MCSE 541: Web Computing and Mining

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

Prof. Dr. Shamim Akhter

@Text Book: Chapter 9,10

Classes and Objects

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

Blueprint/Logical State of a Bike

Construction

Physical State of Bikes

Field Members

- Instance Variable
- Static/Class Variable

Function Members

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

A Simple Class

- Class is created by use of keyword class.

```
modifier class Classname {

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

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

   ...

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

Instance Variable

Handling Static and Non-static variable

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

1)

Program.cs*

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

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

(field) int Program.myVariable

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

2)

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

(field) static int Program.data

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

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

Output

Show output from: Package Manager

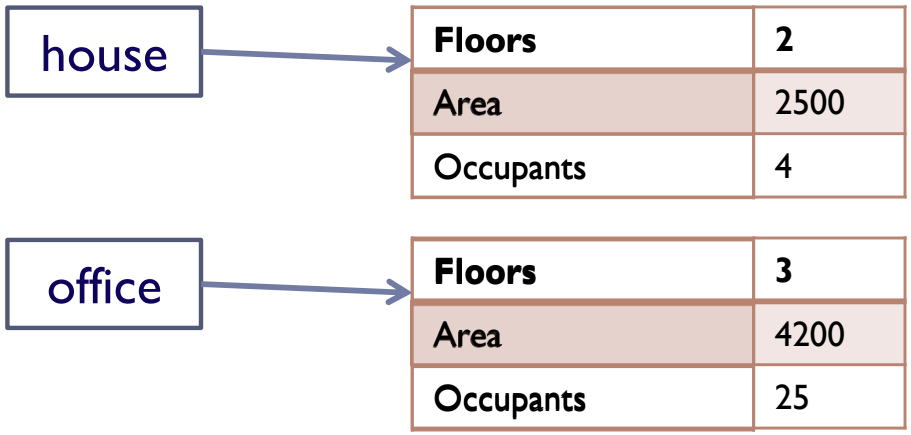
@Text Book: Chapter 6

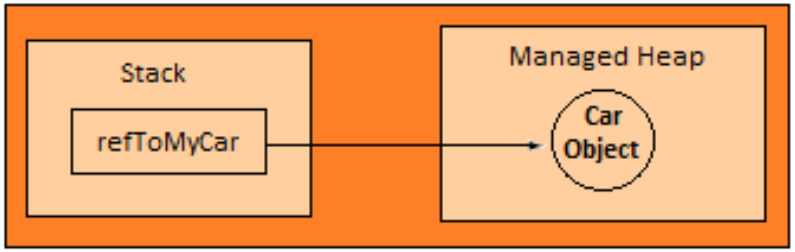
Reference Variable and Assignment

```
using System;

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

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





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

Constructor

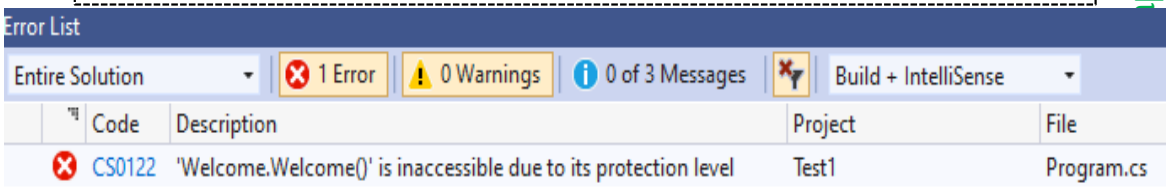
- Constructors
  - are special methods called when a class is instantiated.
  - will not return anything.
  - name is same as class name.
  - By default C# will create default constructor internally.
  - with no arguments and no body is called default constructor.
  - with arguments is called parameterized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

Private Constructor

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

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

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



Parameterized Constructor

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

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

Garbage Collection and Destructors

Memory allocation/deallocation rule:

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

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

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

Destructor Example

```
using System;

  public class Destructor
  {
     public int x;

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

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

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

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

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

- No serialization
- Non deterministic

this Keyword

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

Methods-get( ) and set()

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

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

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

Methods-get( ) and set()

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

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

Methods-get( ) and set()

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

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

Properties

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

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

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

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

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

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

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

Operator Method

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

Two forms of operator methods

General form for overloading a **Unary operator**

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

General form for overloading a **Binary operator**

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

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

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

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

Binary

Solution:

Operator Overloading

Binary Operator Overloading

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

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

        return result;
}
```

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

Unary Operator Overloading

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

    }
}
```

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

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

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

        return result;
}
```

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

++ / -- Postfix and Prefix form

Basically, you've misunderstood how this line works:

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

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

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

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

If you use

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

then that's equivalent to:

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

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

++ / -- Postfix and Prefix form

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

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

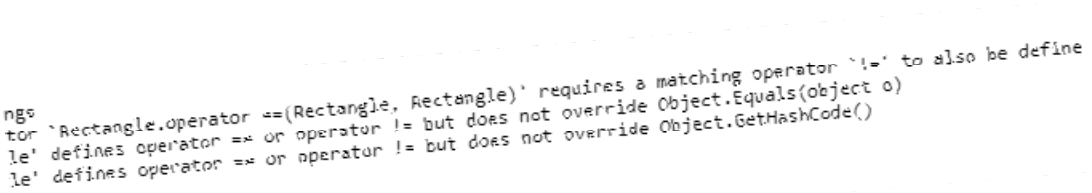
```
class Demo
{
    static void Main(string[] args)
    {
     Program P1 = new Program(2);
     Program P2 = new Program();
     Program P3 = new Program(2);
     P2 = P1++;
     Console.WriteLine("P2.x="+ P2.x+", P1.x=" + P1.x);
     P2 = ++P3;
     Console.WriteLine("P2.x=" + P2.x+", P3.x=" + P3.x);


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

Conditional Operator Overloading

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

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




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

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

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

Overloading the operator methods

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

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

    }
}
```

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

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

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

Indexer

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

General form of a Indexer with get and set accessor

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

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

A simple Indexer

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

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

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

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

Indexers May Not Require an Underline Array

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

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

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

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

MCSE 541:Web Computing and Mining

ASP.NET Core-Middleware and Static files

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

Extension Method

- Additional method to be injected into a class/interface
  - without modifying, deriving or recompiling the original.
  - own custom class, .NET framework classes, or third party classes or interfaces.

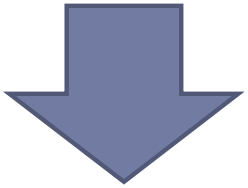
```
int i = 10;
bool result = i.IsGreaterThan(100); //returns false
```

Define an Extension Method

```
namespace ExtensionMethods {
    public static class IntExtensions {
        public static bool IsGreaterThan(this int i, int value) {
            return i > value;
        }
    }
}
```

Binding parameter to bind these methods with int i class.

An extension method can take extra parameters, in addition to an instance of the type that it is extending.



```
using ExtensionMethods;
class Program {
    static void Main(string[] args) {
        int i = 10;
        bool result = i.IsGreaterThan(100);
        Console.WriteLine(result);
    }
}
```

Class Example with Extension Method

```
Using System;
namespace ExtensionMethod {
// Here Geek class contains three methods. Now we want to add two more new methods in it without re-compiling this class
class Geek {
        public void M1()
        {
                Console.WriteLine("Method Name: M1");
         }
         public void M2()
        {
                Console.WriteLine("Method Name: M2");
        }
        public void M3()
        {
                Console.WriteLine("Method Name: M3");
        }
    }
}
```

Defining Extension Methods

```
using System;

namespace ExtensionMethod {

// This class contains M4 and M5 methods. Which we want to add in
//Geek class. NewMethodClass is a static class
static class NewMethodClass {

    public static void M4(this Geek g)
    {
        Console.WriteLine("Method Name: M4");
    }

    public static void M5(this Geek g, string str)
    {
        Console.WriteLine(str);
    }
}

}
```

Call Extension Method

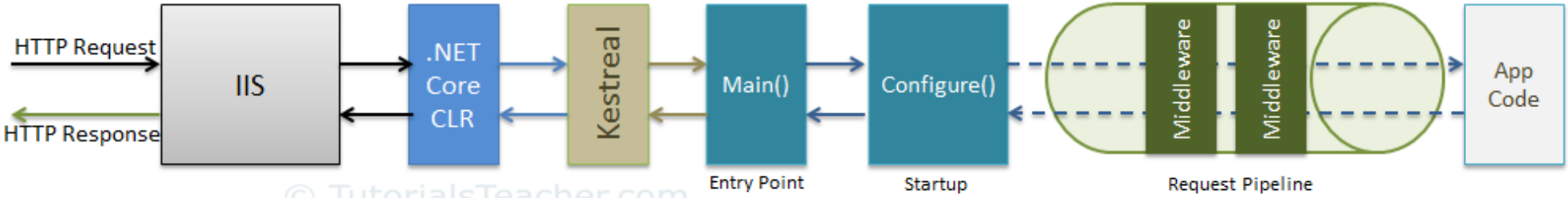
```
using System;

namespace ExtensionMethod {

public class GFG {

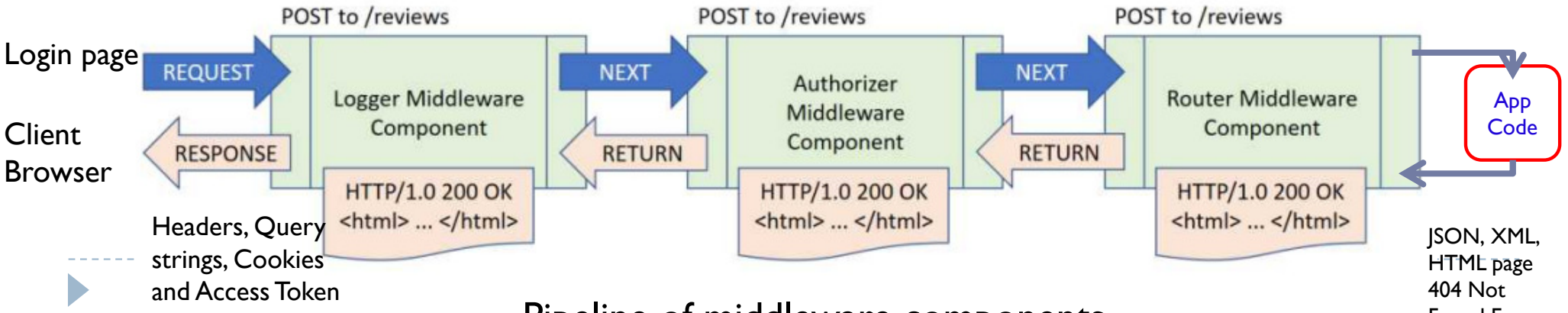
    // Main Method
    public static void Main(string[] args)
    {
        Geek g = new Geek();
        g.M1();
        g.M2();
        g.M3();
        g.M4();
        g.M5("Method Name: M5");
    }
}
}
```

ASP.NET Core Request Processing



Middleware

- A new concept in ASP.NET Core
- A component (class) which is executed on every http request in ASP.NET Core application.
  - How the application behaves if there is an error.
- In the classic ASP.NET,
  - HttpHandlers and HttpModules were part of the request pipeline.
- Middleware is similar to HttpHandlers and HttpModules
  - performs user authentication and authorization.



Pipeline of middleware components

Configure Middleware

- Middleware is configured into the Startup class's Configure method where an IApplicationBuilder interface instance is injected.
- IApplicationBUilder Interface, which is used to defines a class that provides the mechanisms to configure an application's request pipeline.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World!");
                                    });
          // other code removed for clarity..
    }
}
```

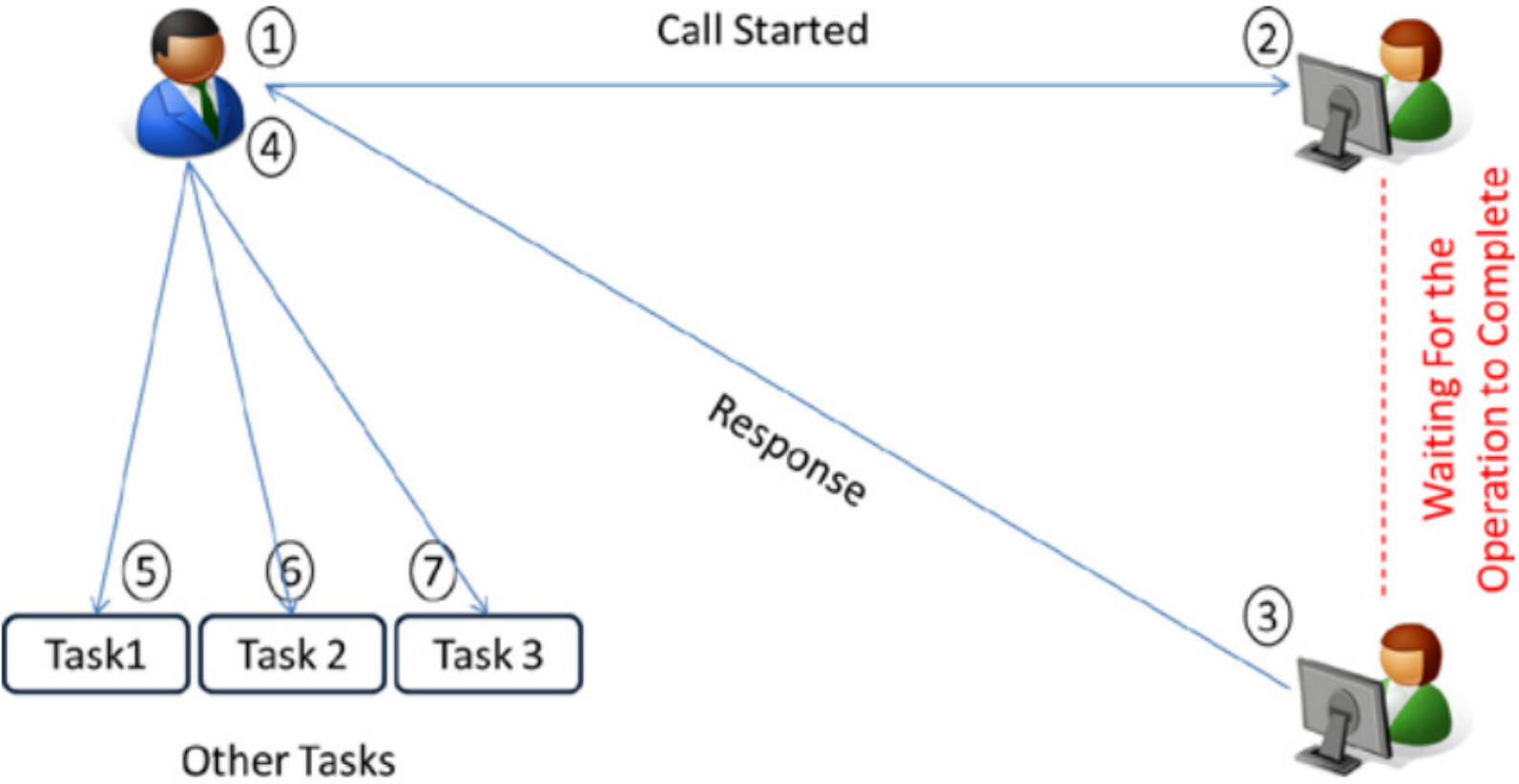
Run() is an extension method on IApplicationBuilder instance which adds a terminal middleware to the application's request pipeline.

The above configured middleware returns a response with a string "Hello World!" for each request.

Synchronous vs Asynchronous

Assume that a few days ago, after I bought a product from Company A, I began having a problem with it. I called the company's support center to explain and sort out the problem. After listening to my explanation, the customer service representative asked me to wait a few minutes. While he tried to solve the problem, I was left hanging on the telephone. The important part here is that I couldn't do anything else until the representative got back to me. Any other tasks I needed to perform were, like me, left on hold while I waited for my call to end.

My situation in this scenario can be related to **synchronous** processing of a long-running operation (Figure 2-1).



Let's construct another scenario. This time I bought a product from Company B; again, there was a problem. I called Company B's support center and explained the problem to a customer service representative. This time, however, since the representative said she would call me back as soon as the problem got sorted out, I could hang up the phone. This allowed me to see to other tasks while Company B's people worked on my problem. Later, the representative called me back and informed me of the problem's resolution. This scenario resembles how an **asynchronous** operation works (see Figure 2-2).

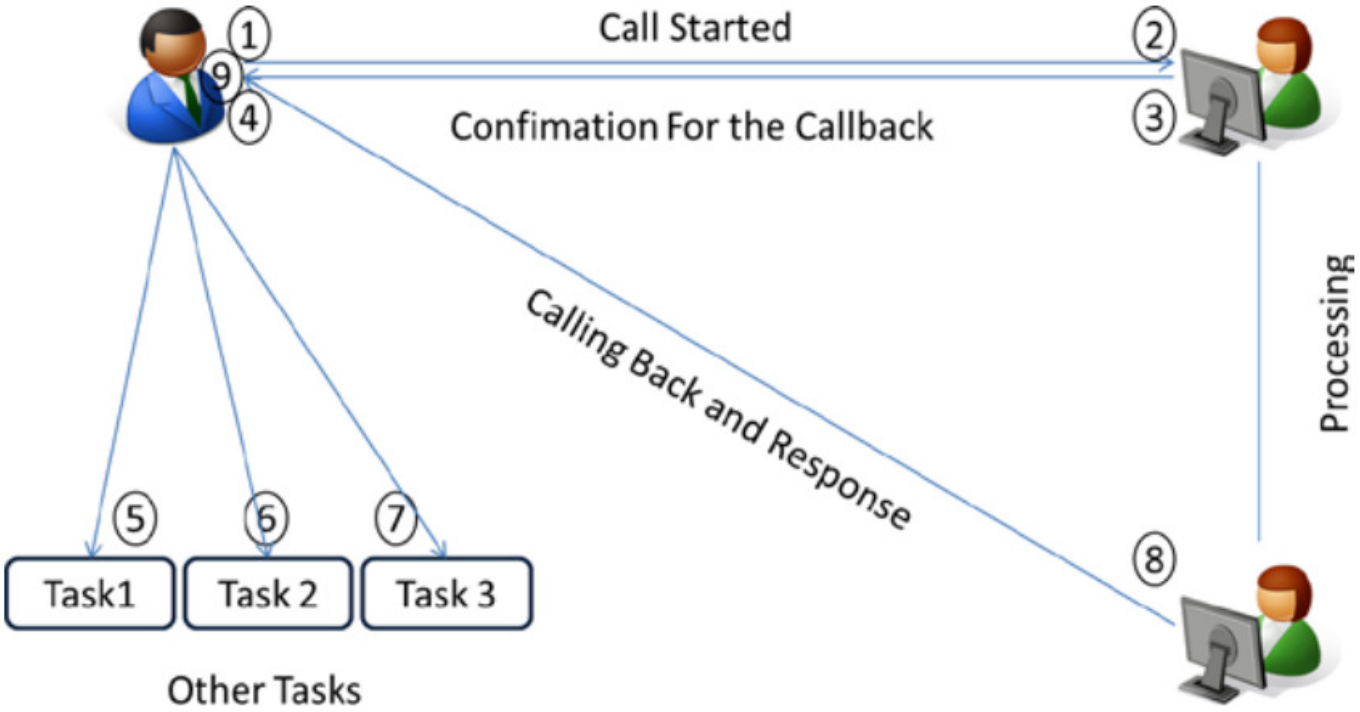


Figure 2-2. *An asynchronous phone call between me and Company B's customer service representative*

Run Method Signature

public static void Run(this IApplicationBuilder app, RequestDelegate handler)

Delegate Signature

Request delegates are used to build the request pipeline.
The request delegates handle each HTTP request.

public delegate Task RequestDelegate(HttpContext context);

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.Run(MyMiddleware);
    }

    private Task MyMiddleware(HttpContext context)
    {
        return context.Response.WriteAsync("Hello World! ");
    }
}
```

- MyMiddleware function is not asynchronous
- Thus will block the thread till the time it completes the execution.
- So, make it asynchronous by using async and await to improve performance and scalability.

