = "Setup angularjs application on visual studio 2019";
});
```

Test.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
    <script src="Scripts/angular.js"></script>
    <script src="Controllers/App.js"></script>
    <script src="Controllers/TestController.js"></script>
</head>
<body ng-app="myapp">
    <div ng-controller="TestController">
        <h2>{{testmessage}}</h2>
    </div>
</body>
</html>
```

REACT

- Unlike Angular, which offers a full Model-View-Controller pattern implementation, React is only concerned with views. It's not a framework, just a library. There are a number of libraries that are designed to be used with React to produce rich single page applications.
- One of React's most important features is its use of a virtual DOM. The virtual DOM provides React with several advantages, including performance (the virtual DOM can optimize which parts of the actual DOM need to be updated) and testability (no need to have a browser to test React and its interactions with its virtual DOM).
- Rather than having a strict separation between code and markup (with references to JavaScript appearing in HTML attributes perhaps), React adds HTML directly within its JavaScript code as JSX. JSX is HTML-like syntax that can compile down to pure JavaScript.

REACT

- For Example

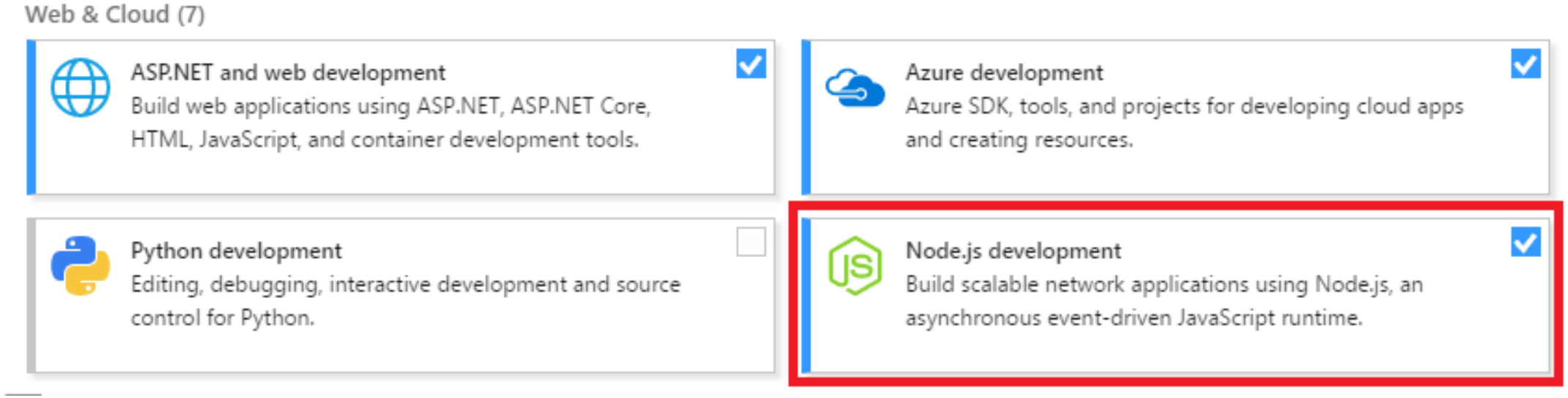
```
<ul>
{ authors.map(author =>
   <li key={author.id}>{author.name}</li>
)}
</ul>
```

- Because React isn't a full framework, you'll typically want other libraries to handle things like routing, web API calls, and dependency management. The nice thing is, you can pick the best library for each of these.

Create a Node.js and React app in Visual Studio

Prerequisites

- Visual Studio installed and the Node.js development workload.
- If you need to install the workload but already have Visual Studio, go to Tools > Get Tools and Features..., which opens the Visual Studio Installer. Choose the Node.js development workload, then choose Modify.



Create a Node.js and React app in Visual Studio

Create a Node.js Web Application Project

1. Open Visual Studio.
2. Create a new project.

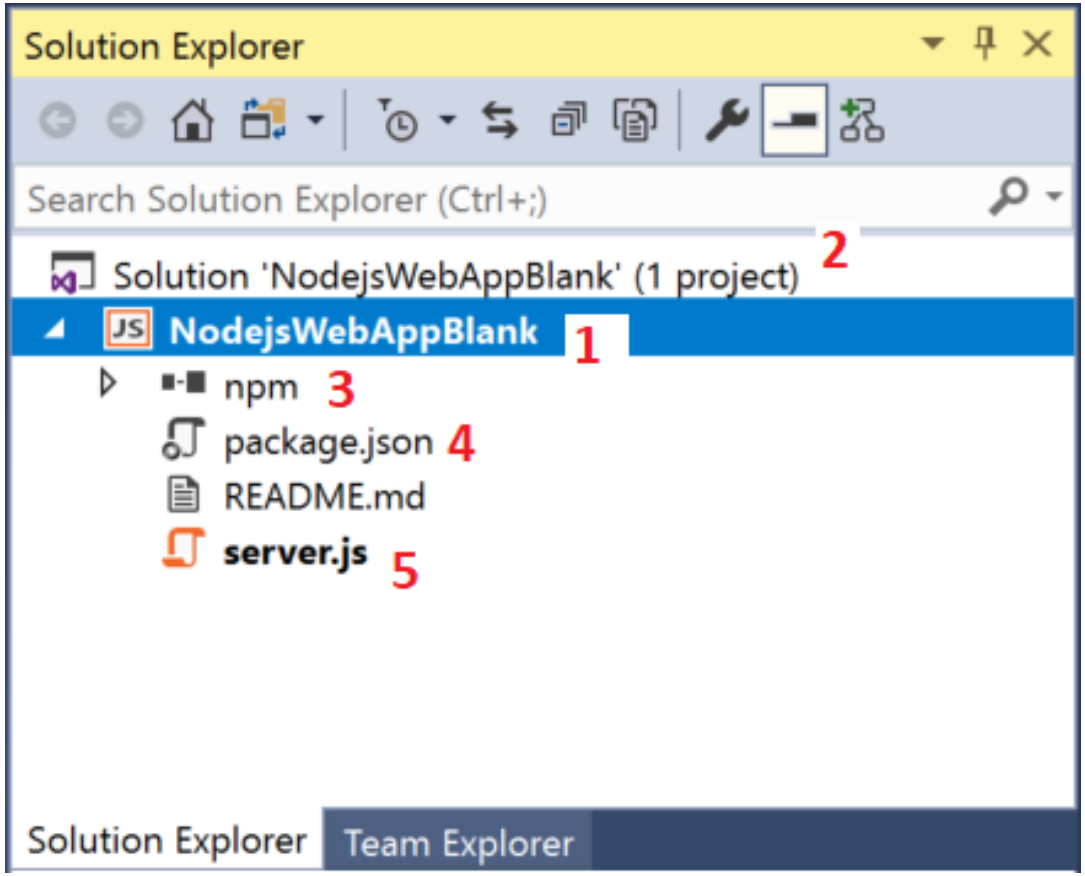
- Press Esc to close the start window. Type Ctrl + Q to open the search box, type Node.js, then choose Blank Node.js Web Application - JavaScript
- In the dialog box that appears, choose Create.
- If you don't see the Blank Node.js Web Application project template, you must add the Node.js development workload.
- Visual Studio creates the new solution and opens your project.

Create a Node.js and React app in Visual Studio

Add npm packages

This app requires a number of npm modules to run correctly.

- react
- react-dom
- express
- path
- ts-loader
- typescript
- webpack
- webpack-cli

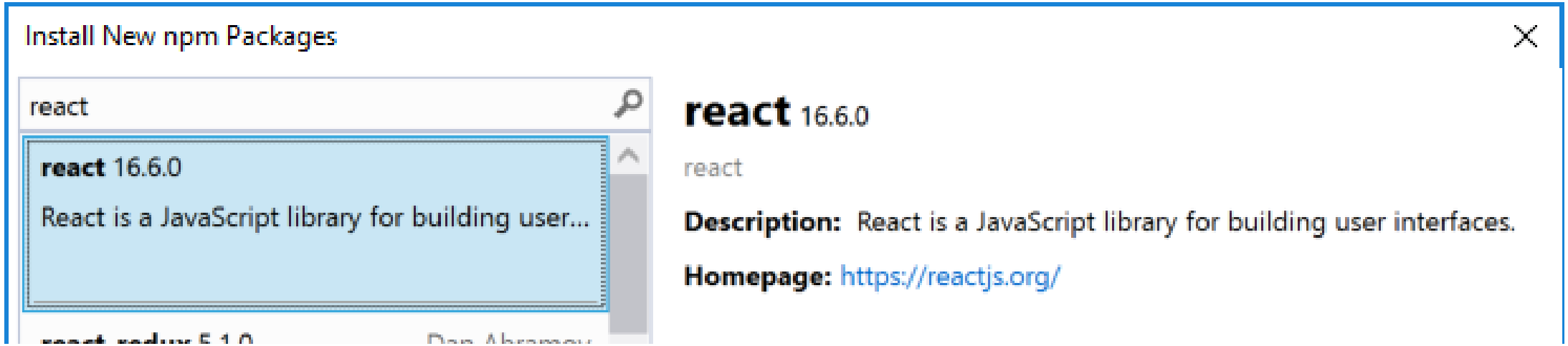


Create a Node.js and React app in Visual Studio

Add npm packages

1. In Solution Explorer (right pane), right-click the npm node in the project and choose Install New npm Packages. In the Install New npm Packages dialog box, you can choose to install the most current package version or specify a version.