Asynchronous MyMiddleware

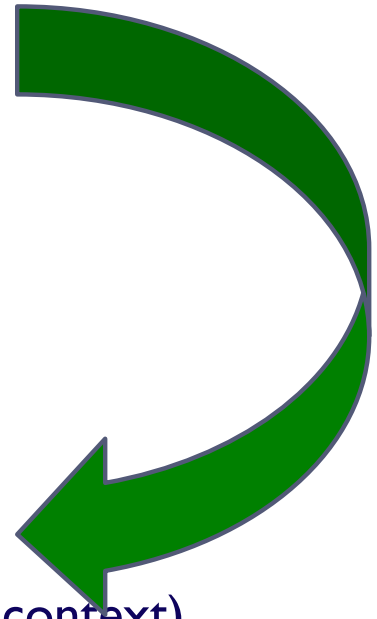
```
private async Task MyMiddleware(HttpContext context)
{
    await context.Response.WriteAsync("Hello World! ");
}
```

- The await operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes.
- When the asynchronous operation completes, the await operator returns the result of the operation, if any.

Run method calls with Delegate

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        Microsoft.AspNetCore.Http.RequestDelegate Rd = MyMiddleware;
            //public delegate Task RequestDelegate(HttpContext context);
         app.Run(Rd);
    }

    private Task MyMiddleware(HttpContext context)
    {
       return context.Response.WriteAsync("Hello World! ");
    }
}
```



```
Microsoft.AspNetCore.Http.RequestDelegate Rd = (HttpContext context)
=>  context.Response.WriteAsync("Hello World10! ");
App.Run(Rd);
```

```
app.Run((HttpContext context) => context.Response.WriteAsync("Hello World10! "));
```

Configure Multiple Middlewares

There will be multiple middleware components in ASP.NET Core application which will be executed sequentially.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World1!");
                                    });
       // the following will never be executed
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World2!");
                                    });

    }
}
```

Use() Extension Method

We can use Use() method to configure multiple middleware in the order we like.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
      app.Use(async (context, next) => { await context.Response.WriteAsync("Hello World1!");

      await  next();
                                          });

       // the following will never be executed
      app.Run(async (context) => { await context.Response.WriteAsync("Hello World2!");
                                   });

    }
}
```

Add Built-in Middleware Via NuGet

Middleware	Description
Authentication	Adds authentication support.
CORS	Configures Cross-Origin Resource Sharing.
Routing	Adds routing capabilities for MVC or web form
Session	Adds support for user session.
StaticFiles	Adds support for serving static files and directory browsing.
Diagnostics	Adds support for reporting and handling exceptions and errors.

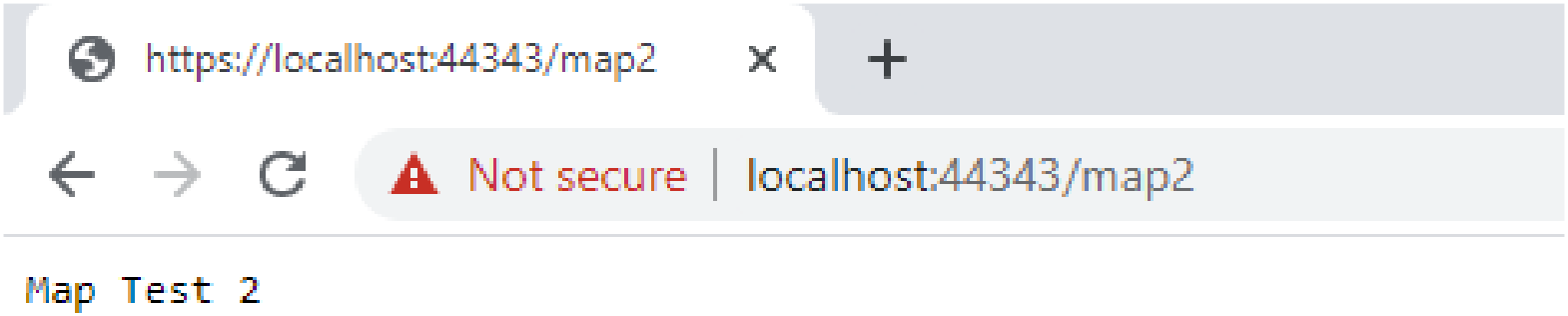
Map Extension Method

- Map extensions are used as a convention for branching the pipeline.
- Map branches the request pipeline based on matches of the given request path.
  - If the request path starts with the given path, the branch is executed.

```
private static void HandleMapTest1(IApplicationBuilder app) {
                app.Run(async context => {
                    await context.Response.WriteAsync("Map Test 1"); });
                }
private static void HandleMapTest2(IApplicationBuilder app) {
                app.Run(async context => {
                      await context.Response.WriteAsync("Map Test 2"); }); }

public void Configure(IApplicationBuilder app) {
                app.Map("/map1", HandleMapTest1);
                app.Map("/map2", HandleMapTest2);
                app.Run(async context =>
 { await context.Response.WriteAsync("Hello from non-Map delegate. <p>"); }); }
```

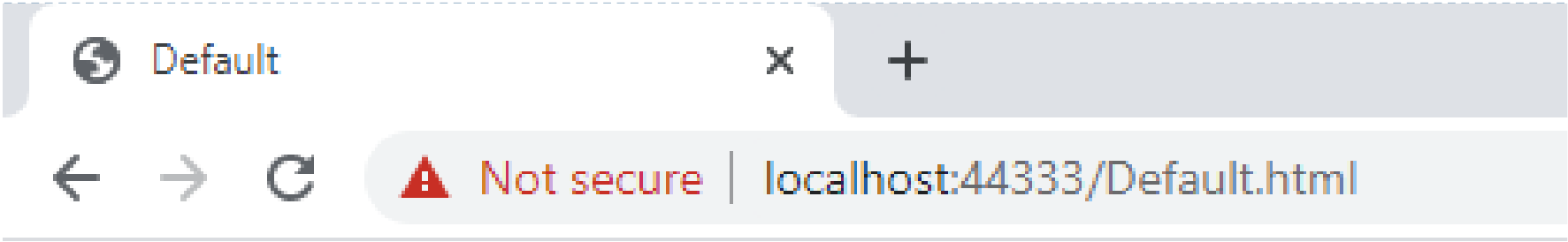
Request	Response
localhost:1234	Hello from non-Map delegate.
localhost:1234/map1	Map Test 1
localhost:1234/map2	Map Test 2
localhost:1234/map3	Hello from non-Map delegate.



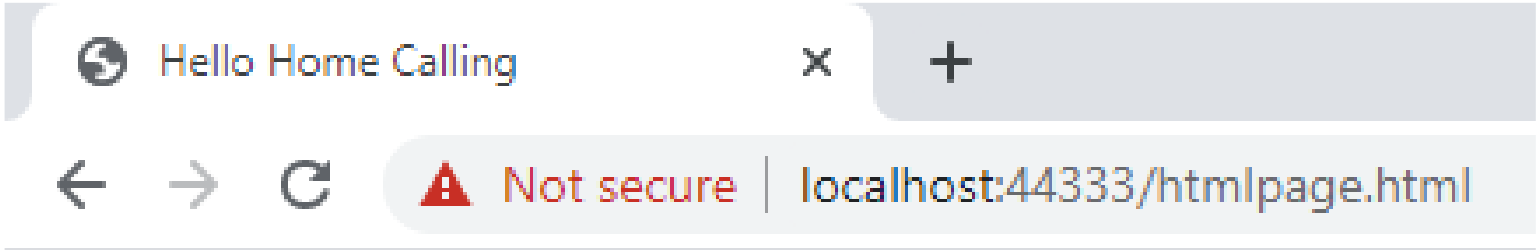
Serving Static Files

- We must include Microsoft.AspNetCore.StaticFiles middleware in the request pipeline.
- However, Microsoft.AspNetCore.All includes the middleware. Thus, we do not need to install for ASP.NET Core 2.x/3.1

Default html at wwwroot folder



Hi Hello From Default Webpage



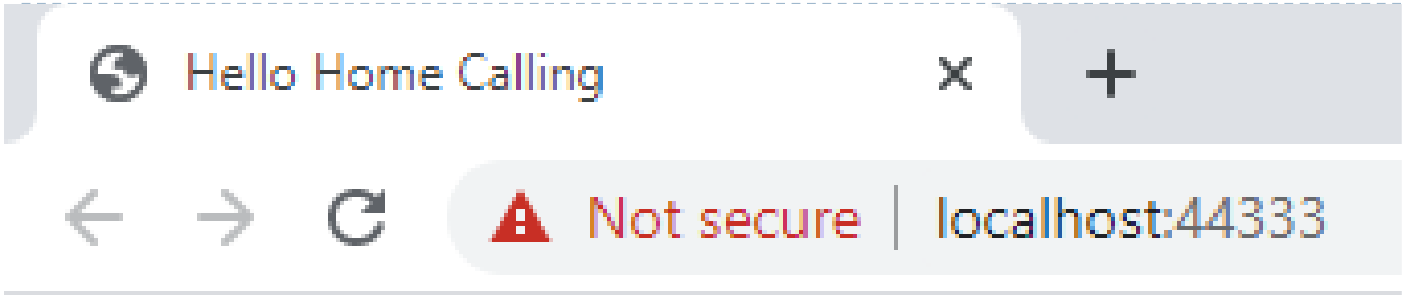
It is just a demo.

app.UseStaticFiles();

```
app.UseStaticFiles();

app.Run(async (context) =>
{
    await context.Response.WriteAsync("Hello World!");
});
```

Setting htmlpage.html is default page



It is just a demo.

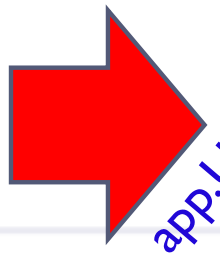
```
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{

    DefaultFilesOptions options = new DefaultFilesOptions();
    options.DefaultFileNames.Clear();
    options.DefaultFileNames.Add("htmlpage.html");

    app.UseDefaultFiles(options);

    app.UseStaticFiles();
```

app.UseDefaultFiles();
app.UseStaticFiles();



app.UseFileServer();

Will calls the Default.html file

Change the htmlpage.html to the following

```
<!DOCTYPE html>
<html><body>
<p>Enter a number and click OK:</p>
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>
<p>If the number is less than 100 or greater than 300, an error message
  will be displayed.</p>
<p id="demo"></p>
<script>
function myFunction() {
  var inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML =inpObj.validationMessage;
  } else {
    document.getElementById("demo").innerHTML = "Input OK";
  }
}
</script></body></html>
```

rangeOverflow

```
<input id="id1" type="number" max="100">
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
  let text = "Value OK";
  if (document.getElementById("id1").validity.rangeOverflow) {
    text = "Value too large";
  }
}
</script
```

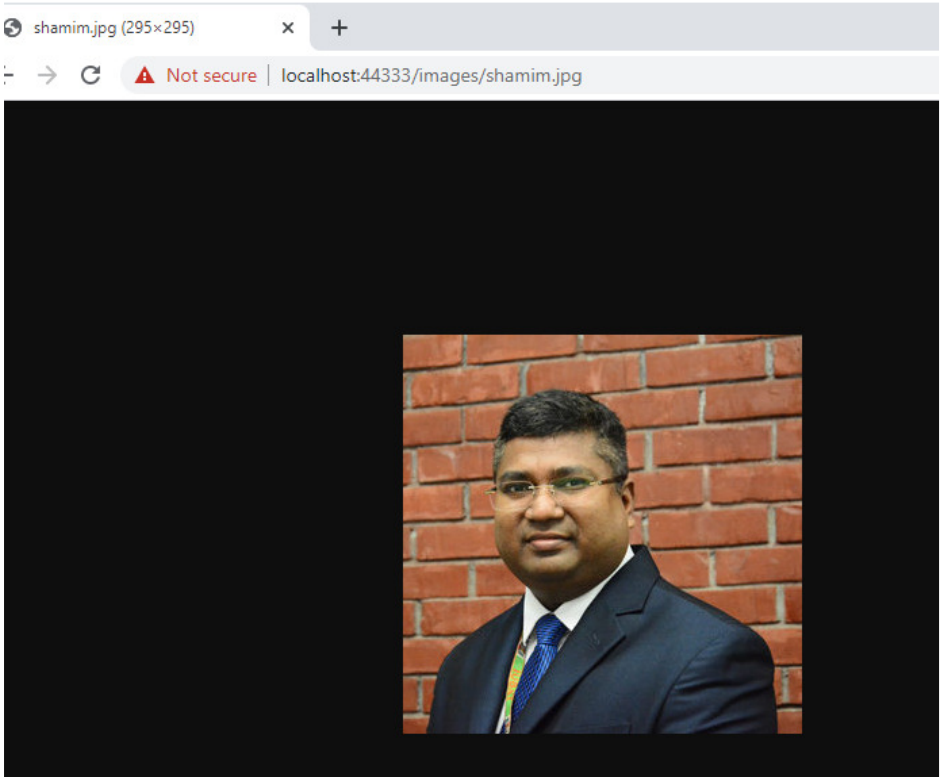
rangeUnderflow

```
<input id="id1" type="number" min="100">
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
  let text = = "Value OK";
  if (document.getElementById("id1").validity.
rangeUnderflow) {
    text = "Value too small";
  }
}
</script>
```

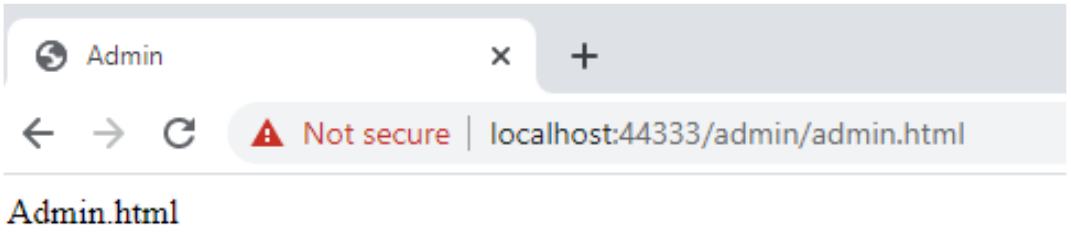
Java scripts calls from JS file




```
app.UseStaticFiles(new StaticFileOptions()
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), @"Images")),
    RequestPath = new PathString("/images")

});
```

Get file from Admin folder



Admin.html

```
//app.UseFileServer();
app.UseStaticFiles(new StaticFileOptions()
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), "admin")),
    RequestPath = new PathString("/Admin")

});
```

Adding a JSON file

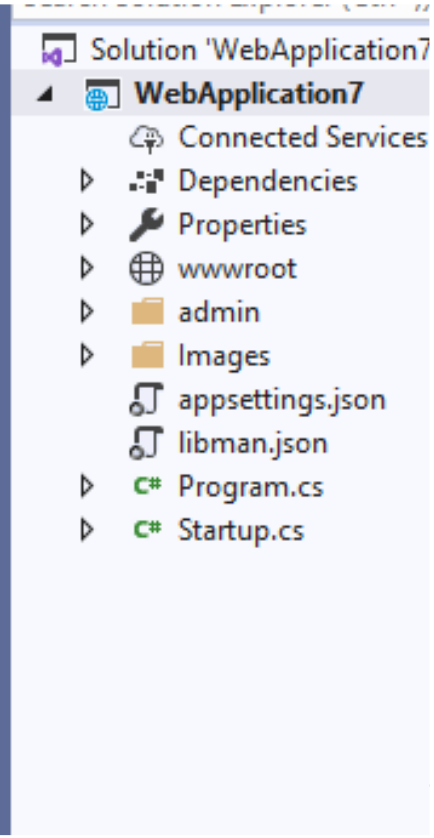
appsettings.json

Schema: http://json.schemastore.org/appsettings

```
1  {
2    "Message":"Hello, from JSON Configuration"
3  }
4
```

```
2 references | 0 exceptions
public IConfiguration Configuration { get; set; }
0 references | 0 exceptions
public Startup()
{
    var builder = new ConfigurationBuilder()
      .SetBasePath(Directory.GetCurrentDirectory())
      .AddJsonFile("appsettings.json");
    Configuration = builder.Build();
}
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)...
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())...

    app.Run(async (context) =>
    {
          var msg = Configuration["Message"];
        await context.Response.WriteAsync(msg);
    });
```

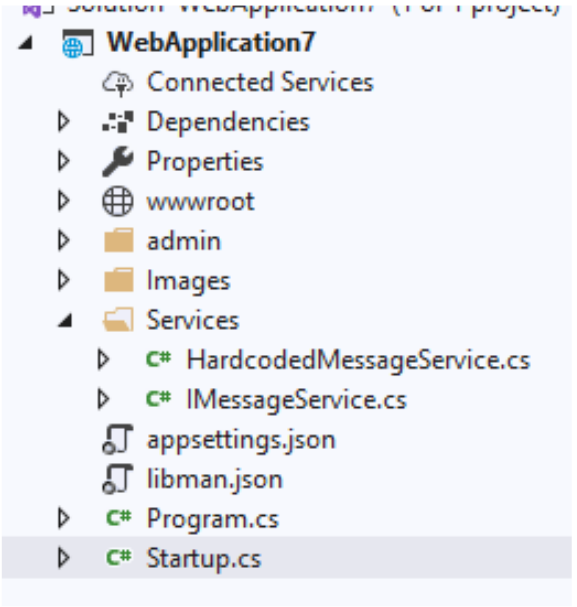


Adding a Service

- Add one interface @ Service folder

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication7.Services
{
    3 references
    public interface IMessageService
    {
        2 references | 0 exceptions
        string GetMessage();
    }
}
```



- Add one class @HardcodedMessageService

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication7.Services
{
    1 reference
    public class HardcodedMessageService : IMessageService
    {
        2 references | 0 exceptions
        public string GetMessage()
        {
            return "Hardcoed message from a service.";
        }
    }
}
```

Configure A Service

```
using WebApplication7.Services;


public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMessageService, HardcodedMessageService>();

}
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    IMessageService msg)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

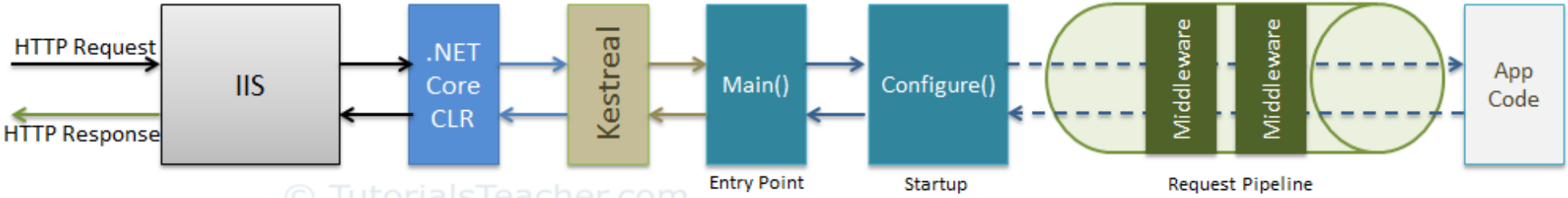
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(msg.GetMessage());
    });
}
```

MCSE 541:Web Computing and Mining

ASP.NET Core-Use of Middleware

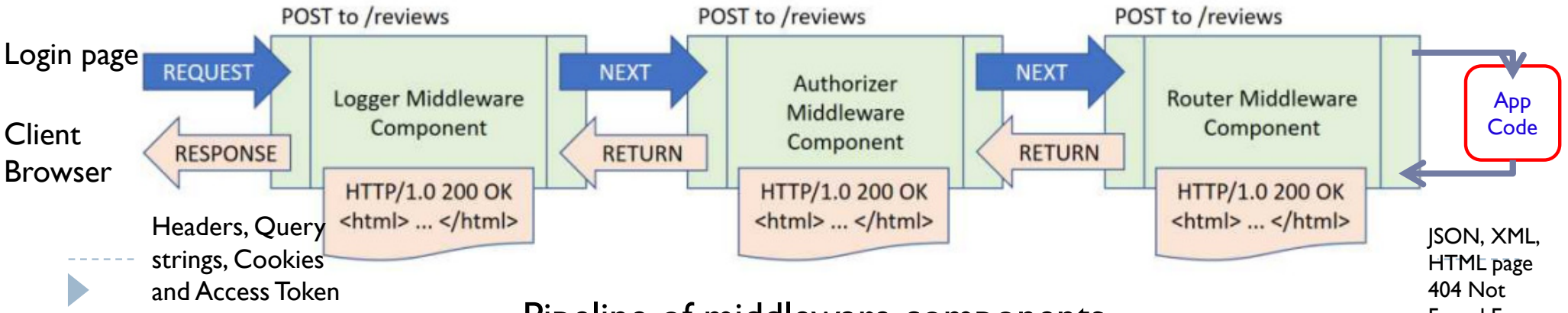
Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

ASP.NET Core Request Processing



Middleware

- A new concept in ASP.NET Core
- A component (class) which is executed on every http request in ASP.NET Core application.
  - E.g. How the application behaves if there is an error.
- In the classic ASP.NET,
  - HttpHandlers and HttpModules were part of the request pipeline.
- Middleware is similar to HttpHandlers and HttpModules
  - E.g. performs user authentication and authorization.

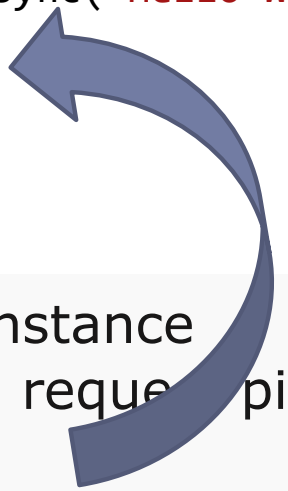


Pipeline of middleware components

Configure Middleware

- Middleware is configured into the Startup class's Configure method where an IApplicationBuilder interface instance is injected.
- IApplicationBuilder Interface, which is used to defines a class that provides the mechanisms to configure an application's request pipeline.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World!");
                                    });
          // other code removed for clarity..
    }
}
```



Run() is an **extension method** on IApplicationBuilder instance which adds a terminal middleware to the application's request pipeline.

The above configured middleware returns a response with a string "Hello World!" for each request.

Extension Method

- Additional method to be injected into a class/interface
  - without modifying, deriving or recompiling the original class.
  - own custom class, .NET framework classes, or third party classes or interfaces can use extension method.

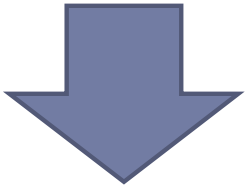
```
int i = 10;
bool result = i.IsGreaterThan(100); //returns false
```

Define an Extension Method

```
namespace ExtensionMethods {
    public static class IntExtensions {
        public static bool IsGreaterThan(this int i, int value) {
            return i > value;
        }
    }
}
```

Binding parameter to bind these methods with int i class.

An extension method can take extra parameters, in addition to an instance of the type that it is extending.



```
using ExtensionMethods;
class Program {
    static void Main(string[] args) {
        int i = 10;
        bool result = i.IsGreaterThan(100);
        Console.WriteLine(result);
    }
}
```

Class Example with Extension Method

```
Using System;
namespace ExtensionMethod {
// Here Geek class contains three methods. Now we want to add two more new methods in it without re-compiling this class
class Geek {
        public void M1()
        {
                Console.WriteLine("Method Name: M1");
         }
         public void M2()
        {
                Console.WriteLine("Method Name: M2");
        }
        public void M3()
        {
                Console.WriteLine("Method Name: M3");
        }
    }
}
```

Defining Extension Methods

```
using System;

namespace ExtensionMethod {

// This class contains M4 and M5 methods. Which we want to add in
//Geek class. NewMethodClass is a static class
static class NewMethodClass {

    public static void M4(this Geek g)
    {
        Console.WriteLine("Method Name: M4");
    }

    public static void M5(this Geek g, string str)
    {
        Console.WriteLine(str);
    }
}

}
```

Call Extension Method

```
using System;

namespace ExtensionMethod {

public class GFG {

    // Main Method
    public static void Main(string[] args)
    {
        Geek g = new Geek();
        g.M1();
        g.M2();
        g.M3();
        g.M4();
        g.M5("Method Name: M5");
    }
}
}
```

Configure Middleware

- Middleware is configured into the Startup class's Configure method where an IApplicationBuilder interface instance is injected.
- IApplicationBUilder Interface, which is used to defines a class that provides the mechanisms to configure an application's request pipeline.

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                          ILoggerFactory loggerFactory)
    { //configure middleware using IApplicationBuilder here..
        app.Run(async (context) => { await context.Response.WriteAsync("Hello World!");
                                    });
          // other code removed for clarity..
     }
}
```

Why- async and await?

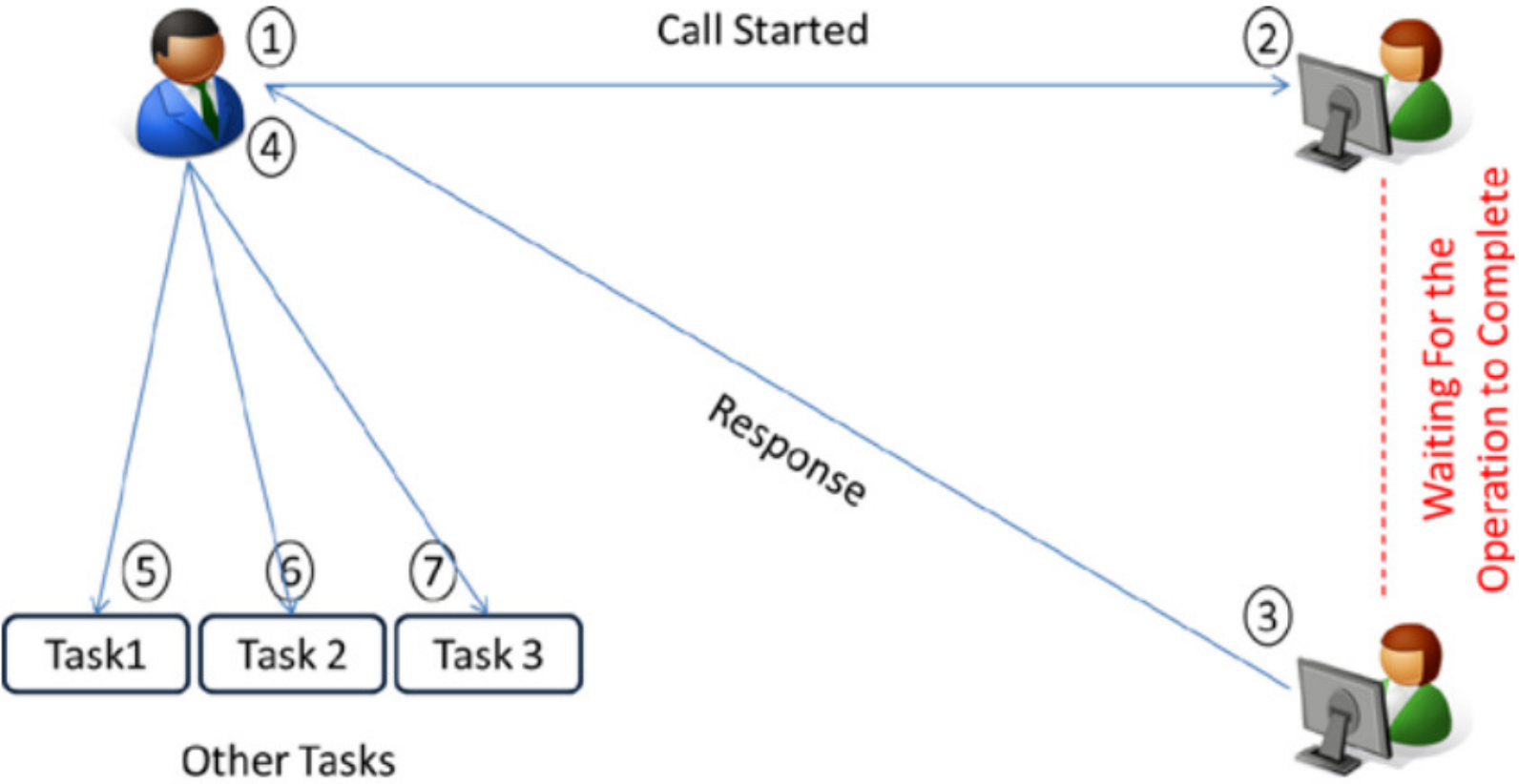
Run() is an extension method on IApplicationBuilder instance which adds a terminal middleware to the application's request pipeline.

The above configured middleware returns a response with a string "Hello World!" for each request.

Synchronous vs Asynchronous

Assume that a few days ago, after I bought a product from Company A, I began having a problem with it. I called the company's support center to explain and sort out the problem. After listening to my explanation, the customer service representative asked me to wait a few minutes. While he tried to solve the problem, I was left hanging on the telephone. The important part here is that I couldn't do anything else until the representative got back to me. Any other tasks I needed to perform were, like me, left on hold while I waited for my call to end.

My situation in this scenario can be related to **synchronous** processing of a long-running operation (Figure 2-1).



Synchronous vs Asynchronous

Let's construct another scenario. This time I bought a product from Company B; again, there was a problem. I called Company B's support center and explained the problem to a customer service representative. This time, however, since the representative said she would call me back as soon as the problem got sorted out, I could hang up the phone. This allowed me to see to other tasks while Company B's people worked on my problem. Later, the representative called me back and informed me of the problem's resolution. This scenario resembles how an **asynchronous** operation works (see Figure 2-2).

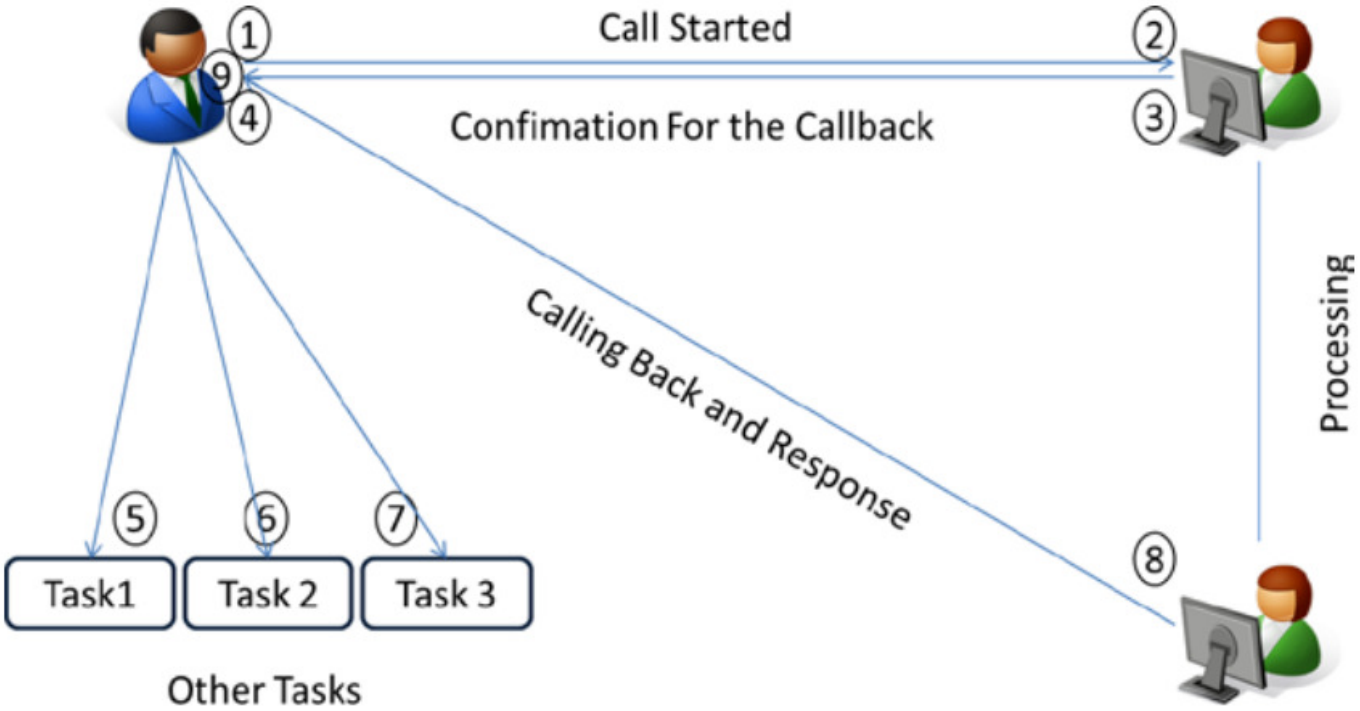


Figure 2-2. *An asynchronous phone call between me and Company B's customer service representative*

Run Method Signature

public static void Run(this IApplicationBuilder app, RequestDelegate handler)

Delegate Signature

Request delegates are used to build the request pipeline.
The request delegates handle each HTTP request.

public delegate Task RequestDelegate(HttpContext context);

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.Run(MyMiddleware);
    }

    private Task MyMiddleware(HttpContext context)
    {
        return context.Response.WriteAsync("Hello World! ");
    }
}
```

- MyMiddleware function is not asynchronous
- Thus will block the thread till the time it completes the execution.
- So, make it asynchronous by using async and await to improve performance and scalability.