2. In the Install New npm Packages dialog box, search for the react package, and select Install Package to install it.



Create a Node.js and React app in Visual Studio

When installed, the package appears under the npm node.

- The project's package.json file is updated with the

new package information including the package version.

- Instead of using the UI to search for and add the

rest of the packages one at a time, paste the

following code into package.json.

To do this, add a dependencies section with this code:

JSON

```
"dependencies": {
  "express": "~4.17.1",
  "path": "~0.12.7",
  "react": "~16.13.1",
  "react-dom": "~16.13.1",
  "ts-loader": "~7.0.1",
  "typescript": "~3.8.3",
  "webpack": "~4.42.1",
  "webpack-cli": "~3.3.11"
}
```

CHOOSING A SPA FRAMEWORK

- When considering which JavaScript framework will work best to support your SPA, keep in mind the following considerations:
  - Is your team familiar with the framework and its dependencies (including TypeScript in some cases)?
  - How opinionated is the framework, and do you agree with its default way of doing things?
  - Does it (or a companion library) include all of the features your app requires?
  - Is it well documented?
  - How active is its community? Are new projects being built with it?
  - How active is its core team? Are issues being resolved and new versions shipped regularly?
- JavaScript frameworks continue to evolve with breakneck speed. Use the considerations listed above to help mitigate the risk of choosing a framework you'll later regret being dependent upon

Discussion Exercise

1. Write about the State Management Strategies.
2. What is Session State? Show with an example to manage session state in ASP.NET Core.
3. Show the difference between TempData and Using HttpContext with suitable example.
4. How do you manage to handle state with client side strategies?

Unit 8

BASIC CONCEPTS ON ASP.NET CORE SECURITY

BASIC CONCEPTS ON ASP.NET CORE SECURITY

- This Unit shows how to add users to an ASP.NET Core application by adding authentication. With authentication, users can register and log in to your app using an email and password. Whenever you add authentication to an app, you inevitably find you want to be able to restrict what some users can do. The process of determining whether a user can perform a given action on your app is called authorization.
- The two concepts are often used together, but they're definitely distinct:

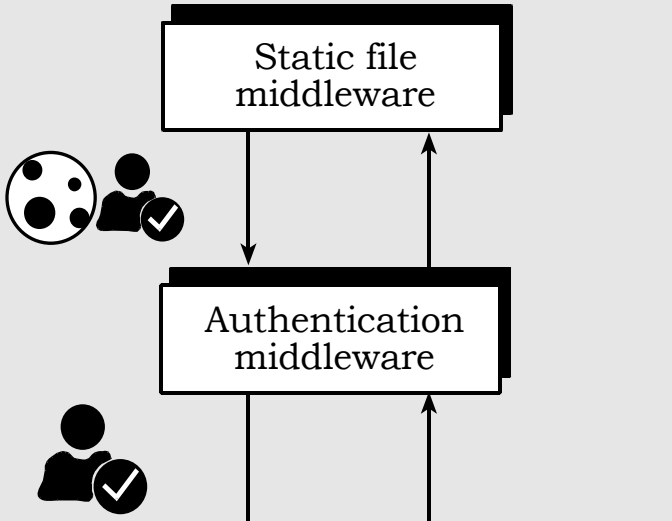
a. Authentication—The process of determining who made a request

b. Authorization—The process of determining whether the requested action is allowed

Authorization in ASP.NET Core

- The ASP.NET Core framework has authorization built in, so you can use it anywhere in your app, but it's most common to apply authorization as part of MVC. For both traditional web apps and web APIs, users execute actions on your controllers. Authorization occurs before these actions execute, as shown in figure 1. This lets you use different authorization requirements for different action methods. As you can see in figure, authorization occurs as part of MvcMiddleware, after AuthenticationMiddewarehas authenticated the request.
- Authorization, is checking whether a particular user has permission to execute an action. In ASP.NET Core, you'd achieve this by checking whether a user has a particular claim.
- A request is made to the URL /recipe/index. MvcMiddleware.The authentication middleware deserializes the ClaimsPrincipal from the encrypted cookie. The authorize filter runs after routing but before model binding or validation. If authorization is successful, the action method executes and generates a response as normal. If authorization fails, the authorize filter returns an error to the user, and the action is not executed.

A request is made
to the URL/receipe/index



The authentication middlware deseralizes the Claims Principal from the encrypted cookie.

The authorize filter runs after routing but before model binding or validation.

If authorization is successful, the action method exeuctes and generates a response as normal.

If authorization is fails, the authorize filter returns an error to the user, and the action is not executed.

Authorization in ASP.NET Core

- There’s an even more basic level of authorization that you haven’t considered yet— only allowing authenticated users to execute an action. There are only two possibilities:
  - The user is authenticated- The action executes as normal.
  - The user is unauthenticated - The user can’t execute the action.
- You can achieve this basic level of authorization by using the [Authorize] attribute. You can apply this attribute to your actions, to restrict them to authenticated (logged-in) users only. If an unauthenticated user tries to execute an action protected with the [Authorize] attribute in this way, they’ll be redirected to the login page.

Authorization in ASP.NET Core

```
public class HomeController : Controller

{

 public IActionResult Index ()

 {

        return View () ;

      [Authorize]

public IActionResult AuthedUseronly ()

{

      return View () ;

}
```

This action can be executed by anyone, even

Applies [Authorize] to individual actions or whole controllers

This action can only be executed by authenticated users.

ASP.NET Core Identity

- ASP.NET Core Identity adds user interface (UI) login functionality to ASP.NET Core web apps and manages users, passwords, profile data, roles, claims, tokens, email confirmation, and more.
- Users can create an account with the login information stored in Identity

Create a Web app with authentication

- Create an ASP.NET Core Web Application project with Individual User Accounts.
  - In Visual Studio, Select File > New > Project.
  - Select ASP.NET Core Web Application. Name the project WebApp1 to have the same namespace as the project download. Click OK.
  - Select an ASP.NET Core Web Application, then select Change Authentication.
  - Select Individual User Accounts and click OK.

ASP.NET Core Identity

- The generated project provides ASP.NET Core Identity as a Razor Class Library. The Identity Razor Class Library exposes endpoints with the Identity area.
- For example:
  - /Identity/Account/Login
  - /Identity/Account/Logout
  - /Identity/Account/Manage

Apply migrations

- Apply the migrations to initialize the database and Run the following command in the Package Manager Console (PMC):
- PM> Update-Database

ASP.NET Core Identity

Test Register and Login

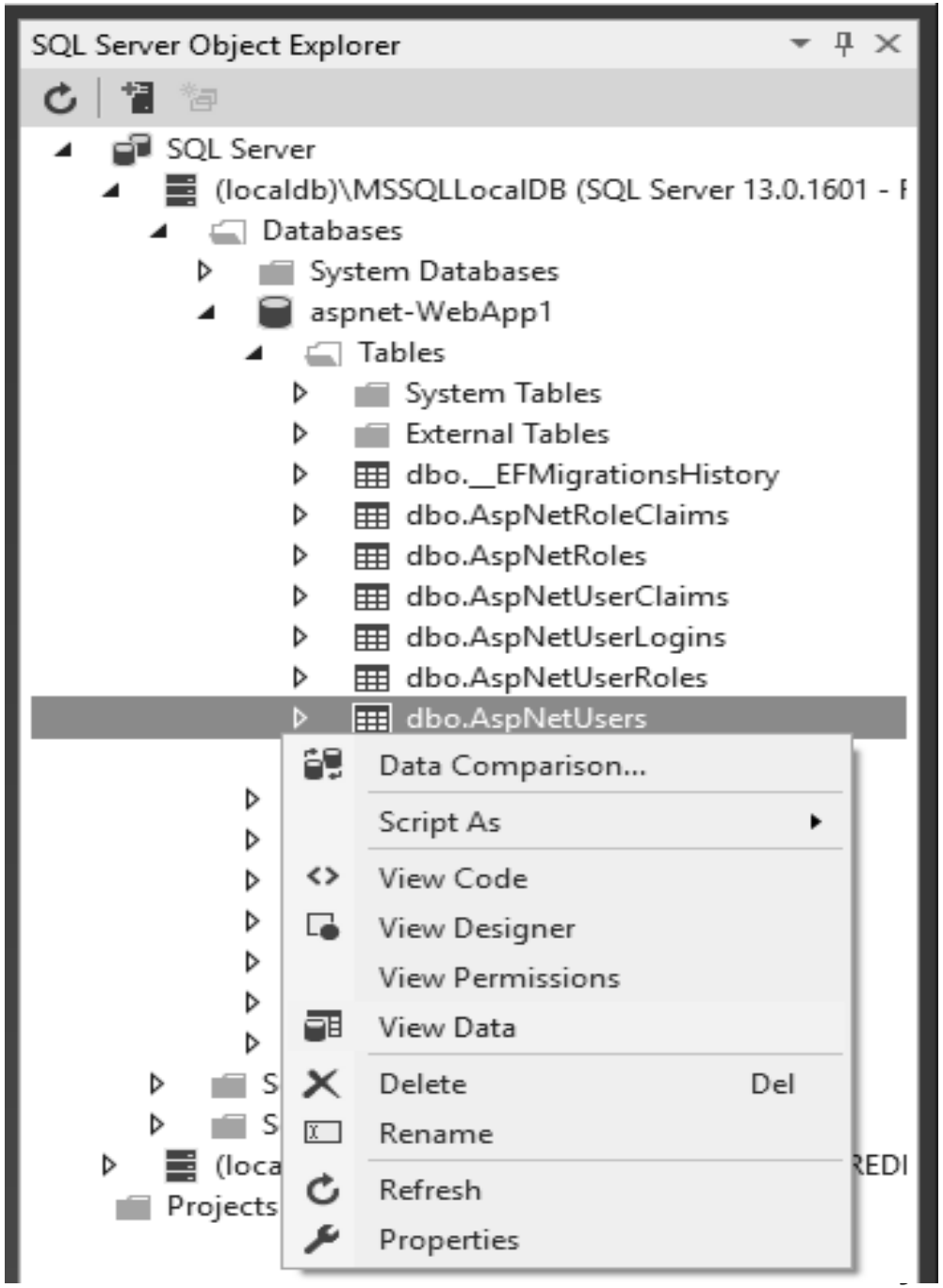
- Run the app and register a user.

Depending on your screen size, you might need to select the navigation toggle button to see the Register andLogin links.

View the Identity database

- From the View menu, select
- SQL Server Object Explorer

Navigate to (localdb)MSSQLLocalDB (SQL Server 13). Right-click on dbo.AspNetUsers > View Data:



ADDING AUTHENTICATION TO APPS AND IDENTITY SERVICE CONFIGURATIONS

- Services are added in ConfigureServices.
- The typical pattern is to call all the Add{Service} methods, and then call all the services.Configure{Service} methods.

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddDbContext<ApplicationDbContext>(options =>
      // options.UseSqlite(
  options.UseSqlServer(
  Configuration.GetConnectionString("DefaultConnection")));
  services.AddDefaultIdentity<IdentityUser>(
    options=>options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

```
services.AddRazorPages();
services.Configure<IdentityOptions>(options =>
    {
        // Password settings.
   options.Password.RequireDigit = true;
   options.Password.RequireLowercase = true;
   options.Password.RequireNonAlphanumeric = true;
   options.Password.RequireUppercase = true;
   options.Password.RequiredLength = 6;
   options.Password.RequiredUniqueChars = 1;

            // Lockout settings.
   options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
   options.Lockout.MaxFailedAccessAttempts = 5;
   options.Lockout.AllowedForNewUsers = true;
```

```
 // User settings.
 options.User.AllowedUserNameCharacters =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+";
 options.User.RequireUniqueEmail = false;
 });
 services.ConfigureApplicationCookie(options =>
     {
         // Cookie settings
 options.Cookie.HttpOnly = true;
 options.ExpireTimeSpan = TimeSpan.FromMinutes(5);
 options.LoginPath = "/Identity/Account/Login";
 options.AccessDeniedPath = "/Identity/Account/AccessDenied";
 options.SlidingExpiration = true;
    });
}
```

- The preceding highlighted code configures Identity with default option values. Services are made available to the app through dependency injection.
- The template-generated app doesn't use authorization.
- app.UseAuthorization is included to ensure it's added in the correct order should the app add authorization.
- UseRouting, UseAuthentication, UseAuthorization, and UseEndpoints must be called in the order shown in the preceding code.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
 if (env.IsDevelopment()) {
     app.UseDeveloperExceptionPage();
     app.UseDatabaseErrorPage();
 }
 else {
   app.UseExceptionHandler("/Error");
   app.UseHsts();
 }
 app.UseHttpsRedirection();
 app.UseStaticFiles();
 app.UseRouting();
 app.UseAuthentication();
 app.UseAuthorization();
app.UseEndpoints(endpoints =>
       {
   endpoints.MapRazorPages();
       });
 }
```

AUTHORIZATION: ROLES, CLAIMS AND POLICIES, SECURING CONTROLLERS AND ACTION METHODS

- When an identity is created it may belong to one or more roles. For example, Admin1 may belong to the Administrator and User roles whilst User1 may only belong to the User role. How these roles are created and managed depends on the backing store of the authorization process. Roles are exposed to the developer through the IsInRole method on the ClaimsPrincipal class.

Roles

- Role-based authorization checks are declarative—the developer embeds them within their code, against a controller or an action within a controller, specifying roles which the current user must be a member of to access the requested resource.

Roles

- For example, the following code limits access to any actions on the AdministrationController to users who are a member of the Administrator role:

```
[Authorize(Roles = "Administrator")]
public class AdministrationController : Controller{}
```

- You can specify multiple roles as a comma separated list:

```
[Authorize(Roles = "HRManager,Finance")]
public class Salary Controller : Controller {}
```

- This controller would be only accessible by users who are members of the HRManager role or the Finance role.

Roles

- If you apply multiple attributes then an accessing user must be a member of all the roles specified; the following sample requires that a user must be a member of both the PowerUser and ControlPanelUser role.

```
[Authorize(Roles = "PowerUser")]
[Authorize(Roles = "ControlPanelUser")]
public class ControlPanelController : Controller
{
}
```

Roles

- You can further limit access by applying additional role authorization attributes at the action level:

```
[Authorize(Roles = "Administrator, PowerUser")]
public class ControlPanelController : Controller {
    public ActionResult SetTime()
    {
    }
    [Authorize(Roles = "Administrator")]
    Public ActionResult ShutDown()
    {
    }
}
```

Policy based Role Checks

- Role requirements can also be expressed using the new Policy syntax, where a developer registers a policy at startup as part of the Authorization service configuration. This normally occurs in ConfigureServices() in your Startup.cs file.

```
public void ConfigureServices(IServiceCollection services)
{
services.AddControllersWithViews();
services.AddRazorPages();
    services.AddAuthorization(options =>
    {
        options.AddPolicy("RequireAdministratorRole",
                          policy =>policy.RequireRole("Administrator"));
        });
}
```

Policy based Role Checks

- Policies are applied using the Policy property on the AuthorizeAttribute attribute:

```
[Authorize(Policy = "RequireAdministratorRole")]
public IActionResult Shutdown() {
        return View();
}
```

- If you want to specify multiple allowed roles in a requirement then you can specify them as parameters to the RequireRole method:

```
options.AddPolicy("ElevatedRights", policy =>
    policy.RequireRole("Administrator", "PowerUser", "BackupAdministrator"));
```

- This example authorizes users who belong to the Administrator, PowerUser or BackupAdministrator roles.

Add Role services to Identity

- Append AddRoles to add Role services:

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddDbContext<ApplicationDbContext>(options =>
   options.UseSqlServer(
       Configuration.GetConnectionString("DefaultConnection")));
  services.AddDefaultIdentity<IdentityUser>().AddRoles<IdentityRole>()
       .AddEntityFrameworkStores<ApplicationDbContext>();
  services.AddControllersWithViews();
  services.AddRazorPages();
}
```

Claims and Policies

- When an identity is created it may be assigned one or more claims issued by a trusted party. A claim is a name value pair that represents what the subject is, not what the subject can do. Claims based authorization, at its simplest, checks the value of a claim and allows access to a resource based upon that value.
- For example, if you want access to a night club the authorization process might be: The door security officer would evaluate the value of your date of birth claim and whether they trust the issuer (the driving license authority) before granting you access.
- An identity can contain multiple claims with multiple values and can contain multiple claims of the same type.