Asynchronous MyMiddleware

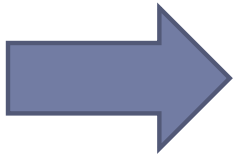
```
private async Task MyMiddleware(HttpContext context)
{
    await context.Response.WriteAsync("Hello World! ");
}
```

- The await operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes.
- When the asynchronous operation completes, the await operator returns the result of the operation, if any.

https://www.c-sharpcorner.com/article/async-and-await-in-c-sharp/

```
using System;
using System.Threading.Tasks;

namespace ASynExample1
{
    class Program
    {
        ***
    }
}
```



```
static async Task Main(string[] args)
 {
     await callMethod();
     Console.ReadKey();
 }

 public static async Task callMethod()
 {
     Method2();
     var count = await Method1();
     Method3(count);
 }
```

```
public static async Task<int> Method1()
{
    int count = 0;
    await Task.Run(() =>
    {
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine(" Method 1");
            count += 1;
        }
    });
    return count;
}

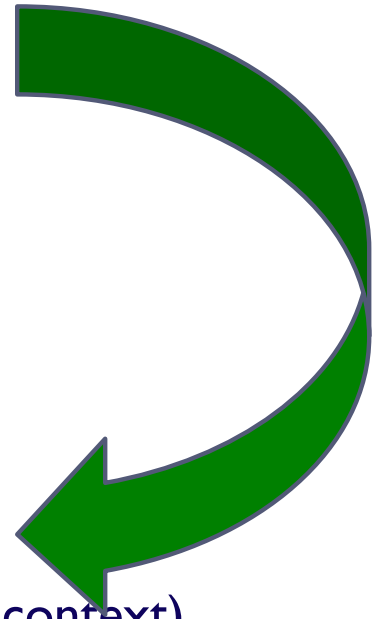
public static void Method2()
{
    for (int i = 0; i < 25; i++)
    {
        Console.WriteLine(" Method 2");
    }
}

public static void Method3(int count)
{
    Console.WriteLine("Total count is " + count);
}
```

Run method calls with Delegate

```
public class Startup {
    public Startup() { }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        Microsoft.AspNetCore.Http.RequestDelegate Rd = MyMiddleware;
            //public delegate Task RequestDelegate(HttpContext context);
         app.Run(Rd);
    }

    private Task MyMiddleware(HttpContext context)
    {
       return context.Response.WriteAsync("Hello World! ");
    }
}
```



```
Microsoft.AspNetCore.Http.RequestDelegate Rd = (HttpContext context)
=>  context.Response.WriteAsync("Hello World10! ");
App.Run(Rd);
```

```
app.Run((HttpContext context) => context.Response.WriteAsync("Hello World10! "));
```

Configure Multiple Middlewares

There will be multiple middleware components in ASP.NET Core application which will be executed sequentially.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World1!");
                                    });
       // the following will never be executed
       app.Run(async (context) => { await context.Response.WriteAsync("Hello World2!");
                                    });

    }
}
```

Use() Extension Method

We can use Use() method to configure multiple middleware in the order we like.

```
public class Startup {
   public Startup() { }
   public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                         ILoggerFactory loggerFactory)
   { //configure middleware using IApplicationBuilder here..
      app.Use(async (context, next) => { await context.Response.WriteAsync("Hello World1!");

      await  next();
                                          });

       // the following will never be executed
      app.Run(async (context) => { await context.Response.WriteAsync("Hello World2!");
                                   });

    }
}
```

Add Built-in Middleware Via NuGet

Middleware	Description
Authentication	Adds authentication support.
CORS	Configures Cross-Origin Resource Sharing.
Routing	Adds routing capabilities for MVC or web form
Session	Adds support for user session.
StaticFiles	Adds support for serving static files and directory browsing.
Diagnostics	Adds support for reporting and handling exceptions and errors.

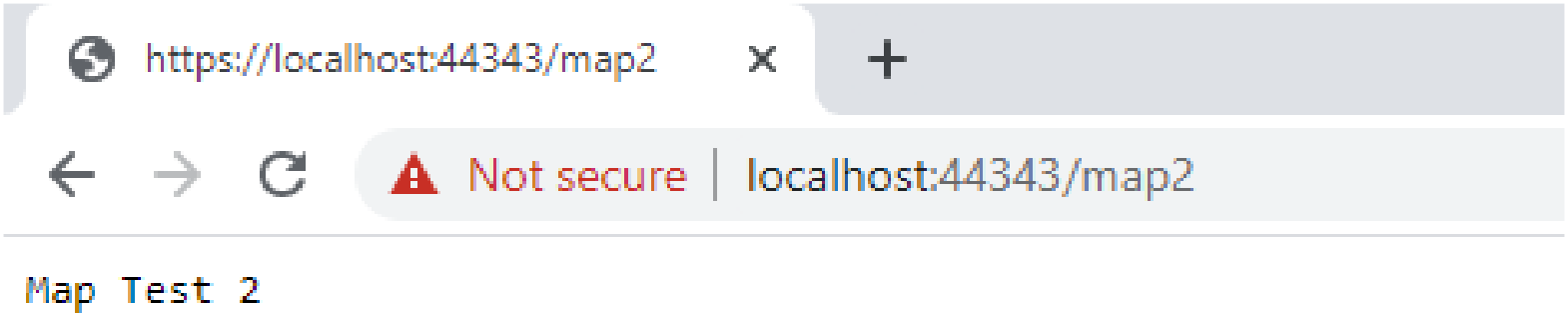
Map Extension Method

- Map extensions are used as a convention for branching the pipeline.
- Map branches the request pipeline based on matches of the given request path.
  - If the request path starts with the given path, the branch is executed.

```
private static void HandleMapTest1(IApplicationBuilder app) {
                app.Run(async context => {
                    await context.Response.WriteAsync("Map Test 1"); });
                }
private static void HandleMapTest2(IApplicationBuilder app) {
                app.Run(async context => {
                      await context.Response.WriteAsync("Map Test 2"); }); }

public void Configure(IApplicationBuilder app) {
                app.Map("/map1", HandleMapTest1);
                app.Map("/map2", HandleMapTest2);
                app.Run(async context =>
 { await context.Response.WriteAsync("Hello from non-Map delegate. <p>"); }); }
```

Request	Response
localhost:1234	Hello from non-Map delegate.
localhost:1234/map1	Map Test 1
localhost:1234/map2	Map Test 2
localhost:1234/map3	Hello from non-Map delegate.



Concrete middleware using Map() in ASP.NET Core 2.x

- In ASP.NET Core 2.x, you could include the middleware as a conditional branch in your middleware pipeline, so that it only executes if the app receives a request starting with "/version":

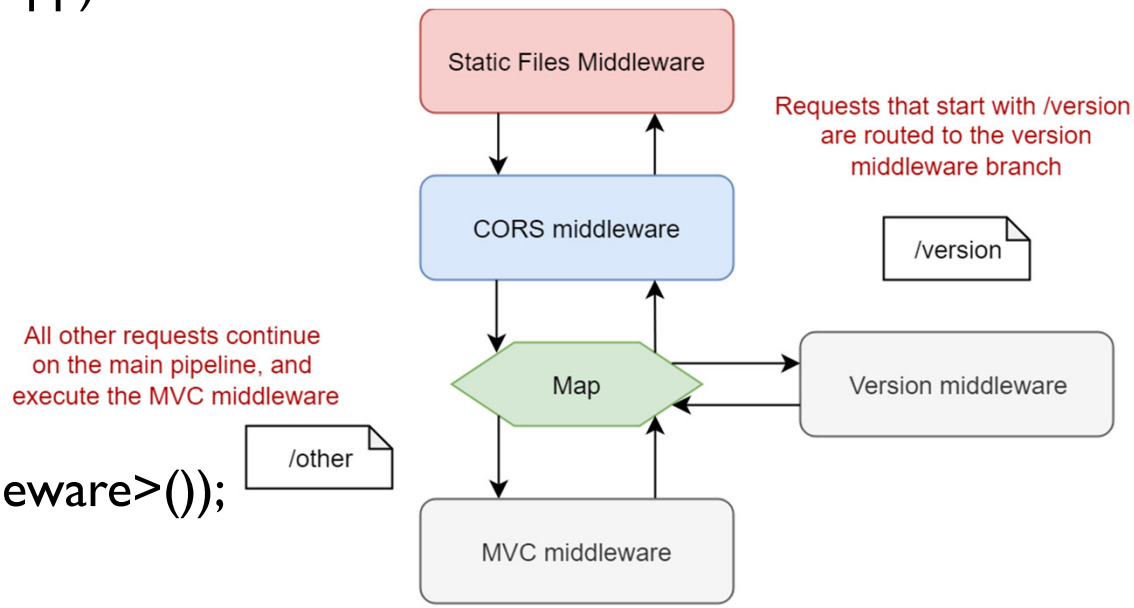
```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    app.UseCors();

    app.Map("/version", versionApp =>

versionApp.UseMiddleware<VersionMiddleware>());

    app.UseMvcWithDefaultRoute();
}
```



- https://andrewlock.net/converting-a-terminal-middleware-to-endpoint-routing-in-aspnetcore-3/

Before ASP.NET Core 3.0

Before ASP.NET Core 3.0 the Route resolution & invoking of the Route were part of the MVC Middleware. We defined the routes while configuring the MVC is using the app.UseMvc as shown below

```
app.UseMvc(routes =>  {
      routes.MapRoute("default",
                  "{controller}/{action}");
});
```

When a new request arrives, the routing middleware parses the incoming URL.

- A matching route is searched in the RouteCollection.
- If a matching route is found, control is passed to the RouteHandler
- If a matching route is not found, the next middleware is invoked.

What is an Endpoint

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

- Metadata of the request.
- The delegate (Request handler) that ASP.NET core uses to process the request.

We define the Endpoint at the application startup using the UseEndpoints method.

The ASP.NET Core routing is now not tied up only to the MVC Endpoints. You can define an Endpoint, which can also hit a Razor page, SignalR etc.

The Endpoint Routing has three components

- Defining the Endpoints.
- Route matching & Constructing an Endpoint (UseRouting).
- Endpoint Execution (UseEndpoints).

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

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

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

Configure the Endpoint

We configure the Endpoint in the app.UseEndpoints method.

The following method adds the default MVC Conventional route using the MapControllerRoute method.

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

To setup an attribute based Routing use the method MapControllers. We use the attribute based Routing to create a route to Rest API (Web API Controller Action Method). You can also use it to create a route to MVC Controller action method.

```
endpoints.MapControllers();
```

MapControllerRoute & MapControllers methods hides all the complexities of setting up the Endpoint from us. Both sets up the Endpoint to Controller action methods

You can also create an Endpoint to a custom delegate using MapGet method. MapGet accepts two argument. One is Route Pattern (/ in the example) and a request delegate.

```
endpoints.MapGet("/", async context =>
  {
    await context.Response.WriteAsync("Hello World");
  });
```

Converting the middleware to endpoint routing

- In ASP.NET Core 3.0, we use endpoint routing, so the routing step is separated from the invocation of the endpoint. In practical terms that means we have two pieces of middleware:
  - EndpointRoutingMiddleware that does the actual routing i.e. calculating which endpoint will be invoked for a given request URL path.
  - EndpointMiddleware that invokes the endpoint.

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    // Add the EndpointRoutingMiddleware
    app.UseRouting();

    // All middleware from here onwards know which endpoint will be invoked
    app.UseCors();

    // Execute the endpoint selected by the routing middleware
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
    });
}
```

Map the VersionMiddleware to endpoint routing

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    app.UseRouting();

    app.UseCors();

    app.UseEndpoints(endpoints =>
    {
        // Add a new endpoint that uses the VersionMiddleware
        endpoints.Map("/version", endpoints.CreateApplicationBuilder()
            .UseMiddleware<VersionMiddleware>()
            .Build())
            .WithDisplayName("Version number");

        endpoints.MapDefaultControllerRoute();
    });
}
```

Map Version Extension Method

```
public static class VersionEndpointRouteBuilderExtensions
{
    public static IEndpointConventionBuilder MapVersion(this IEndpointRouteBuilder endpoints,
string pattern)
    {
        var pipeline = endpoints.CreateApplicationBuilder()
            .UseMiddleware<VersionMiddleware>()
            .Build();

        return endpoints.Map(pattern, pipeline).WithDisplayName("Version number");
    }
}
```

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    app.UseRouting();

    app.UseCors();

    // Execute the endpoint selected by the routing middleware
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapVersion("/version");
        endpoints.MapDefaultControllerRoute();
    });
}
```

Building Custom Middleware

- So, as you know from the definition of middleware, a RequestDelegate is a method that will handle an HTTP request. All the information about that HTTP request will be inside of the HttpContext passed as a parameter inside of RequestDelegate. Based on this HttpContext, the middleware will be able to inspect the request and produce a response.
- Let's call our middleware component SayHello, which we pass inside of Use() .

```
app.Use(SayHello);
```

```
private RequestDelegate SayHello(RequestDelegate arg)
{
    return async ctx =>
    {
        await ctx.Response.WriteAsync("Hello, World");
    };
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        // The default HSTS value is 30 days. You may want to change thi
        app.UseHsts();
    }
    app.Use(SayHello);

    app.UseHttpsRedirection();
    app.UseStaticFiles();

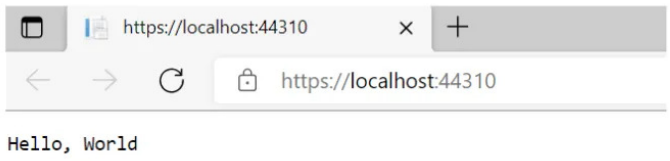
    app.UseRouting();

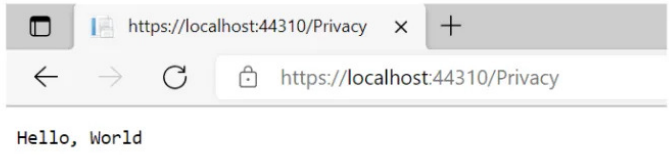
    app.UseAuthorization();

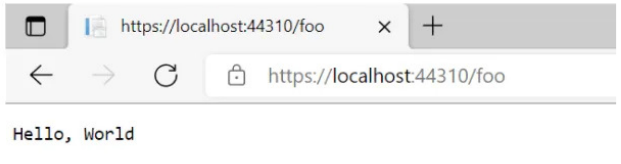
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    });
}
```

Place the app.use at the beginning.

DeveloperExceptionPage is the first component in the middleware pipeline, always pass the HTTP request onto the next component in the pipeline. The next component is our SayHello. From its definition, we know this component will always return a response – the message "Hello, World!". Once SayHello produces a response, it will be passed back to the previous component in the pipeline, DeveloperExceptionPage.







```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        // The default HSTS value is
        app.UseHsts();
    }

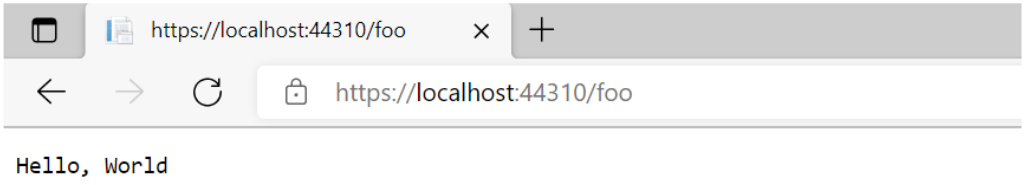

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    });
    app.Use(SayHello);
}
```

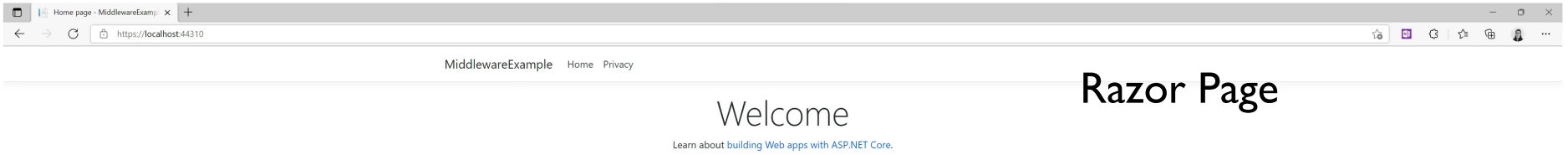
When the HTTP request is valid, meaning that the Endpoints middleware does have something to return, then our web app will work exactly like before we added the new piece of middleware. This is because the HTTP request never reaches the SayHello. If, however, the HTTP request is an invalid URL, we will then hit the end of the pipeline, as none of the earlier components will be able to return anything.



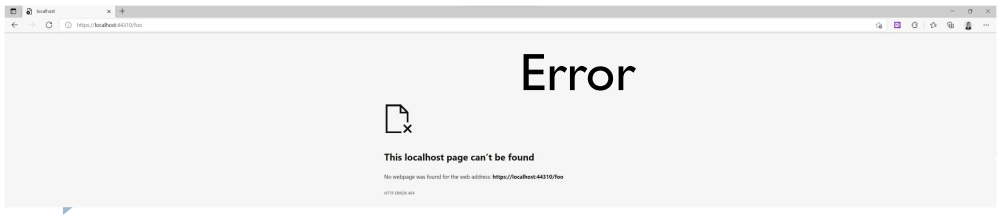
Place the app.use at the last.