Adding claims checks

- First you need to build and register the policy. This takes place as part of the Authorization service configuration, which normally takes part in ConfigureServices() in your Startup.cs file.

```
public void ConfigureServices(IServiceCollection services) {
services.AddControllersWithViews();
services.AddRazorPages();
services.AddAuthorization(options =>
    {
    options.AddPolicy("EmployeeOnly", policy =>
policy.RequireClaim("EmployeeNumber"));
    });
}
```

Adding claims checks

- In this case the EmployeeOnly policy checks for the presence of an EmployeeNumber claim on the current identity. You then apply the policy using the Policy property on the AuthorizeAttribute attribute to specify the policy name;

```
[Authorize(Policy = "EmployeeOnly")]
public IActionResult VacationBalance() {  return View();  }
```

- The AuthorizeAttribute attribute can be applied to an entire controller, in this instance only identities matching the policy will be allowed access to any Action on the controller.

```
[Authorize(Policy = "EmployeeOnly")]
public class VacationController:Controller {
        public ActionResult VacationBalance() { }
}
```

Policies

- If you apply multiple policies to a controller or action, then all policies must pass before access is granted. For example:

```
[Authorize(Policy = "EmployeeOnly")]
public class SalaryController : Controller{
        public ActionResult Payslip()
        {
        }
        [Authorize(Policy = "HumanResources")]
        public      ActionResult    UpdateSalary()
        {
        }
}
```

Policy-based authorization in ASP.NET Core

- Underneath the covers, role-based authorization and claims-basedauthorization use a requirement, a requirement handler, and a pre-configured policy.
- An authorization policy consists of one or more requirements. It's registered as part of the authorization service configuration, in the Startup.ConfigureServices method:

```
public void ConfigureServices(IServiceCollection services) {
    services.AddControllersWithViews();
    services.AddRazorPages();
    services.AddAuthorization(options =>
    {
        options.AddPolicy("AtLeast21", policy =>
        policy.Requirements.Add(new MinimumAgeRequirement(21)));
    });
}
```

Apply policies to MVC controllers

- If you're using Razor Pages, see Apply policies to Razor Pages in this document.
- Policies are applied to controllers by using the [Authorize] attribute with the policy name. For example:

```
using Microsoft.AspNetCore.Authorization;
Using Microsoft.AspNetCore.Mvc;
[Authorize(Policy = "AtLeast21")]
public class AlcoholPurchaseController : Controller {
  public IActionResult Index() => View();
}
```

Apply policies to Razor Pages

- Policies are applied to Razor Pages by using the [Authorize] attribute with the policy name. For example:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc.RazorPages;
[Authorize(Policy = "AtLeast21")]
public class AlcoholPurchaseModel : PageModel
{
}
```

- Policies cannot be applied at the Razor Page handler level, they must be applied to the Page.Policies can be applied to Razor Pages by using an authorization convention.

Securing Action Method in Controller

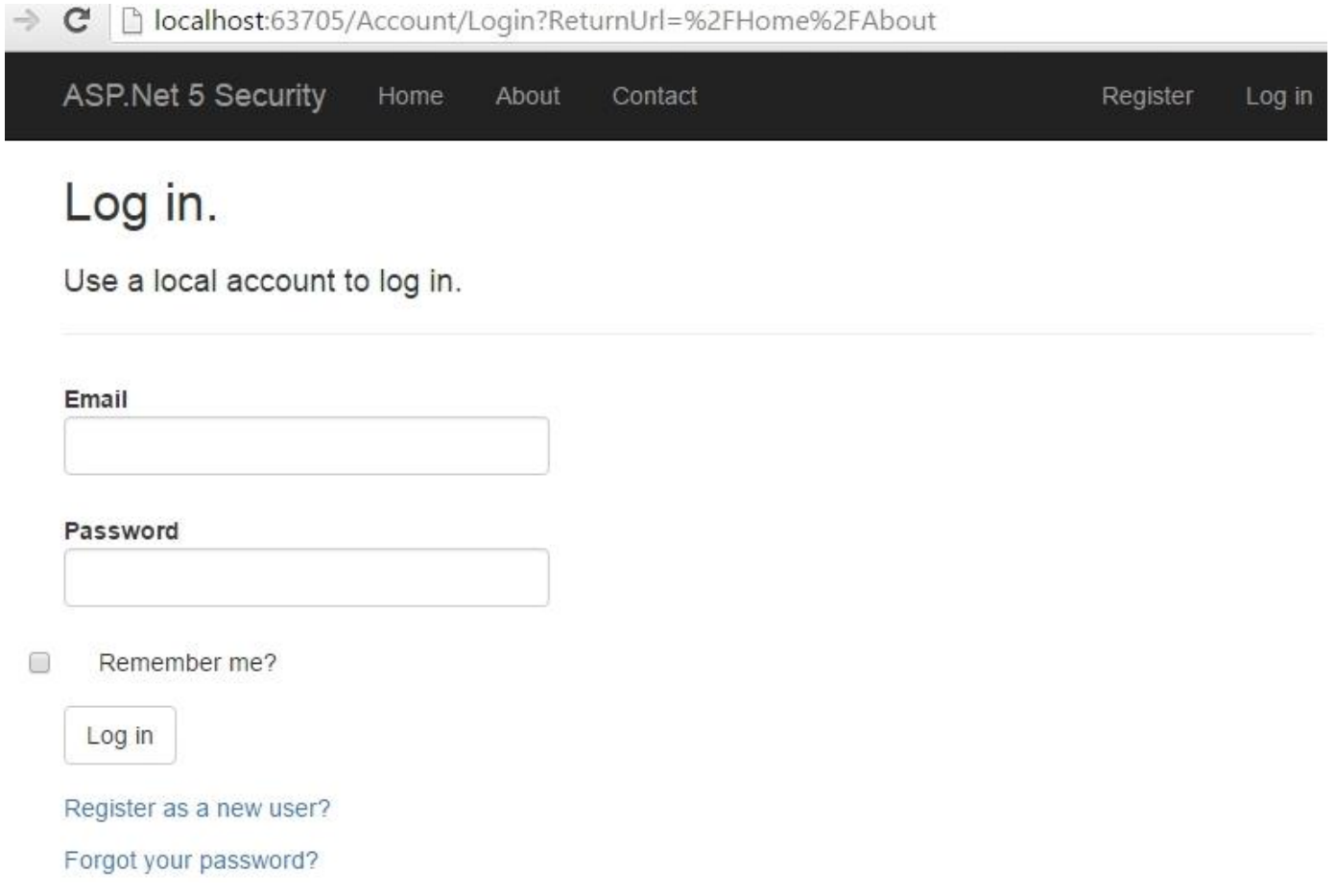
- Let’s assume that the About page is a secure page and only authenticated users should be able to access it. We just have to decorate the About action method in the Home controller with an[Authorize] attribute:

```
[Authorize]
public IActionResult About() {
   ViewData["Message"] = "This is my about page";
   return View();
}
```

- Making the preceding change will redirect user to the log-in page when user tries to access the log-in page without logging in to the application:

Securing Action Method in Controller

- Making the preceding change will redirect the user to the log-in page when the user tries to access the log-in page without logging in to the application:



Unit 9

HOSTING AND DEPLOYING ASP.NET CORE APPLICATION

HOSTING AND DEPLOYING ASP.NET CORE APPLICATION

- Once you successfully developed your web application, you may require to host the application to the server so that other people can access it. The process of deploying/installing an application into the server is called "Hosting".

Web Server

- A web server is a process for hosting web applications, which responds to HTTP requests and delivers contents and services. A web server allows an application to process messages that arrive through specific TCP ports (by default). Default port for HTTP traffic is 80, and the one for HTTPS is 443.
- When you visit a website in your browser, you don’t typically specify the port number unless the web server is configured to receive traffic on ports other than the default.
- Some of the web servers that we can use to host ASP.NET Core are: Microsoft IIS, Apache, NGINX.

IIS web server

- The IIS web server comes from the Microsoft stable and runs only on the Microsoft Windows operating system. It is actually not free, since it comes as a part of the Windows operating system. You might feel comfortable with IIS if you have already used the Windows OS ecosystem. It also comes with the support of the .NET framework which was released by Microsoft and support services for IIS are provided directly by Microsoft.

Advantages of IIS

- Has the support of Microsoft.
- You can have access to the .NET framework along with ASPX scripts.
- Can be easily integrated with other Microsoft services like ASP, MS SQL etc.

Apache web server

- Apache is an open source web server which was developed and maintained Apache Software Foundation. It is a result of the collaborative efforts which was aimed at creating a robust and secure commercial grade web server which adhered to all the HTTP standards.
- It has been the market leader since it entered the web server market in 1995 and remains the web server of choice for its ability to function across multiple platforms.
- Apache is equally efficient on almost every operating system but finds can be found to be in maximum use when combined with Linux.

Advantages of Apache

- As it open source, so there are no licensing fees.
- It is flexible, meaning that you can choose the modules you want.
- Has a high level of security.
- Strong user community to provide backend support.
- Runs equally well on UNIX, Linux, MacOS, Windows.

NGINX

- NGINX is a robust web server which was developed by Russian developer Igor Sysoev. It is a free open-source HTTP server which can be used as a mail proxy, reverse proxy server when required. Most importantly, it can take care of a huge number of concurrent users with minimal resources in an efficient manner. NGINX, is particularly of great help when the situation of handling massive web traffic arises.
- NGINX has a lightweight architecture and is highly efficient. This is probably the only web server which can handle huge traffic with very limited hardware resources. NGINX acts as a sort of shock absorber which protects Apache servers when faced with security vulnerabilities and sudden traffic spikes.

Advantages of NGINX

- Open source.
- A high speed web server which can be used as a reverse-proxy server.
- Can be used better in a virtual private server environment.

HOSTING MODELS IN ASP.NET CORE

- There are 2 types of hosting models in ASP.NET Core:
  - Out-of-process Hosting
  - In-process Hosting
- Before ASP.Net Core 2.2 we have only one hosting model, which is Out-of-process but after due to the performance we have In Process Hosting Model in 2.2+ versions.