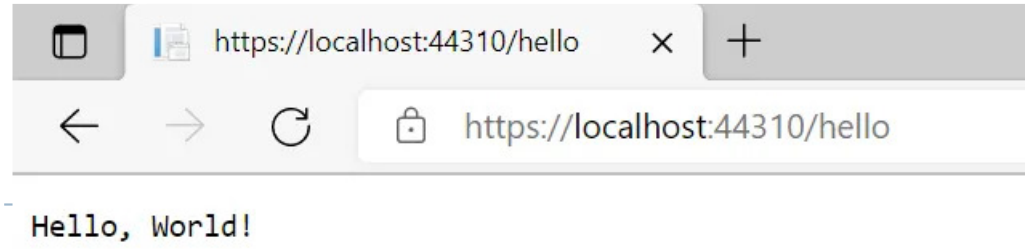
Activate SayHello only when the URL in the request is “/hello”.

```
private RequestDelegate SayHello(RequestDelegate next)
    {
        return async ctx =>
        {
            if (ctx.Request.Path.StartsWithSegments("/hello"))
            {
                await ctx.Response.WriteAsync("Hello, World!");
            }
            else
            {
                    await next(ctx);
            }
        };
    }
```



Razor Page





Using Endpoints

```
app.UseEndpoints(endpoints =>
{
    endpoints.Map("/hello", endpoints.CreateApplicationBuilder()
        .Use(async (context, next) =>
                    { await context.Response.WriteAsync("Hello, World!");
                        await next();
                    })

         .Build());
    endpoints.MapRazorPages();
});
```

MCSE 541:Web Computing and Mining

ASP.NET Core-Dependency Injection

Prof. Dr. Shamim Akhter
Stamford University Bangladesh

Dependency Injection

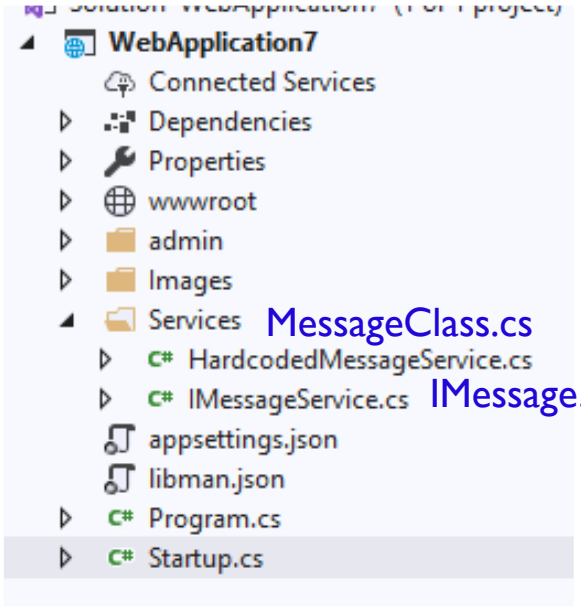
- Dependency injection (also known as DI) is a design pattern
  - in which an object does not create it dependent classes, but asks for it.
- ASP.NET Cores Dependency injection framework
  - Uses DI Container or IoC Container to implement that design pattern (create the object)

Adding an Interface and a New class into Services folder

- Add one interface @ Service folder

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication7.Services
{
    3 references
    public interface IMessageService
    {
        2 references | 0 exceptions
        string GetMessage();
    }
}
```

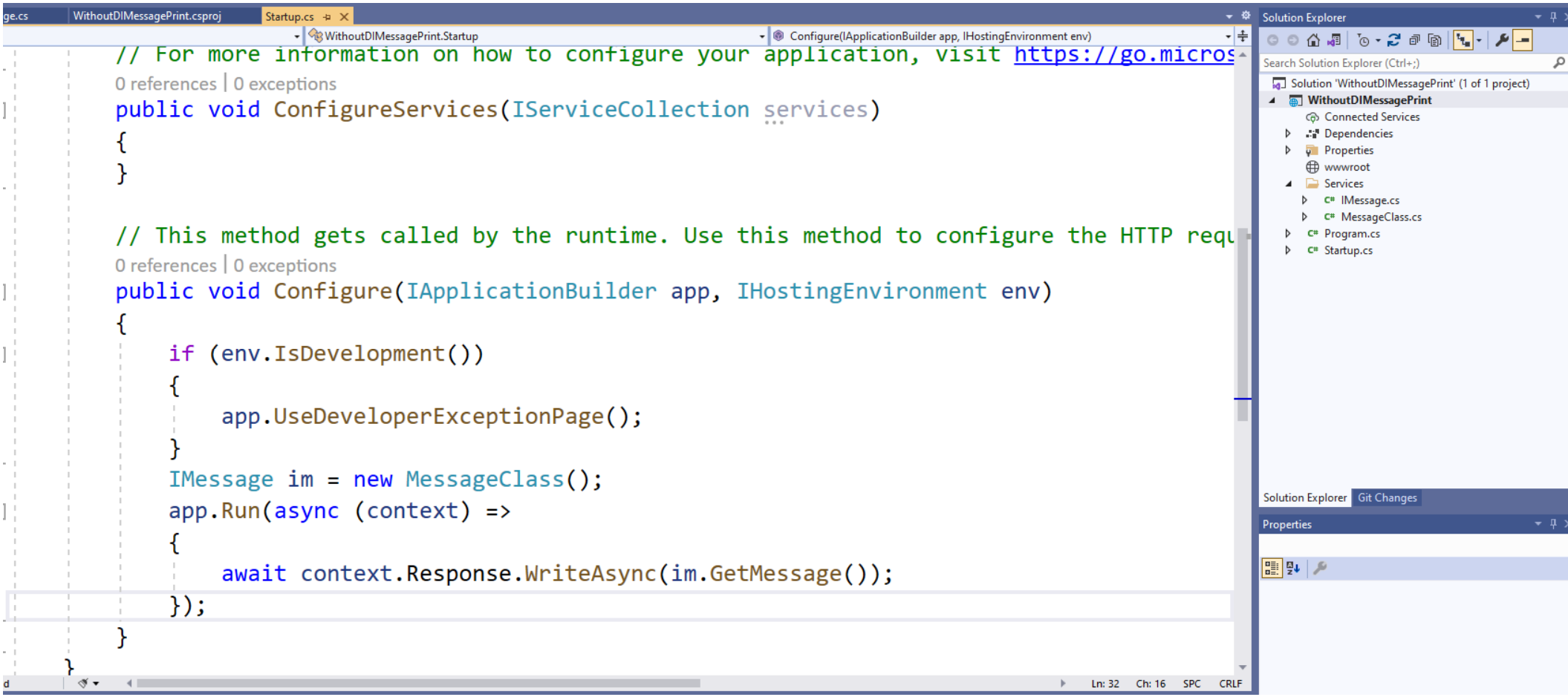


- Add one class @HardcodedMessageService

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication7.Services
{
    1 reference
    public class HardcodedMessageService : IMessageService
    {
        2 references | 0 exceptions
        public string GetMessage()
        {
            return "Hardcoed message from a service.";
        }
    }
}
```

Without Dependency Injection



Configure A Service (With Dependency Injection)

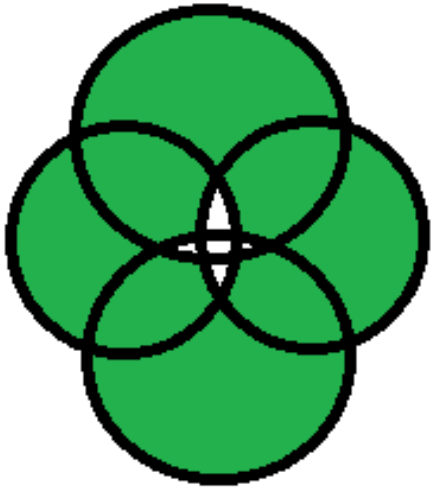
```
using WebApplication7.Services;


public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMessageService, HardcodedMessageService>();

}
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    IMessageService msg)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

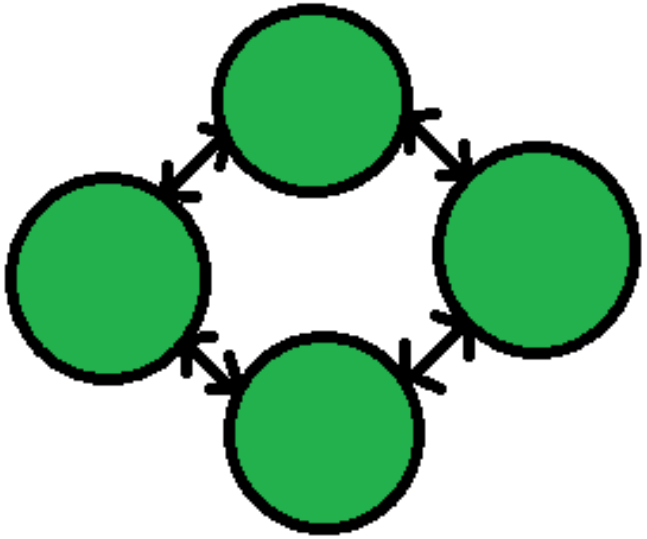
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(msg.GetMessage());
    });
}
```

Tightly Coupling vs Loose Coupling



Tight coupling:

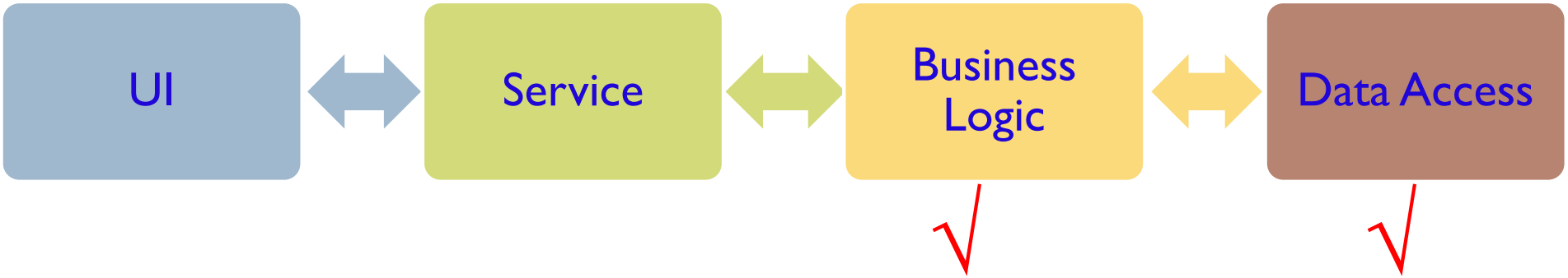
1. More Interdependency
2. More coordination
3. More information flow



Loose coupling:

1. Less Interdependency
2. Less coordination
3. Less information flow

N-Tier Architecture



In the typical n-tier architecture, the User Interface (UI) uses Service layer to retrieve or save data. The Service layer uses the `BusinessLogic` class to apply business rules on the data. The `BusinessLogic` class depends on the `DataAccess` class which retrieves or saves the data to the underlying database. This is simple n-tier architecture design. Let's focus on the `BusinessLogic` and `DataAccess` classes to understand IoC.

Tightly Couple Classes

```
public class CustomerBusinessLogic
{
    DataAccess _dataAccess;

    public CustomerBusinessLogic()
    {
        _dataAccess = new DataAccess();
    }

    public string GetCustomerName(int id)
    {
        return _dataAccess.GetCustomerName(id);
    }
}

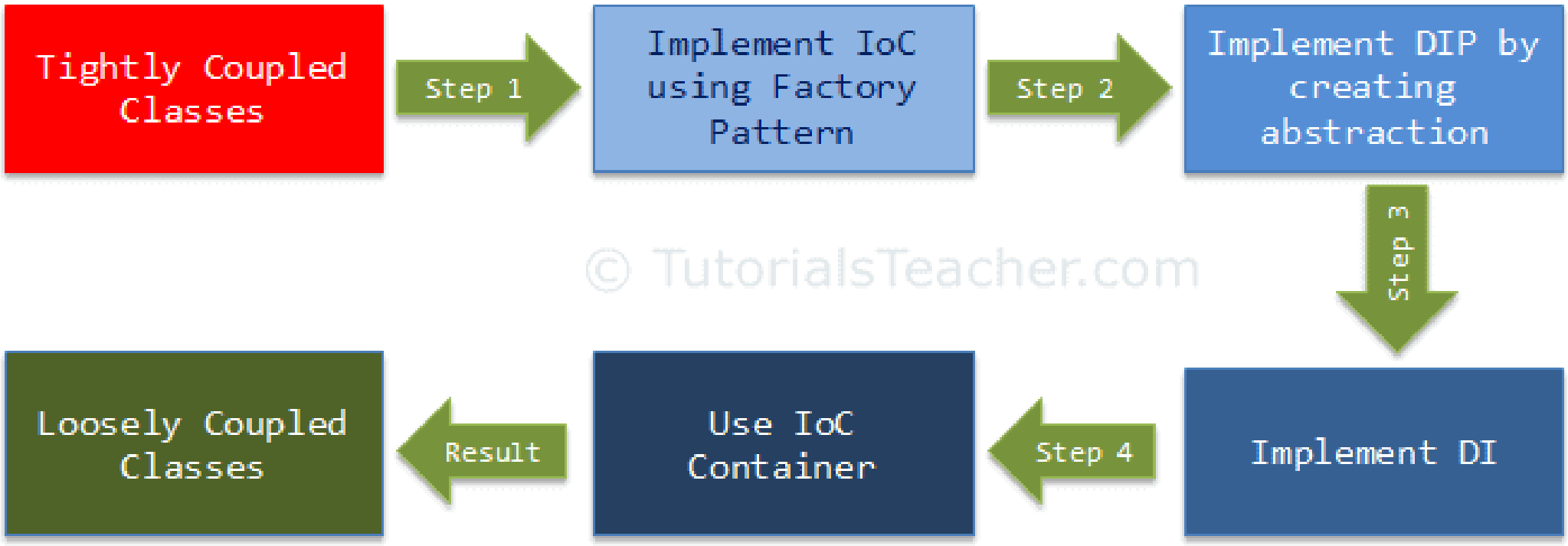
public class DataAccess
{
    public DataAccess()
    {
    }

    public string GetCustomerName(int id) {
        return "Dummy Customer Name"; // get it from DB in real app
    }
}
```

Factory Design Pattern

- According to Gang of Four
  - Factory Design Pattern states that **"A factory is an object which is used for creating other objects"**.
  - Factory Design pattern,
    - we create an object without exposing the creation logic to the client
    - and the client will refer to the newly created object using a common interface.

- The following figure illustrates how we are going to achieve loosely coupled design step by step.



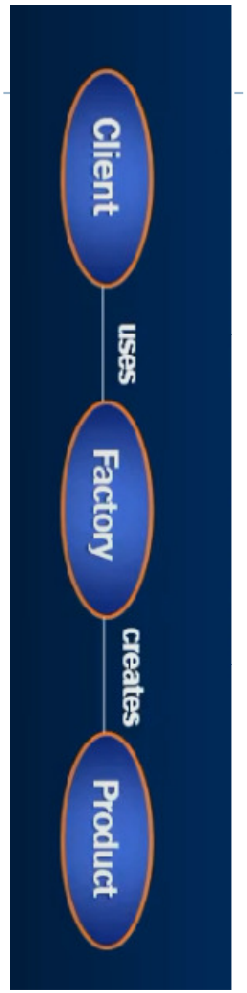
Implementing IoC using factory pattern

```
public class CustomerBusinessLogic
{
    public CustomerBusinessLogic()
    {
    }

    public string GetCustomerName(int id)
    {
        DataAccess _dataAccess = DataAccessFactory.GetDataAccessObj();

        return _dataAccess.GetCustomerName(id);
    }
}

public class DataAccessFactory
{
    public static DataAccess GetDataAccessObj()
    {
        return new DataAccess();
    }
}
```




```
public class DataAccess
{
    public DataAccess()
    {
    }

    public string GetCustomerName(int id) {
        return "Dummy Customer Name"; // get it from DB in real app
    }
}
```

Dependency Inversion Principle

1. High-level modules should not depend on low-level modules. Both should depend on the abstraction.
2. Abstractions should not depend on details. Details should depend on abstractions.

In English, abstraction means something which is non-concrete. In programming terms, the above `CustomerBusinessLogic` and `DataAccess` are concrete classes, meaning we can create objects of them. So, abstraction in programming means to create an interface or an abstract class which is non-concrete. This means we cannot create an object of an interface or an abstract class. As per DIP, `CustomerBusinessLogic` (high-level module) should not depend on the concrete `DataAccess` class (low-level module). Both classes should depend on abstractions, meaning both classes should depend on an interface or an abstract class.

DIP Continue

```
public interface ICustomerDataAccess
{
    string GetCustomerName(int id);
}

public class CustomerDataAccess: ICustomerDataAccess
{
    public CustomerDataAccess() {
    }

    public string GetCustomerName(int id) {
        return "Dummy Customer Name";
    }
}

public class DataAccessFactory
{
    public static ICustomerDataAccess GetCustomerDataAccessObj()
    {
        return new CustomerDataAccess();
    }
}
```

DIP Continue

```
public class CustomerBusinessLogic
{
    ICustomerDataAccess _custDataAccess;

    public CustomerBusinessLogic()
    {
        _custDataAccess = DataAccessFactory.GetCustomerDataAccessObj();
    }

    public string GetCustomerName(int id)
    {
        return _custDataAccess.GetCustomerName(id);
    }
}
```

Polymorphic Reference

Types of Dependency Injection

As you have seen above, the injector class injects the service (dependency) to the client (dependent). The injector class injects dependencies broadly in three ways: through a constructor, through a property, or through a method.

Constructor Injection: In the constructor injection, the injector supplies the service (dependency) through the client class constructor.

Property Injection: In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.

Method Injection: In this type of injection, the client class implements an interface which declares the method(s) to supply the dependency and the injector uses this interface to supply the dependency to the client class.

Example: Constructor Injection - C#

```
public class CustomerBusinessLogic
{
    ICustomerDataAccess _dataAccess;

    public CustomerBusinessLogic(ICustomerDataAccess custDataAccess)
    {
        _dataAccess = custDataAccess;
    }

    public CustomerBusinessLogic()
    {
        _dataAccess = new CustomerDataAccess();
    }

    public string ProcessCustomerData(int id)
    {
        return _dataAccess.GetCustomerName(id);
    }
}

public interface ICustomerDataAccess
{
    string GetCustomerName(int id);
}
```

```
public class CustomerDataAccess: ICustomerDataAccess
{
    public CustomerDataAccess()
    {
    }

    public string GetCustomerName(int id)
    {
        //get the customer name from the db in real application
        return "Dummy Customer Name";
    }
}
```

Example: Inject Dependency - C#

```
public class CustomerService
{
    CustomerBusinessLogic _customerBL;

    public CustomerService()
    {
        _customerBL = new CustomerBusinessLogic(new CustomerDataAccess());
    }

    public string GetCustomerName(int id) {
        return _customerBL.ProcessCustomerData(id);
    }
}
```

Injects the objects to another

As you can see in the above example, the `CustomerService` class creates and injects the `CustomerDataAccess` object into the `CustomerBusinessLogic` class. Thus, the `CustomerBusinessLogic` class doesn't need to create an object of `CustomerDataAccess` using the `new` keyword or using factory class. The calling class (CustomerService) creates and sets the appropriate DataAccess class to the `CustomerBusinessLogic` class. In this way, the `CustomerBusinessLogic` and `CustomerDataAccess` classes become "more" loosely coupled classes.