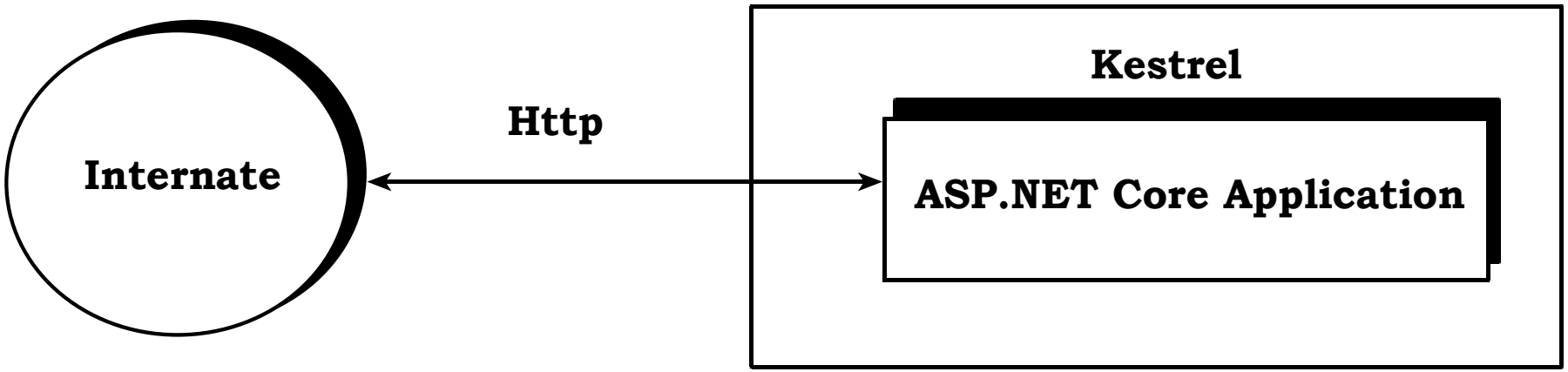
Out-of-process Hosting Model

- In Out-of-process hosting models, we can either use the Kestrel server directly as a user request facing server or we can deploy the app into IIS which will act as a proxy server and sends requests to the internal Kestrel server. In this type of hosting model, we have two options:
  - Using Kestrel
  - Using Proxy Server

Out-of-process Hosting Model

Using Kestrel

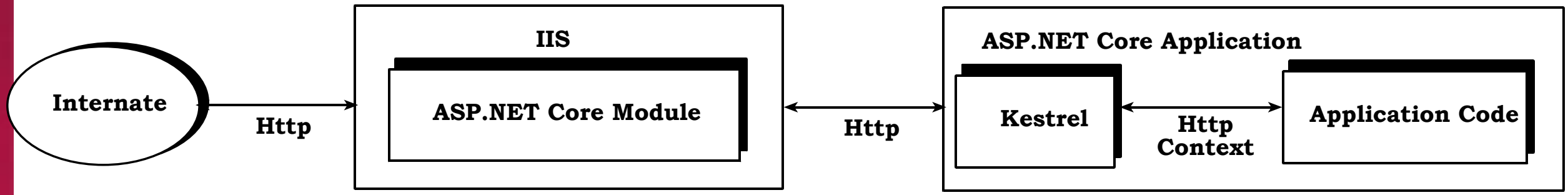
- Kestrel is a cross-platform web server for ASP.NET Core. Kestrel is the webserver that's included by default in ASP.NET Core project templates.
- Kestrel itself acts as edge server which directly server user requests. It means that we can only use the Kestrel server for our application.



Out-of-process Hosting Model

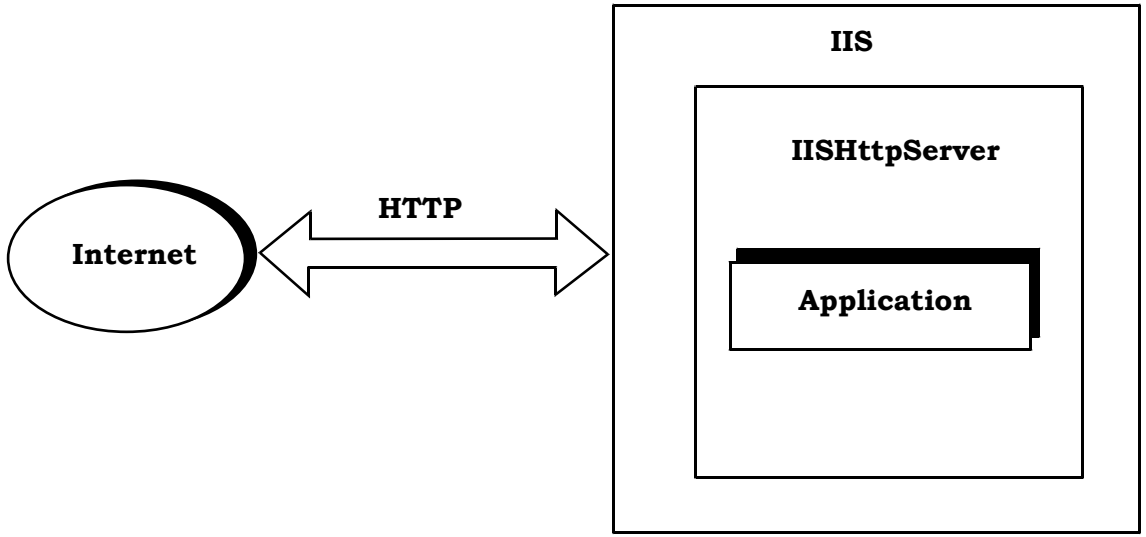
Using a Proxy Server

- Due to limitations of the Kestrel server, we cannot use this in all the apps. In such cases, we have to use powerful servers like IIS, NGINX or Apache. So, in that case, this server acts as a reserve proxy server which redirects every request to the internal Kestrel sever where our app is running. Here, two servers are running. One is IIS and another is Kestrel.
- This model is a default model for all the applications implemented before .NET Core 2.2. But there are some of the limitations of using this type such as performance slowness.



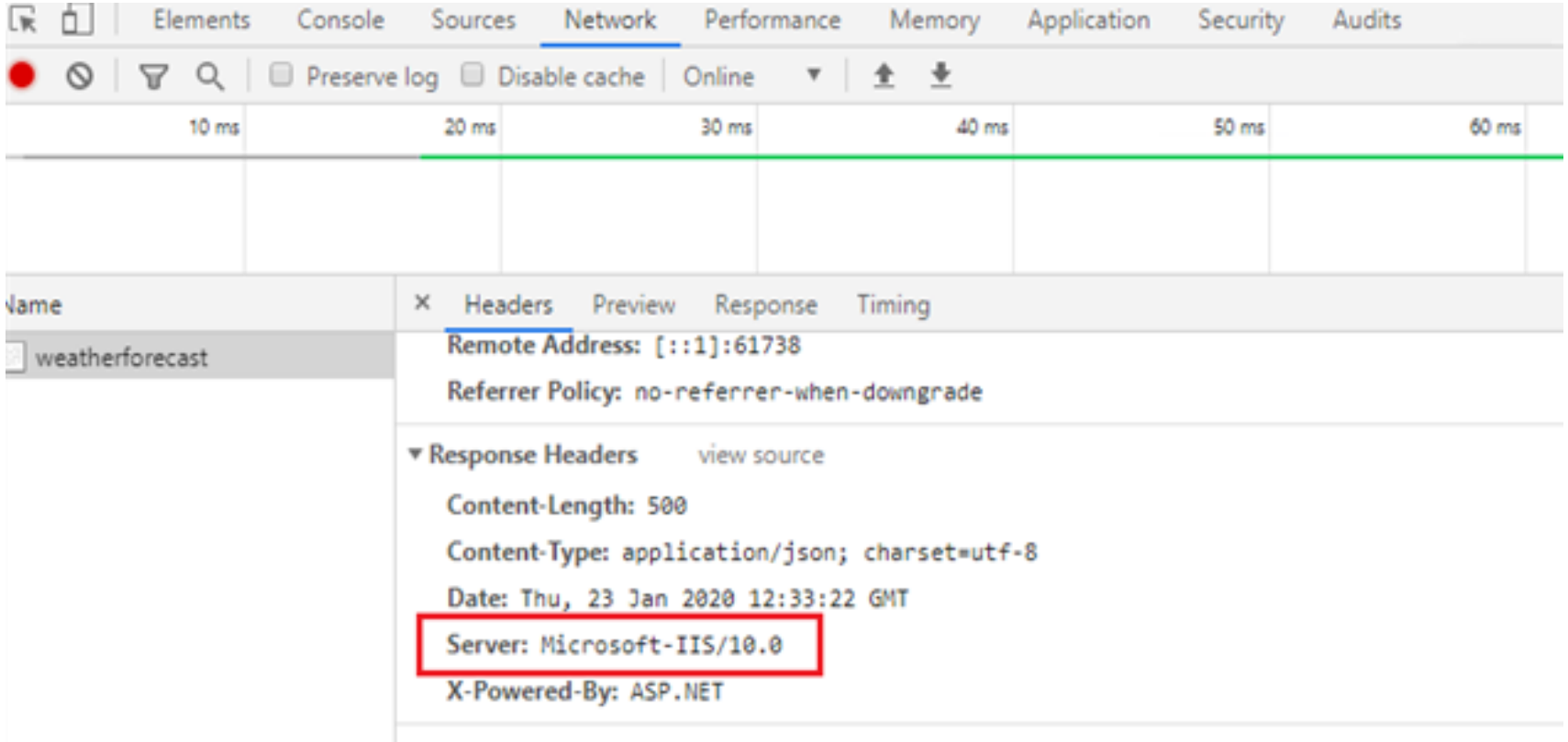
In-process Hosting Model

- After the release of .NET Core 2.2, it introduced a new type of hosting which is called In-process hosting. In this type, only one server is used for hosting like IIS, Nginx or Linux. It means that the App is directly hosted inside of IIS. No Kestrel server is being used. IIS HTTP Server (IISHttpServer) is used instead of the Kestrel server to host apps in IIS directly. ASP.NET Core 3.1 onwards In-process hosting model is used as a default model whenever you create a new application using an existing template.



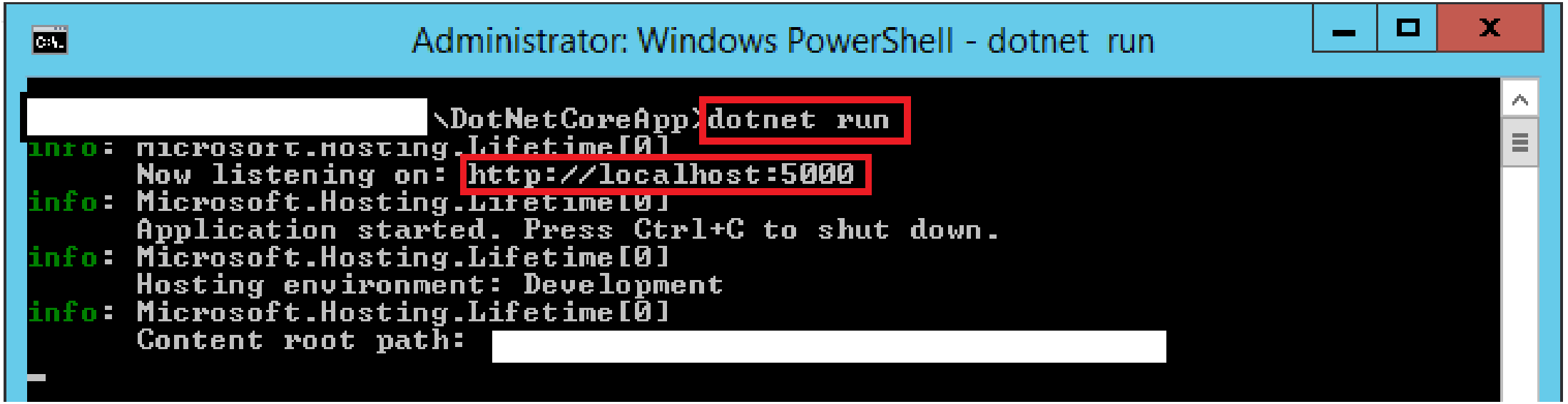
Let’s see different types of hosting models

- Now let's see how to check which hosting model is being used.
- Run the application on the IISExpress server, then open the browsers network tab and check for the first call. Under the server section, you will able to see it showing Microsoft IIS.



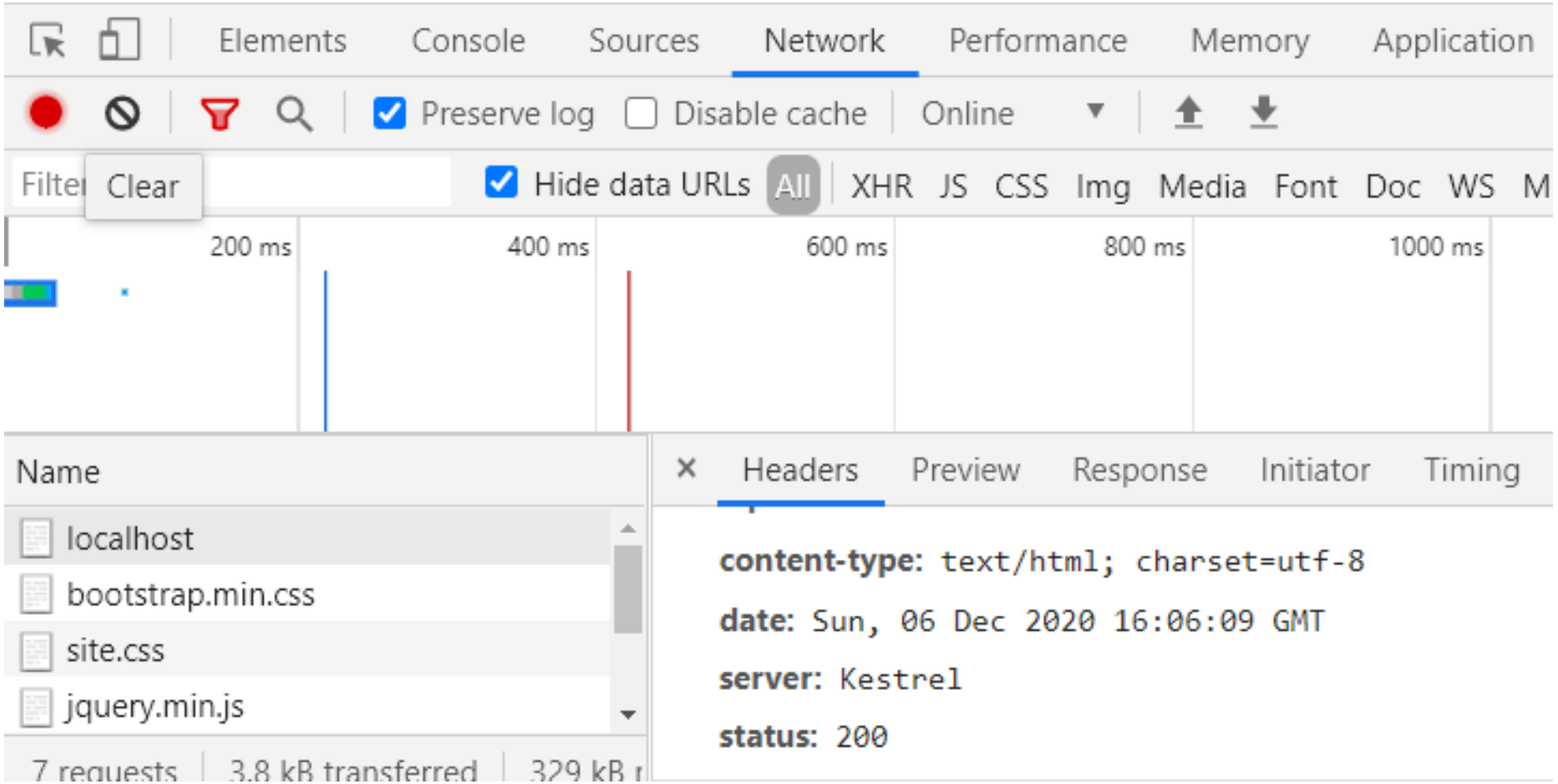
Let’s see different types of hosting models

- Stop the app and open the command prompt and run the same application using dotnet CLI using the command dotnet run. Now it will host app on http://localhost:5000.



Let’s see different types of hosting models

- Browse the URL and open the network tab to see the server attribute as Kestrel.



DEPLOY .NET CORE APPLICATION ON LINUX

- When Microsoft launched their .Net Core framework the key selling point was it is a cross-platform framework, which means that now we can host our .Net application not only on Windows but on Linux too.
- Let's see how we can deploy .Net core application on Linux.

Step 1 - Publish your .Net Core application

- First, create a .Net core application on VS; you can make an MVC project or Web API project and if you already have an existing project, then open it.

1. Right Click on your project
2. Click on publish
3. Now create a new publish profile, and browse the folder where you want to publish your project dll
4. Click on publish so it will create your dll in the folder

DEPLOY .NET CORE APPLICATION ON LINUX

Step 2 - Install required .Net Module on Linux

- Now we have our web application dll and now we need to host it on the Linux environment. First, we need to understand how the deployment works in Linux. .Net applications run on Kestrel servers and we run Apache or Nginx server in Linux environments, which acts as a proxy server and handles the traffic from outside the machine and redirects it to the Kestrel server so we will have Apache or Nginx server as the middle layer.
- Here we will use Apache as a proxy server.
- First, we need to install the .Net core module in our Linux environment. For that run the following commands