DI Container or IoC container

- IoC Container is a framework for implementing automatic dependency injection.
- It manages object creation and it's life-time, and also injects dependencies to the class.

All the containers must provide easy support for the following DI lifecycle.

- **Register:** The container must know which dependency to instantiate when it encounters a particular type. This process is called registration. Basically, it must include some way to register type-mapping.
- **Resolve:** When using the IoC container, we don't need to create objects manually. The container does it for us. This is called resolution. The container must include some methods to resolve the specified type; the container creates an object of the specified type, injects the required dependencies if any and returns the object.
- **Dispose:** The container must manage the lifetime of the dependent objects. Most IoC containers include different lifetimemanagers to manage an object's lifecycle and dispose it.

Configure A Service (With Dependency Injection)

```
using WebApplication7.Services;

public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IMessageService, HardcodedMessageService>();
}
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    IMessageService msg)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(msg.GetMessage());
    });
}
```

Register

Resolution

Managing the Service Lifetime

- DI Container must need to decide the life time of the services.
- Service lifetime will be decided during the service registration process.

Most IOC containers allow a “lifetime” to be applied when wiring up dependencies. With the ASP.NET Core Dependency Injection framework, the following life cycles are available.

•Transient – Each time a transient object is requested, a new instance will be created
•Scoped – The same object will be used when requested within the same request
•Singleton – The same object will always be used across all requests

Service Types

Singleton

- The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance.
- services.AddSingleton<IMySingletonService, MySingletonService>();
- Now, each time an instance of MySingletonService Class is asked for, it will return the same instance every time for the lifetime of your application.

Scoped

- services are created per scope/request.
- In a web application, every web request creates a new separated service scope.
- That means scoped services are generally created per web request.

Transient

- Services are created every time they are injected or requested.
- Lightweight and stateless services.

Service Interfaces and Class

```
public interface ITransientService
{
    Guid GetID();
}

public interface IScopedService
{
    Guid GetID();
}

public interface ISingletonService
{
    Guid GetID();
}
```

```
public class SomeService : ITransientService, IScopedService, ISingletonService
{
    Guid id;
    public SomeService()
    {
        id = Guid.NewGuid();
    }

    public Guid GetID()
    {
        return id;
    }
}
```

- A GUID (Global Unique Identifier) is **a 128-bit integer used as a unique identifier.**
- System.Guid id.
- Guid.NewGuid() makes an actual guid with a unique value,
- GUIDs are most commonly written in text as a sequence of hexadecimal digits as such, 3F2504E0-4F89-11D3-9A0C-0305E82C3301

Register the Services

```
services.AddTransient<ITransientService, SomeService>();
```

Inject it into Controller

```
public class HomeController : Controller
{
    ITransientService _transientService1;
    ITransientService _transientService2;

    public HomeController(ITransientService transientService1,
                          ITransientService transientService2)
    {
        _transientService1 = transientService1;
        _transientService2 = transientService2;
    }

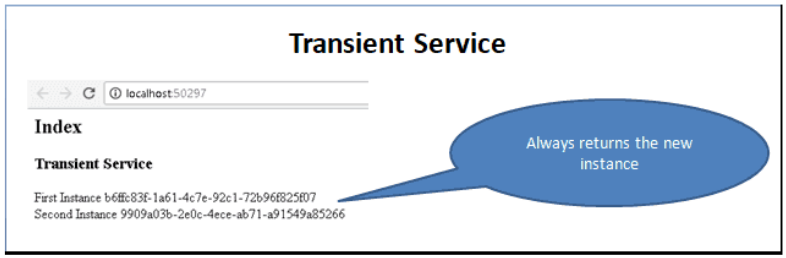
    public IActionResult Index()
    {

        ViewBag.message1 ="First Instance " + _transientService1.GetID().ToString();
        ViewBag.message2 ="Second Instance "+ _transientService2.GetID().ToString();

        return View();
    }
}
```

View

```
<h3>Transient Service</h3>
@ViewBag.message1
</br>
@ViewBag.message2
```



Other Services

```
services.AddScoped<IScopedService, SomeService>();
```

```
services.AddSingleton<ISingletonService, SomeService>();
```

```
public class HomeController : Controller
{
    ITransientService _transientService1;
    ITransientService _transientService2;

    IScopedService _scopedService1;
    IScopedService _scopedService2;

    ISingletonService _singletonService1;
    ISingletonService _singletonService2;

    public HomeController(ITransientService transientService1,
                          ITransientService transientService2,
                          IScopedService scopedService1,
                          IScopedService scopedService2,
                          ISingletonService singletonService1,
                          ISingletonService singletonService2)
    {

        _transientService1 = transientService1;
        _transientService2 = transientService2;

        _scopedService1 = scopedService1;
        _scopedService2 = scopedService2;

        _singletonService1 = singletonService1;
        _singletonService2 = singletonService2;

     }
```

```
    public IActionResult Index()
    {

        ViewBag.message1 ="First Instance " + _transientService1.GetID().ToString();
        ViewBag.message2 ="Second Instance "+ _transientService2.GetID().ToString();


        ViewBag.message3 = "First Instance " + _scopedService1.GetID().ToString();
        ViewBag.message4 = "Second Instance " + _scopedService2.GetID().ToString();

        ViewBag.message5 = "First Instance " + _singletonService1.GetID().ToString()
        ViewBag.message6 = "Second Instance " + _singletonService2.GetID().ToString(

        return View();
    }
}
```

View

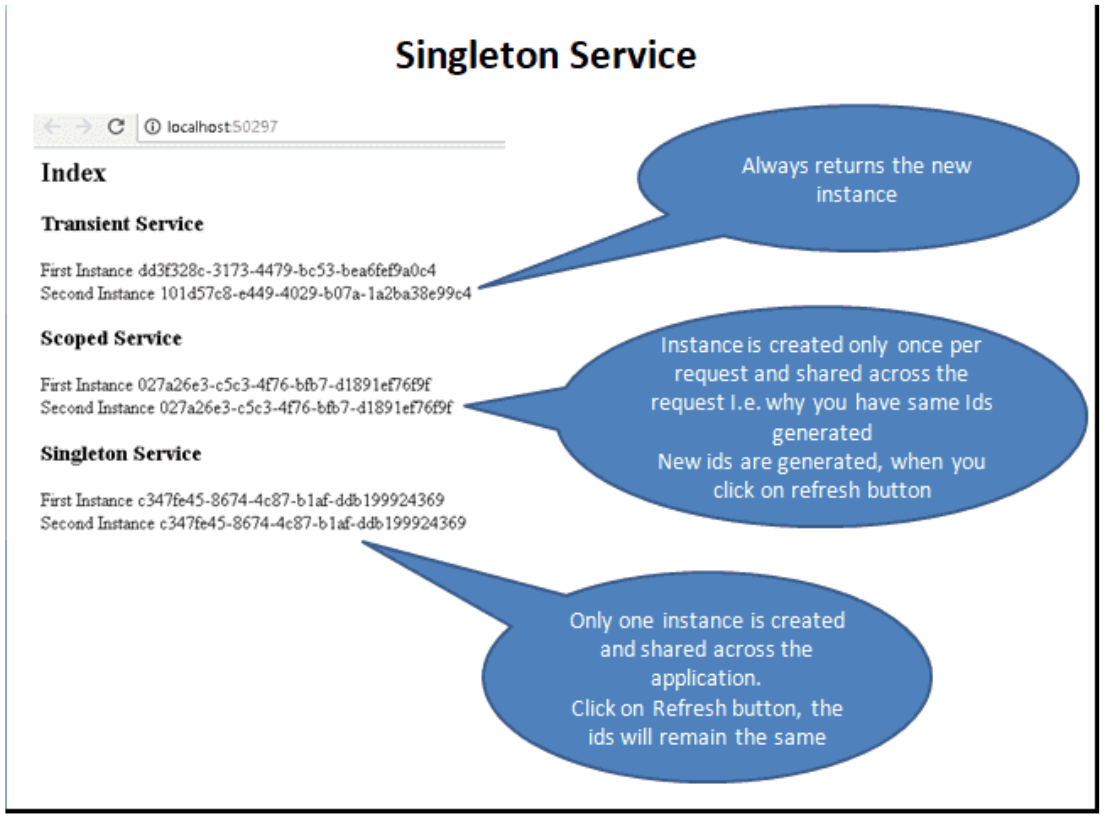
```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<h3>Transient Service</h3>
@ViewBag.message1
</br>
@ViewBag.message2

<h3>Scoped Service</h3>
@ViewBag.message3
</br>
@ViewBag.message4

<h3>Singleton Service</h3>
@ViewBag.message5
</br>
@ViewBag.message6
```



MCSE 541:Web Computing and Data Mining

ASP.NET Core-MVC

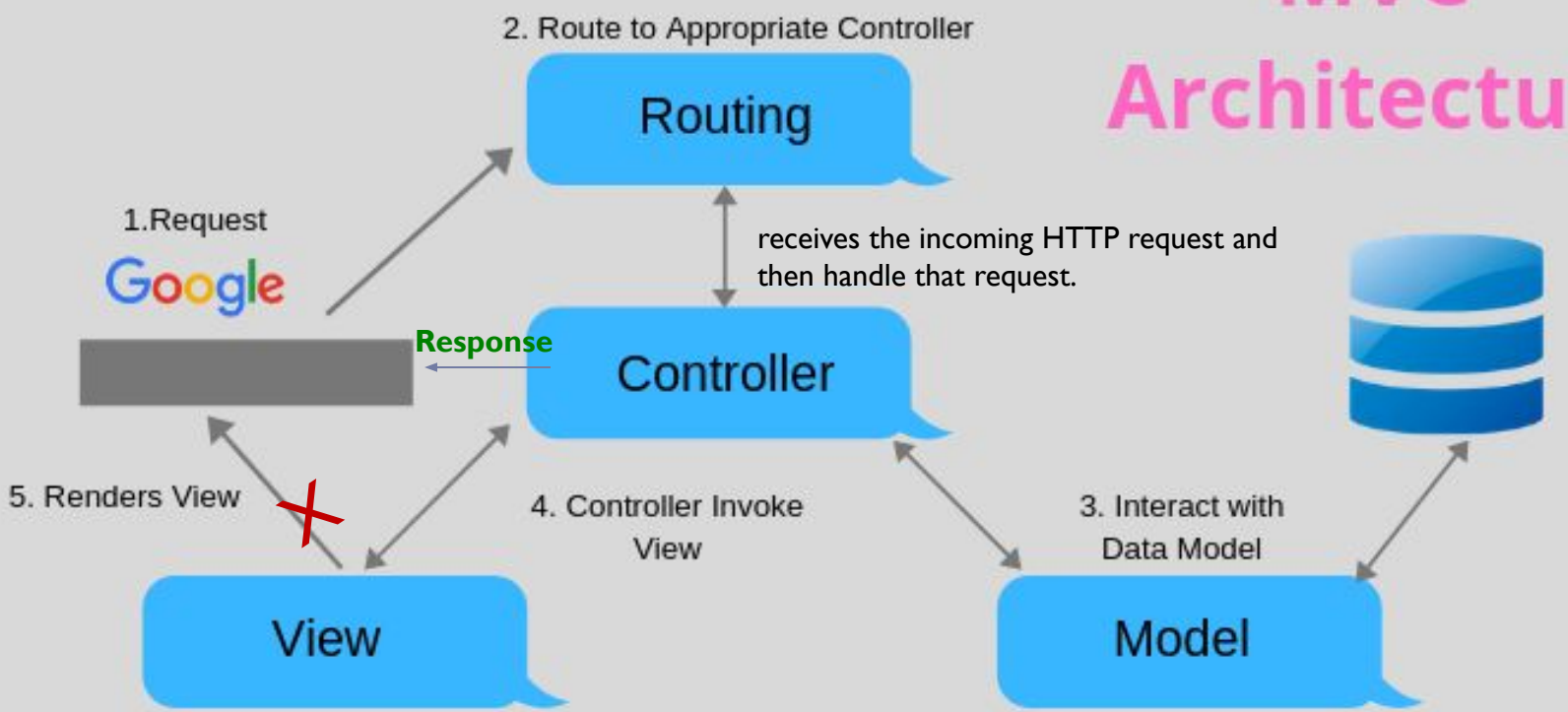
Prof. Dr. Shamim Akhter

MVC (Model-View-Controller) Architecture

- A concept in software programming
- Divides the program into three parts:
  - Model, View and Controller.
- MVC becomes popular
  - to design web and mobile application
  - famous Frameworks uses MVC like Ruby on rails, Js, Django (Python Framework), Symphony (PHP Framework), CakePHP (PHP), Laravel (PHP framework), CherryPy (Python Framework), react, Angular, and etc.

ASP.NET MVC is a **Framework** whereas **MVC** is a **Design Pattern**.

MVC Architecture



Create HTML (UI) to render the domain data or business data passed by Controller.

Contains a set of classes to represent the domain or business data as well as logic to manage the data.

Example: Student Details Web Page

- Request url:"**http://dotnettutorials.net/student/details/2**"

```
public class StudentController : Controller
{
    public ActionResult Details(int studentId)
    {
        StudentBusinessLayer studentBL = new StudentBusinessLayer();
        Student studentDetail = studentBL.GetById(studentId);
        return View(studentDetail);
    }
}
```

Student ID: 2
Name: James
Gender: Male
Branch: CSE
Section: A2

Model

```
public class Student
{
    public int StudentID { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }
    public string Branch { get; set; }
    public string Section { get; set; }
}
```

Model to represent the student data

```
public class StudentBusinessLayer
{
    public Student GetById(int StudentID)
    {
        Student student = new Student()
        {
            StudentID = StudentID,
            Name = "James",
            Gender = "Male",
            Branch = "CSE",
            Section = "A2",
        };
        return student;
    }
}
```

Model to manage the Student data

View

```
@model FirstMVCApplication.Models.Student
<html>
<head>
    <title>Student Details</title>
</head>
<body>
    <br />
    <br />
    <table>
        <tr>
            <td>Student ID: </td>
            <td>@Model.StudentID</td>
        </tr>
        <tr>
            <td>Name: </td>
            <td>@Model.Name</td>
        </tr>
        <tr>
            <td>Gender: </td>
            <td>@Model.Gender </td>
        </tr>
        <tr>
            <td>Branch: </td>
            <td>@Model.Branch</td>
        </tr>
        <tr>
            <td>Section: </td>
            <td>@Model.Section </td>
        </tr>
    </table>
</body>
</html>
```

Each developer can work on different parts of the application. For example, one developer may work on the view while the second developer can work on the controller logic and the third developer may work on the business logic.

Look Again

- "**http://dotnettutorials.net/student/details/2**"

```
public class StudentController : Controller
{
    public ActionResult Details(int studentId)
    {
        StudentBusinessLayer studentBL = new StudentBusinessLayer();
        Student studentDetail = studentBL.GetById(studentId);
        return View(studentDetail);
    }
}
```

Something is missing??

Where this mapping is defined?

- Routing-a middleware must be able to route incoming HTTP requests to a controller

Mapping is defined within the RegisterRoutes() of the RouteConfig class at RouteConfig.cs class within the App_Start Folder.