```
sudo apt-get update
sudo apt-get install apt-transport-https
sudo apt-get update
sudo apt-get install dotnet-sdk-3.1
sudo apt-get install dotnet-runtime-3.1
sudo apt-get install aspnetcore-runtime-3.1
```

DEPLOY .NET CORE APPLICATION ON LINUX

Step 3 - Install and configure Apache Server

- So now we have all the required .Net packages. I have installed an additional package so if you are running a different project it will help.
- Now install the Apache server,

```
sudo apt-get install apache2
sudo a2enmod proxy proxy_httpproxy_htmlproxy_wstunnel
sudo a2enmod rewrite
```

- Now we need to make a conf file to set up our proxy on Apache. Create the following file:

```
sudo nano /etc/apache2/conf-enabled/netcore.conf
```

- Now copy the following configuration in that file,

DEPLOY .NET CORE APPLICATION ON LINUX

Now copy the following configuration in that file,

```
<VirtualHost *:80>
  ServerName www.DOMAIN.COM
  ProxyPreserveHost On
  ProxyPass / http://127.0.0.1:5000/
  ProxyPassReverse / http://127.0.0.1:5000/
  RewriteEngine on
  RewriteCond %{HTTP:UPGRADE} ^WebSocket$ [NC]
  RewriteCond %{HTTP:CONNECTION} Upgrade$ [NC]
  RewriteRule /(.*) ws://127.0.0.1:5000/$1 [P]
  ErrorLog /var/log/apache2/netcore-error.log
  CustomLog /var/log/apache2/netcore-access.log common
</VirtualHost>
```

DEPLOY .NET CORE APPLICATION ON LINUX

<VirtualHost *:80>

This tag defines the IP and port it will bind Apache so we will access our application from outside our Linux environment through this Ip:Port.

Now restart the Apache server,

- sudo service apache2 restart
- sudoapachectlconfigtest

DEPLOY .NET CORE APPLICATION ON LINUX

Step 4 - Configure and Start Service

- Move your dll to the defined path with the below command.

  "sudo cp -a ~/release/ /var/netcore/"

- Create a service file for our .Net application

  "sudo nano /etc/systemd/system/ServiceFile.service"

- Copy the following configuration in that file and it will run our application,

DEPLOY .NET CORE APPLICATION ON LINUX

```
[Unit]
Description=ASP .NET Web Application
[Service]
WorkingDirectory=/var/netcore
ExecStart=/usr/bin/dotnet /var/netcore/Application.dll
Restart=always
RestartSec=10
SyslogIdentifier=netcore-demo
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production
[Install]
WantedBy=multi-user.target
```

DEPLOY .NET CORE APPLICATION ON LINUX

- *ExecStart=/usr/bin/dotnet /var/netcore/Application.dll* in this line replace Application.dll with your dll name that you want to run.
- Now start the service. Instead of the service name in the below commands use the name of the file made above,
  - sudosystemctl enable {Service Name}
  - sudosystemctl start {Service Name}
- Now your proxy server and kestrel server is running and you can access your application through any ip with port 80.
- To redeploy the code your need to replace the dll and stop and start your service again through the following commands
  - sudosystemctl stop {Service Name}
  - sudosystemctl start {Service Name}

ASP.NET CORE MODULE

- The ASP.NET Core Module is a native IIS module that plugs into the IIS pipeline to either:
  - Host an ASP.NET Core app inside of the IIS worker process (w3wp.exe), called the in-process hosting model.
  - Forward web requests to a backend ASP.NET Core app running the Kestrel server, called the out-of-process hosting model.
- **Supported Windows versions**
  - Windows 7 or later
  - Windows Server 2012 R2 or later
- When hosting in-process, the module uses an in-process server implementation for IIS, called IIS HTTP Server (IISHttpServer).
- When hosting out-of-process, the module only works with Kestrel. The module doesn't function with HTTP.sys.

ASP.NET CORE MODULE

In-process hosting model

- ASP.NET Core apps default to the in-process hosting model.
- The following characteristics apply when hosting in-process:

1. IIS HTTP Server (IISHttpServer) is used instead of Kestrel server. For in-process, CreateDefaultBuilder calls UseIIS to:
    - Register the IISHttpServer.
    - Configure the port and base path the server should listen on when running behind the ASP.NET Core Module.
    - Configure the host to capture startup errors.

ASP.NET CORE MODULE

2. The requestTimeout attribute doesn't apply to in-process hosting.

3. Sharing an app pool among apps isn't supported. Use one app pool per app.

4. When using Web Deploy or manually placing an app_offline.htm file in the deployment, the app might not shut down immediately if there's an open connection. For example, a websocket connection may delay app shut down.

5. The architecture (bitness) of the app and installed runtime (x64 or x86) must match the architecture of the app pool.

6. Client disconnects are detected. The HttpContext.RequestAborted cancellation token is cancelled when the client disconnects.

7. In ASP.NET Core 2.2.1 or earlier, GetCurrentDirectory returns the worker directory of the process started by IIS rather than the app's directory (for example, C:\Windows\System32\inetsrv for w3wp.exe).

ASP.NET CORE MODULE

Out-of-process hosting model

- To configure an app for out-of-process hosting, set the value of the <AspNetCoreHostingModel> property to OutOfProcess in the project file (.csproj):

XMLCopy

```
<PropertyGroup>
<AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

- In-process hosting is set with InProcess, which is the default value.
- The value of <AspNetCoreHostingModel> is case insensitive, so inprocess and outofprocess are valid values.
- Kestrel server is used instead of IIS HTTP Server (IISHttpServer).
- For out-of-process, CreateDefaultBuilder calls UseIISIntegration to:
  - Configure the port and base path the server should listen on when running behind the ASP.NET Core Module.
  - Configure the host to capture startup errors.

DOCKER AND CONTAINERIZATION

- .NET Core can easily run in a Docker container. Containers provide a lightweight way to isolate your application from the rest of the host system, sharing just the kernel, and using resources given to your application. The Docker client has a CLI that you can use to manage images and containers.
- An image is an ordered collection of filesystem changes that form the basis of a container. The image doesn't have a state and is read-only. Much the time an image is based on another image, but with some customization. For example, when you create an new image for your application, you would base it on an existing image that already contains the .NET Core runtime.
- Because containers are created from images, images have a set of run parameters (such as a starting executable) that run when the container starts.
- Role requirements can also be expressed using the new Policy syntax, where a developer registers a policy at startup as part of the Authorization service configuration. This normally occurs in ConfigureServices() in your Startup.cs file.

DOCKER AND CONTAINERIZATION

Containers

- A container is a runnable instance of an image. As you build your image, you deploy your application and dependencies. Then, multiple containers can be instantiated, each isolated from one another. Each container instance has its own filesystem, memory, and network interface.

Registries

- Container registries are a collection of image repositories. You can base your images on a registry image. You can create containers directly from an image in a registry. The relationship between Docker containers, images, and registries is an important concept when architecting and building containerized applications or microservices. This approach greatly shortens the time between development and deployment.
- Docker has a public registry hosted at the Docker Hub that you can use. .NET Core related images are listed at the Docker Hub.

DOCKER AND CONTAINERIZATION

Dockerfile

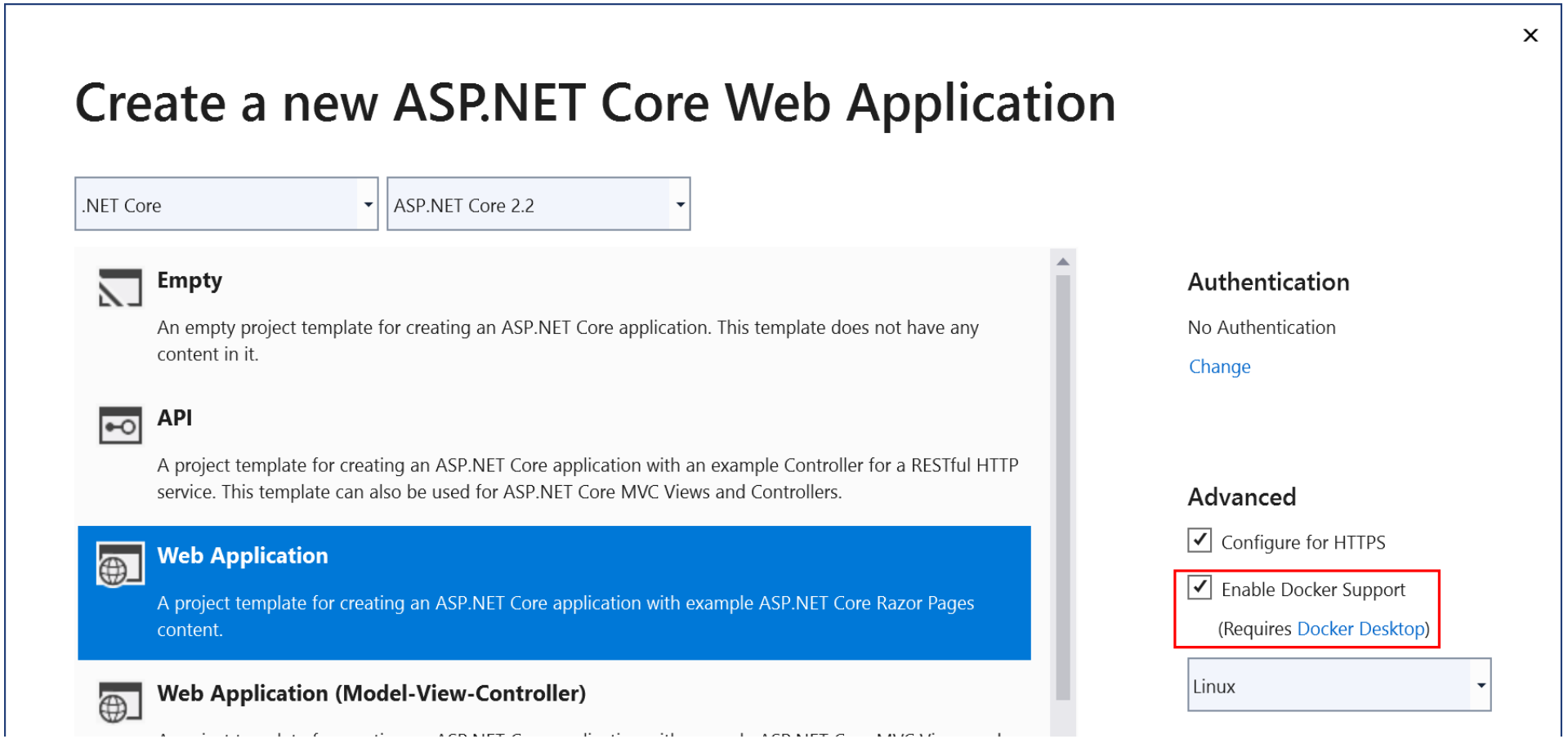
- A Dockerfile is a file that defines a set of instructions that creates an image. Each instruction in the Dockerfile creates a layer in the image. For the most part, when you rebuild the image, only the layers that have changed are rebuilt. The Dockerfile can be distributed to others and allows them to recreate a new image in the same manner you created it. While this allows you to distribute the instructions on how to create the image, the main way to distribute your image is to publish it to a registry.