ASP.Net 4.5 MVC Template

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new {
                controller = "Home",
                action = "Index",
                id = UrlParameter.Optional
            }
        );
    }
}
```

Default Route
URL Pattern
Default Controller
Default Action Method

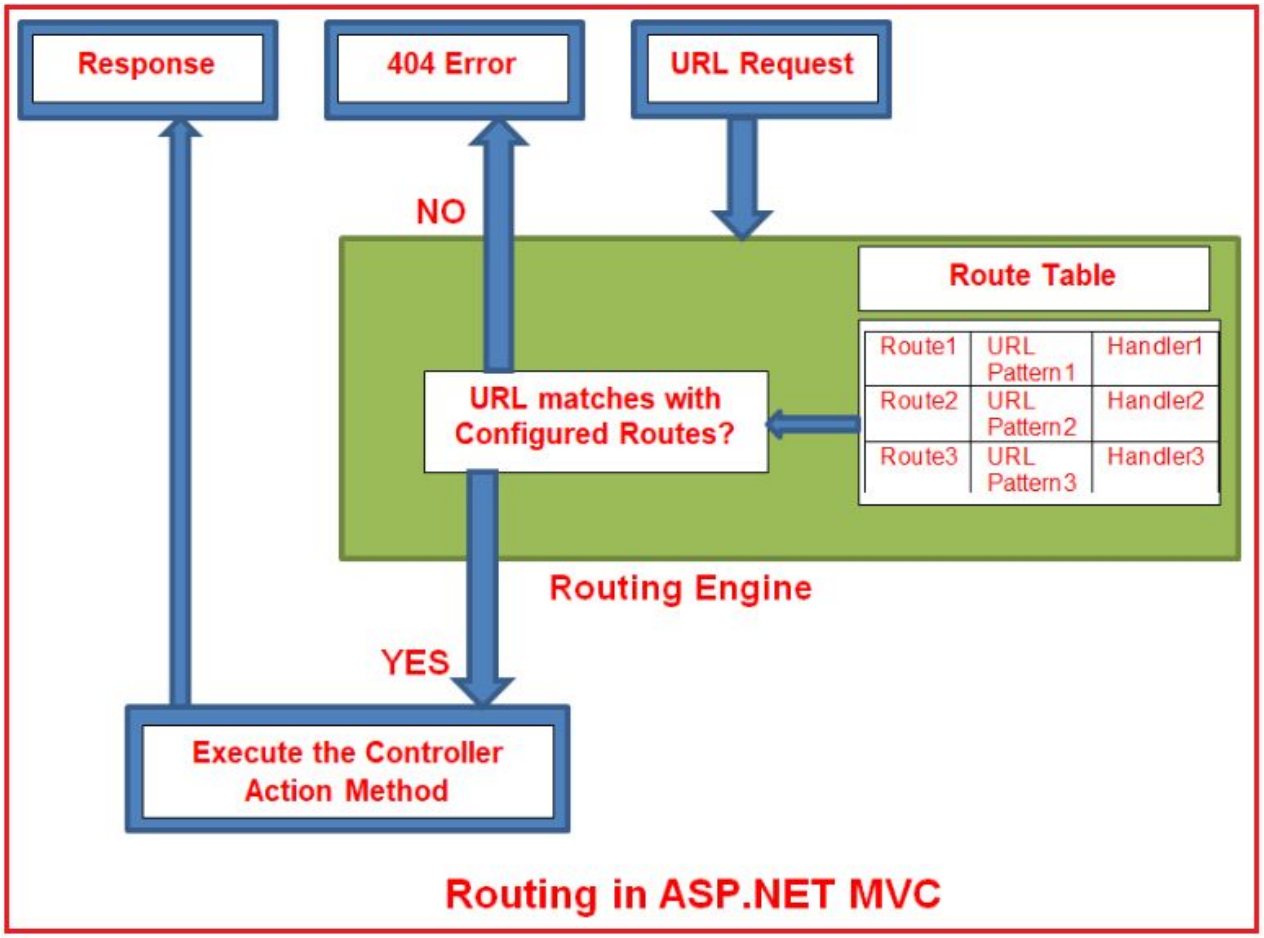
RouteConfig class

```
namespace FirstMVCDemo
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Employee",
                url: " Employee/{id}",
                defaults: new { controller = "Employee", action = "Index" }
            );

            routes.MapRoute(
                name: "Default", //Route Name
                url: "{controller}/{action}/{id}", //Route Pattern
                defaults: new
                {
                    controller = "Home", //Controller Name
                    action = "Index", //Action method Name
                    id = UrlParameter.Optional //Defaut value for above defined
parameter
                }
            );
        }
    }
}
```

Routing in ASP.NET MVC



Employee example from book.

Routing define in Startup class

/Home/Index/123

ASP.Net Core 2.1 MVC

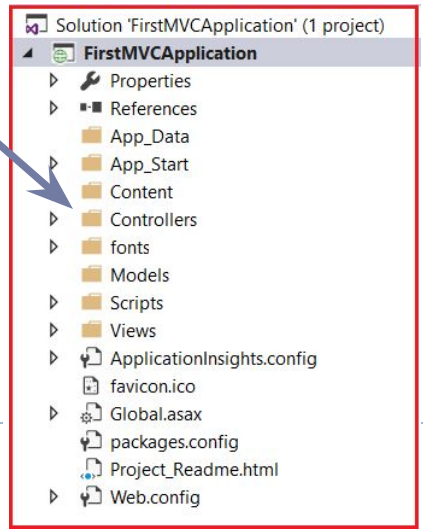
```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Or with an explicit method called from the **UseMvc** method inside the **Configure** method:

```
app.UseMvc(ConfigureRoutes);
```

```
private void ConfigureRoutes(IRouteBuilder routeBuildrer)
{
    routeBuildrer.MapRoute("Default",
        "{controller=Home}/{action=Index}/{Id?}");
}
```

HomeController class, Index method, Id parameter



- **http://localhost:53605/ =>**
  controller = **Home**, action = **Index**, id = **none**, since default value of controller and action are Home and Index respectively.
- **http://localhost:53605/Home =>**
  controller = **Home**, action = **Index**, id = **none**, since default value of action is Index
- **http://localhost:53605/Home/Index =>**
  controller = **Home**, action = **Index**, id=**none**
- **http://localhost:53605/Home/Index/5 =>**
  controller = **Home**, action = **Index**, id = **5**

/Employee/Name/
/Employee/Index
/Employee

```
public class EmployeeController
{
    public string Name()
    {
        return "Jonas";
    }

    public string Country()
    {
        return "Sweden";
    }

    public string Index()
    {
        return "Hello from Employee";
    }
}
```

Types of Routing

- Convention-Based Routing
- Attribute Routing
  - Assign attributes to the controller class and its action methods. The metadata in those attributes tell ASP.NET when to call a specific controller and action.

```
[Route("employee")]
public class EmployeeController
```

Ambiguous Action Exception

To solve this, you can specify the **Route** attribute for each of the action methods, and use an empty string for the default action. Let's make the **Index** action the default action, and name the routes for the other action methods the same as the methods.

```
[Route("")]
public string Index()
{
    return "Hello from Employee";
}

[Route("name")]
public string Name()
{
    return "Jonas";
}

[Route("country")]
public string Country()
{
    return "Sweden";
}
```

Attribute Routing

Let's clean up the controller and make its route more reusable. Instead of using a hardcoded value for the controller's route, you can use the **[controller]** token that represents the name of the controller class (*Employee* in this case). This makes it easier if you need to rename the controller for some reason.

```
[Route("[controller]")]
public class EmployeeController
```

You can do the same for the action methods, but use the **[action]** token instead. ASP.NET will then replace the token with the action's name.

```
[Route("[action]")]
public string Name()
{
    return "Jonas";
}
```

```
[Route("")]
[Route("[action]")]
public string Index()
{
    return "Hello from Employee";
}
```

Attribute Routing

```
[Route("company/[controller]/[action]")]
public class EmployeeController
{
    public string Name()
    {
        return "Jonas";
    }

    public string Country()
    {
        return "Sweden";
    }

    public string Index()
    {
        return "Hello from Employee";
    }
}
```

No need to place on top of every action

IActionResult

The controller actions that you have seen so far have all returned strings. When working with actions, you rarely return strings. Most of the time you use the **IActionResult** return type, which can return many types of data, such as objects and views. To gain access to **IActionResult** or derivations thereof, the controller class must inherit the **Controller** class.

There are more specific implementations of that interface, for instance the **ContentResult** class, which can be used to return simple content such as strings. Using a more specific return type can be beneficial when unit testing, because you get a specific data type to test against.

Another return type is **ObjectType**, which often is used in Web API applications because it turns the result into an object that can be sent over HTTP. JSON is the default return type, making the result easy to use from JavaScript on the client. The data carrier can be configured to deliver the data in other formats, such as XML.

Implementing ContentResult

Let's change the **Name** action to return a **ContentResult**.

1. Open the **EmployeeController** class.
2. Have the **EmployeeController** class inherit the **Controller** class.

```
public class EmployeeController : Controller
```

3. Change the **Name** action's return type to **ContentResult**.

```
public ContentResult Name()
```

4. Change the **return** statement to return a content object by calling the **Content** method, and pass in the string to it.

```
public ContentResult Name()
{
    return Content("Jonas");
}
```

5. Save all files, open the browser, and navigate to the *Company/Employees/Name* URL.
6. Your name should be returned to the browser, same as before.

ObjectResult

```
public class Video
{
    public int Id { get; set; }
    public string Title { get; set; }
}
```

Video Model Class

```
public class HomeController : Controller
{
    public ObjectResult Index()
    {
        var model = new Video { Id = 1, Title = "Shreck" };
        return new ObjectResult(model);
    }
}
```

Controller Class

Introduction to Views

The most popular way to render a view from a ASP.NET Core MVC application is to use the Razor view engine. To render the view, a **ViewResult** is returned from the controller action using the **View** method. It carries with it the name of the view in the filesystem, and a model object if needed.

```
public ViewResult Index()
{
    var model = new List<Video>
    {
        new Video { Id = 1, Title = "Shreck" },
        new Video { Id = 2, Title = "Despicable Me" },
        new Video { Id = 3, Title = "Megamind" }
    };
    return View(model);
}
```

```
@model IEnumerable<AspNetCoreVideo.Models.Video>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Video</title>
</head>
<body>
    <table>
        @foreach (var video in Model)
        {
        <tr>
            <td>@video.Id</td>
            <td>@video.Title</td>
        </tr>
        }
    </table>
</body>
</html>
```

MCSE 541: Web Computing and Mining

ASP.NET Core-Razor Views

Prof. Dr. Shamim Akhter

Razor

- A view engine in ASP.NET MVC
- Allows to write mix of HTML and server side code using C#
- C# syntax has .cshtml file extension
- Razor syntax has following Characteristics:
  - Compact: enables to minimize number of characters and keystrokes required to write a code.
  - Easy to Learn: familiar language C# or Visual Basic.
  - Intelligence: supports statement completion within Visual Studio.
- Supports inline or multi-statement code block
  - <h2>@DateTime.Now.ToShortDateString()</h2>
  - Output: 08-09-2014

```
@{
      var date = DateTime.Now.ToShortDateString();
      var message = "Hello World";
    }
<h2>Today's date is: @date </h2>
<h3>@message</h3>
```

Use @: or <text>/<text> to display texts within code block.

The C# Script Compilation

- Modern Browsers have Javascript Interpreters(Javascript engine).
  - So when the browser come across javascript code,each line is executed by the Javascript engine.
- But in case of Java or C++ or C# you convert the code into an intermediate form.
- Normally in ASP.NET MVC, the views are not compiled until they are requested by the browser.
  - To avoid this you can precompile the views. Precompile also helps to identify any errors upfront than at runtime.
- We need to Add the following lines in CSProj file:

```
"tools": { "Microsoft.AspNetCore.Mvc.Razor.ViewCompilation.Tools": {
"version": "1.1.0-preview4-final" }
```

if-else condition

```
@ if(DateTime.IsLeapYear(DateTime.Now.Year) )
   {
      @DateTime.Now.Year @:is a leap year.
   }
   else
   {
       @DateTime.Now.Year @:is not a leap year.
   }
```

@ is razor syntax to access the server side variable.

for loop

```
@for (int i = 0; i < 5; i++)
{
    @i.ToString() <br />
}
```

Model

Use @model to use model object anywhere in the view.

```
@model Student <h2>Student Detail:</h2>
  <ul>
      <li>Student Id: @Model.StudentId</li>
      <li>Student Name: @Model.StudentName</li>
      <li>Age: @Model.Age</li>
 </ul>
```

HTML Helpers

- HtmlHelper class generates html elements using the model class object in razor view.
- It binds the model object to html elements to display value of model properties into html elements and also assigns the value of the html elements to the model properties while submitting web form.

```
@model IEnumerable<MVC_BasicTutorials.Models.Student>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.StudentName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Age)
        </th>
        <th></th>
    </tr>
```

@Html.ActionLink("Create New", "Create") would generate anchor tag
Create New.

Extension method for HtmlHelper

HtmlHelper

<a asp-controller="Speaker" asp-action="Evaluations">Speaker Evaluations

Speaker Evaluations

<html> <body> <a asp-controller="Speaker" asp-action="Detail" asp-route-id="@Model.SpeakerId">SpeakerId: @Model.SpeakerId </body> </html>

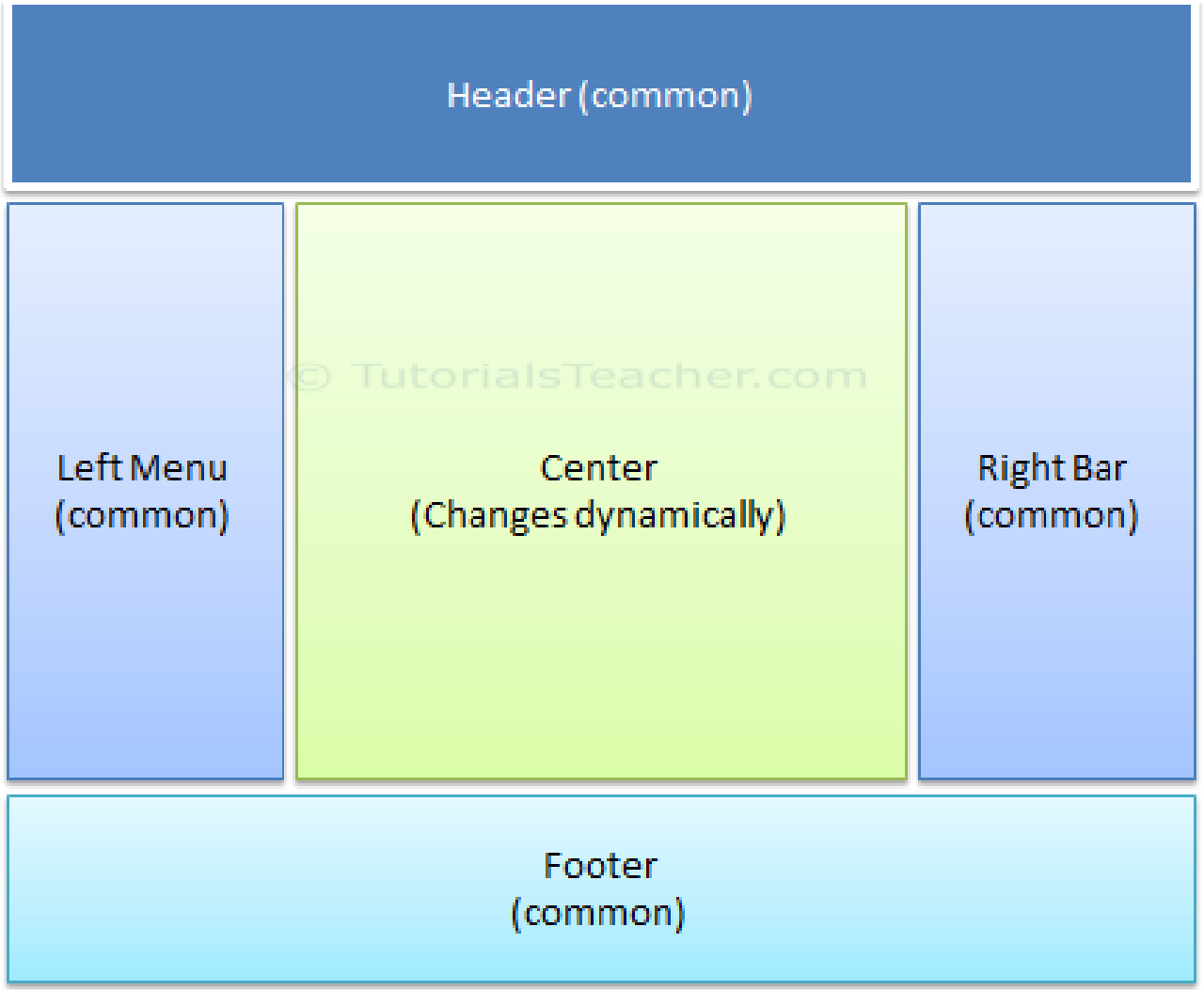
SpeakerId: 12

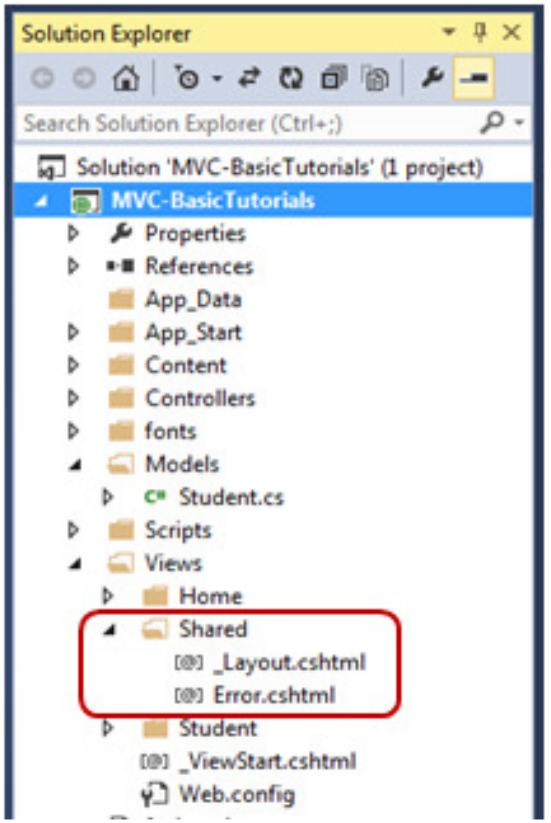
HTML Helper methods

HtmlHelper	Strongly Typed HtmlHelpers	Html Control
Html.ActionLink		Anchor link
Html.TextBox	Html.TextBoxFor	Textbox
Html.TextArea	Html.TextAreaFor	TextArea
Html.CheckBox	Html.CheckBoxFor	Checkbox
Html.RadioButton	Html.RadioButtonFor	Radio button
Html.DropDownList	Html.DropDownListFor	Dropdown, combobox
Html.ListBox	Html.ListBoxFor	multi-select list box
Html.Hidden	Html.HiddenFor	Hidden field
Password	Html.PasswordFor	Password textbox
Html.Display	Html.DisplayFor	Html text
Html.Label	Html.LabelFor	Label
Html.Editor	Html.EditorFor	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type.

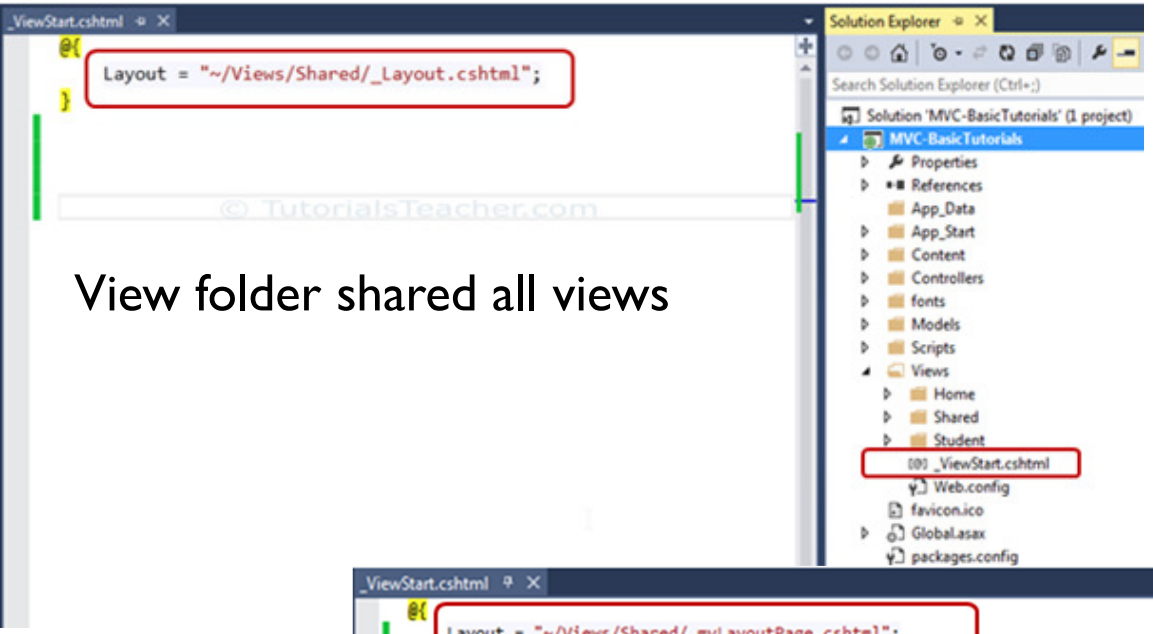
Layout View

- Common parts in the UI such as the logo, header, left navigation bar, right bar or footer section.