Docker support in Visual Studio

- Docker support is available for ASP.NET projects, ASP.NET Core projects, and .NET Core and .NET Framework console projects.

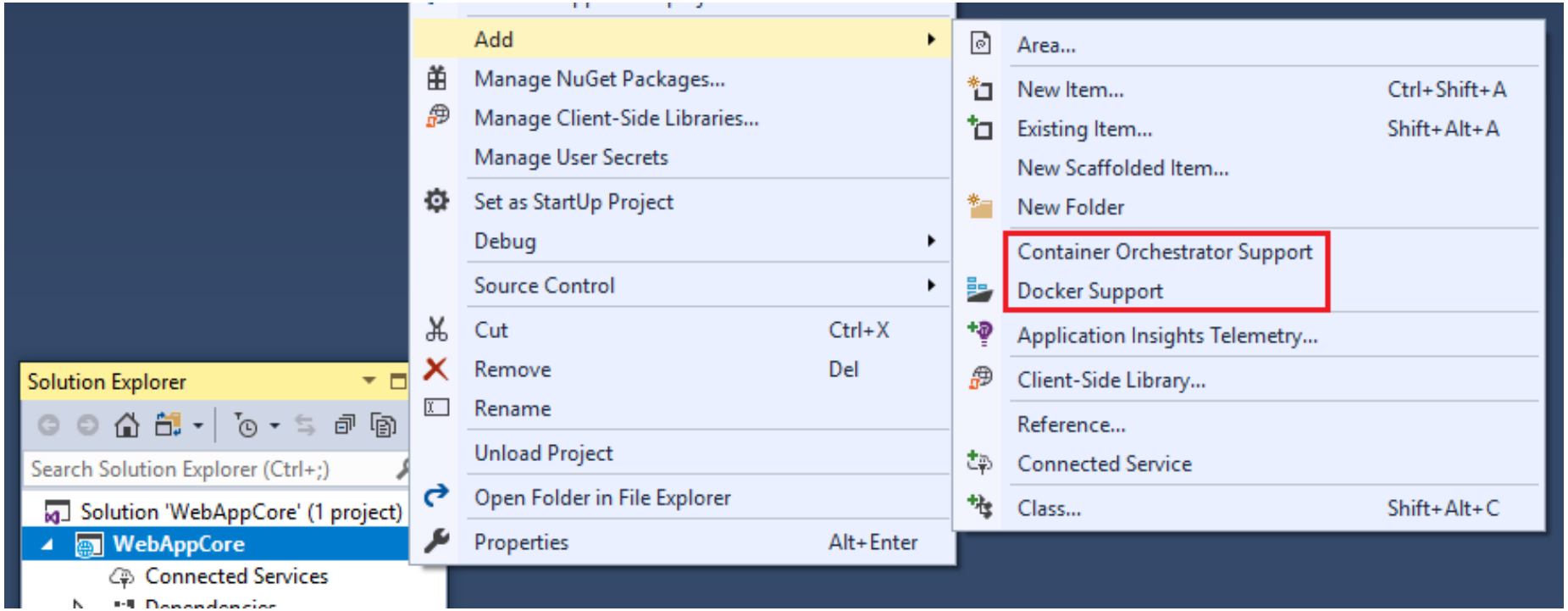
Adding Docker support

- You can enable Docker support during project creation by selecting Enable Docker Support when creating a new project, as shown.



Adding Docker support

- You can add Docker support to an existing project by selecting Add > Docker Support in Solution Explorer. The Add > Docker Support and Add > Container Orchestrator Support commands are located on the right-click menu (or context menu) of the project node for an ASP.NET Core project in Solution Explorer, as shown.



Adding Docker support

- When you add or enable Docker support,

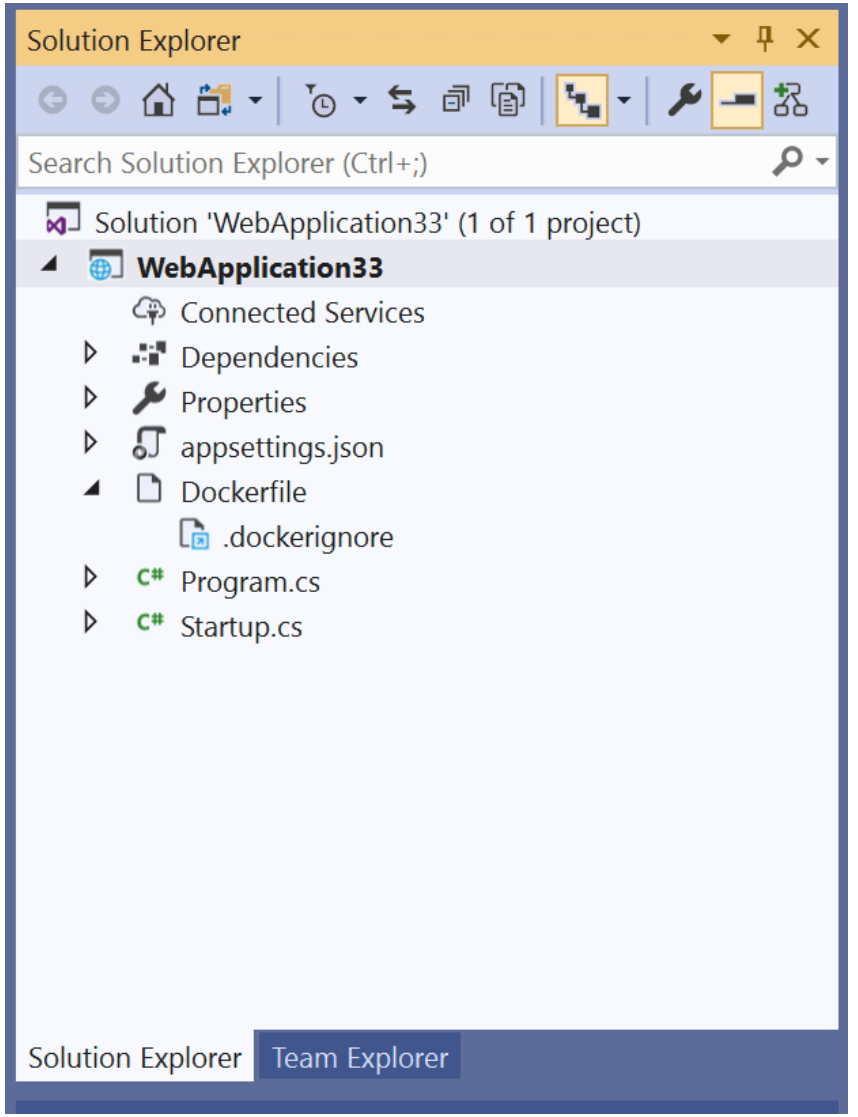
Visual Studio adds the following to the project:

  - a Dockerfile file
  - a .dockerignore file
  - a NuGet package reference to the

Microsoft.VisualStudio.Azure.Containers.Tools.Targets

- The solution looks like this once

you add Docker support:



DEPLOY YOUR ASP.NET CORE APP TO AZURE

Publish to Azure App Service

- https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-aspnet-core-ef-step-05?view=vs-2019

Discussion Exercise

1. What is a web server? List down the of Web Server you can find to host ASP.NET Core Application.
2. Explain about the Hosting Models in ASP.NET Core
3. What is IIS? How can you deploy your ASP.NET Core Application on IIS Server?
4. Write down the details steps to host ASP.NET Application on Linux.
5. What is ASP.NET Core Module? Explain.
6. What is Docker and Containers.
7. How do you deploy ASP.NET Core app to Azure?