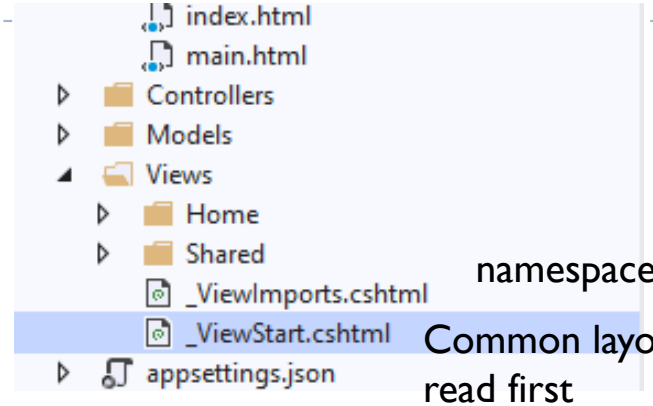




Use Layout View

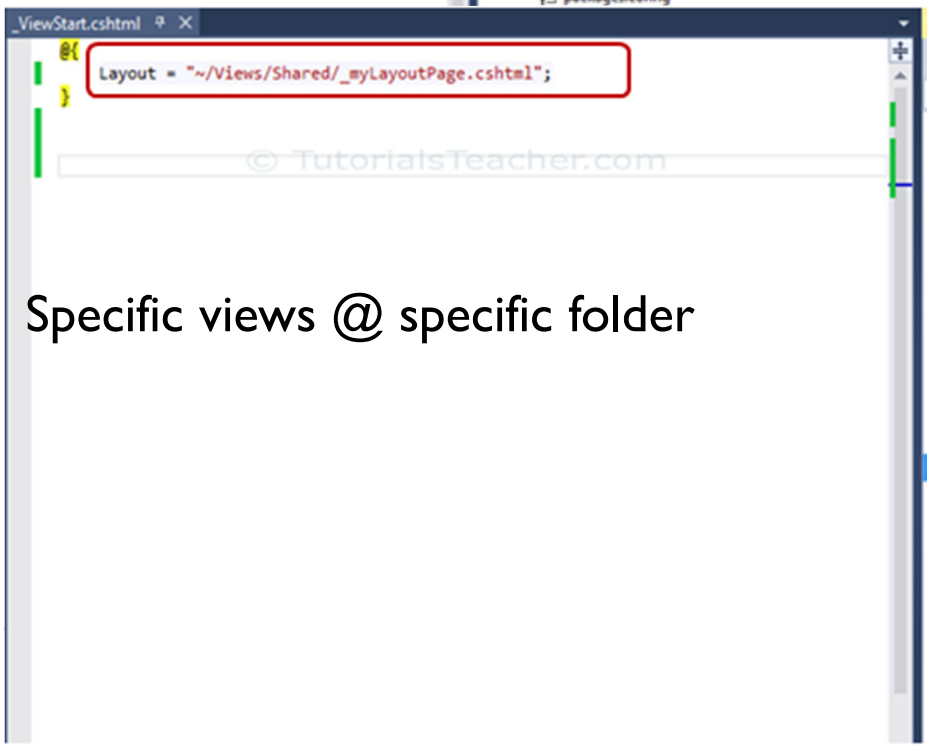


View folder shared all views

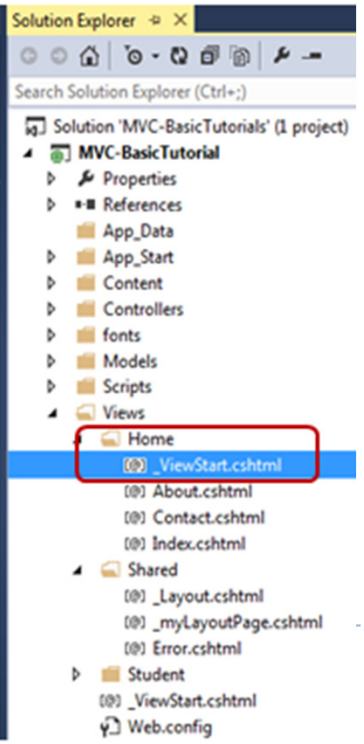


namespaces

Common layout read first



Specific views @ specific folder



Override default layout page

```
@{
    ViewBag.Title = "Home Page";
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";
}

<div class="jumbotron">
    <h1>ASP.NET</h1>
    <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications
    <p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
</div>

<div class="row">
    <div class="col-md-4">
        <h2>Getting started</h2>
        <p>
            ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that
            enables a clean separation of concerns and gives you full control over markup
            for enjoyable, agile development.
```

Specify Layout Page in ActionResult Method

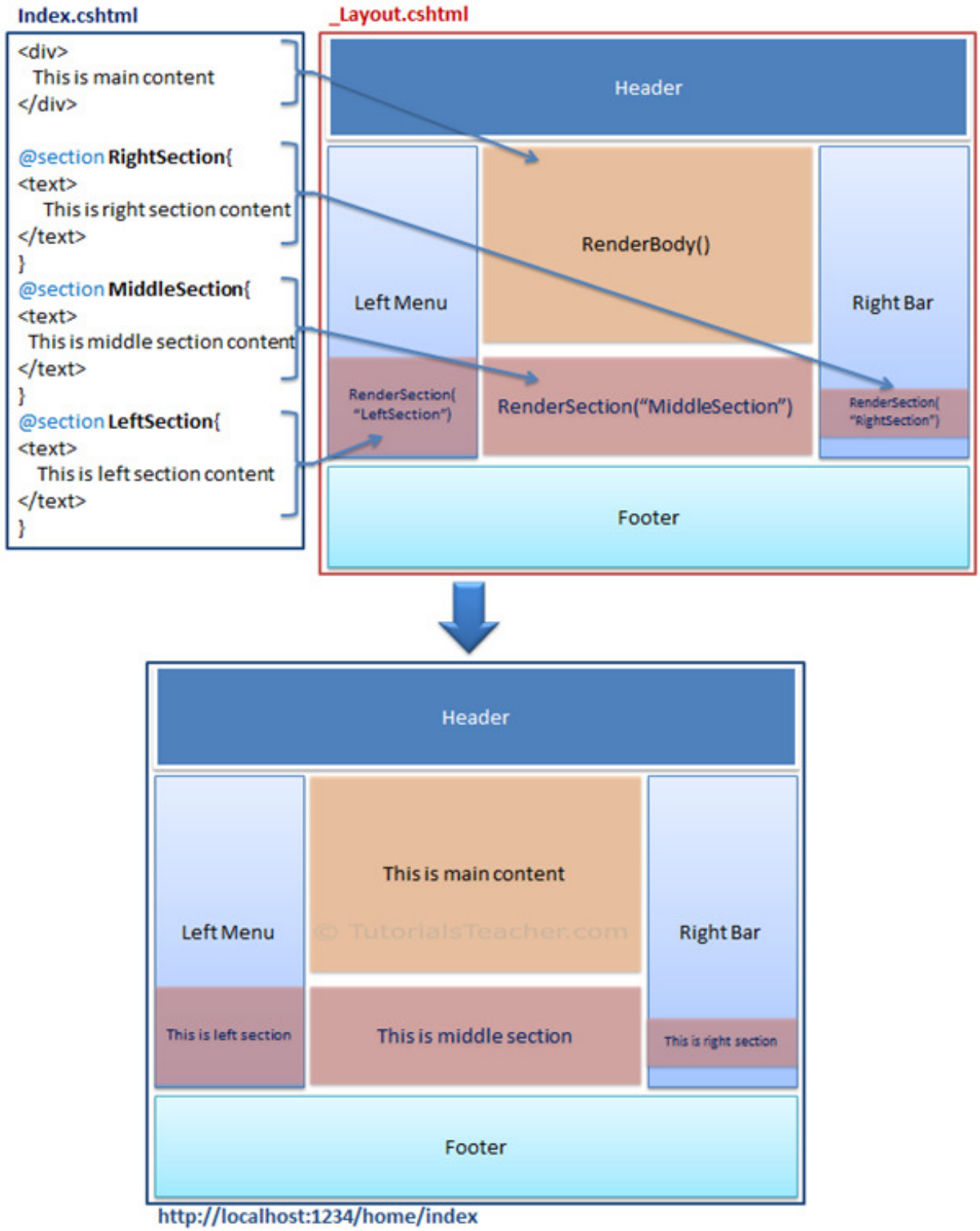
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("Index", "_myLayoutPage");
    }

    public ActionResult About()
    {
        return View();
    }

    public ActionResult Contact()
    {
        return View();
    }
}
```

Rendering Methods

Method	Description
RenderBody()	Renders the portion of the child view that is not within a named section. Layout view must include RenderBody() method.
RenderSection(string name)	Renders a content of named section and specifies whether the section is required. RenderSection() is optional in Layout view.


Index.cshtml

This is main content

@section RightSection{
<text>
This is right section content
</text>
}
@section MiddleSection{
<text>
This is middle section content
</text>
}
@section LeftSection{
<text>
This is left section content
</text>
}
_Layout.cshtml
Header
Left Menu
RenderBody()
Right Bar
RenderSection("LeftSection")
RenderSection("MiddleSection")
RenderSection("RightSection")
Footer
Header
Left Menu
This is main content
© TutorialsTeacher.com
Right Bar
This is left section
This is middle section
This is right section
Footer
http://localhost:1234/home/index

ASP.NET MVC - ViewBag

- Transfers temporary data (which is not included in model) from the controller to the view.

@ is razor syntax to access the server side variable.

Server Code

```
namespace MVC_BasicTutorials.Controllers
{
    public class StudentController : Controller
    {
        IList<Student> studentList = new List<Student>() {
                    new Student(){ StudentID=1, StudentName="Steve", Age = 21 },
                    new Student(){ StudentID=2, StudentName="Bill", Age = 25 },
                    new Student(){ StudentID=3, StudentName="Ram", Age = 20 },
                    new Student(){ StudentID=4, StudentName="Ron", Age = 31 },
                    new Student(){ StudentID=5, StudentName="Rob", Age = 19 }
                };
        // GET: Student
        public ActionResult Index()
        {
            ViewBag.TotalStudents = studentList.Count();

            return View();
        }

    }
}
```

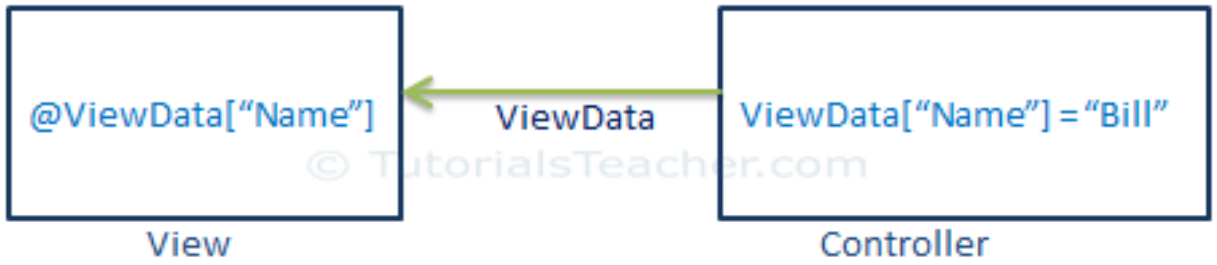
Script Code

```
<label>Total Students:</label>  @ViewBag.TotalStudents
```

```
Total Students: 5
```

ASP.NET MVC - ViewData

- ViewData is similar to ViewBag. It is useful in transferring data from Controller to View.
- ViewData is a dictionary which can contain key-value pairs where each key must be string.



Server Code

```
public ActionResult Index()
{
    IList<Student> studentList = new List<Student>();
    studentList.Add(new Student(){ StudentName = "Bill" });
    studentList.Add(new Student(){ StudentName = "Steve" });
    studentList.Add(new Student(){ StudentName = "Ram" });

    ViewData["students"] = studentList;

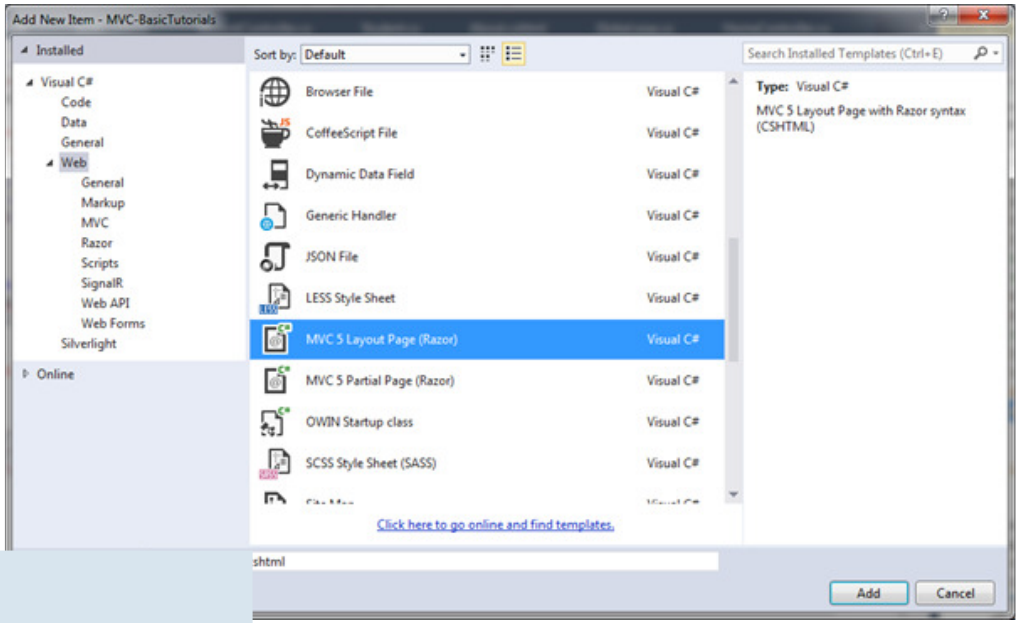
    return View();
}
```

Script Code

```
<ul>
@foreach (var std in ViewData["students"] as IList<Student>)
{
    <li>
        @std.StudentName
    </li>
}
```

Create Layout View

- Right click on shared folder -> select Add -> click on **New Item..**



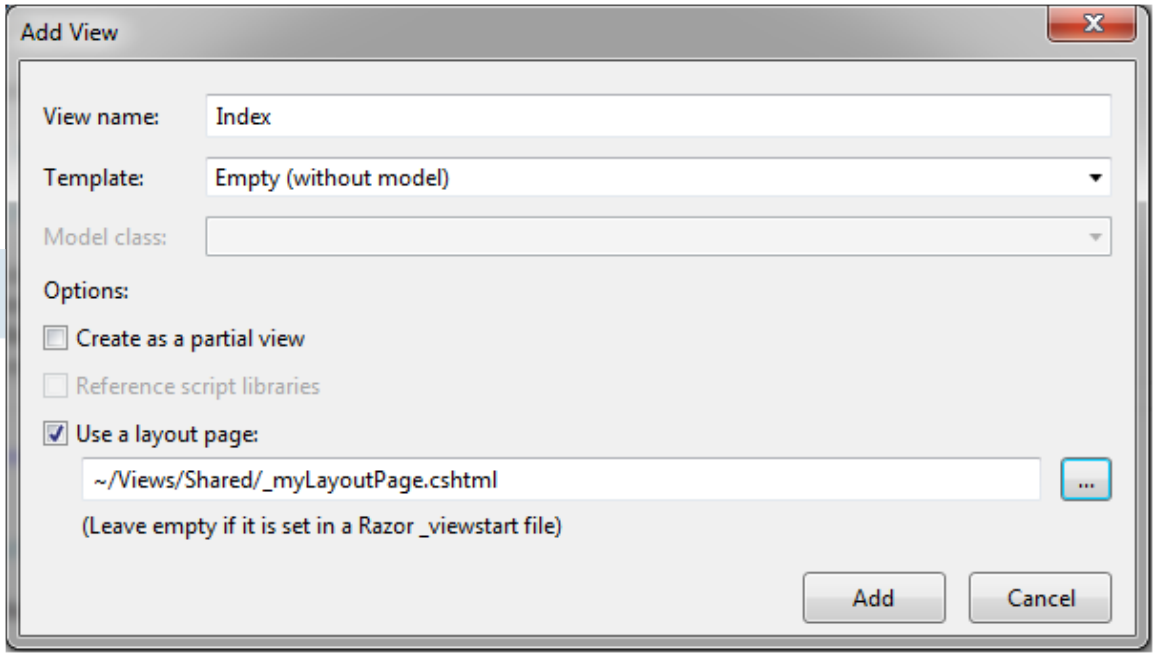
_myLayoutPage.cshtml:

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

Adding RenderSection at Layout View

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
        @Styles.Render("~/Content/css")
        @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div>
        @RenderBody()
    </div>
    <footer class="panel-footer">
        @RenderSection("footer", true)
    </footer>
</body>
</html>
```

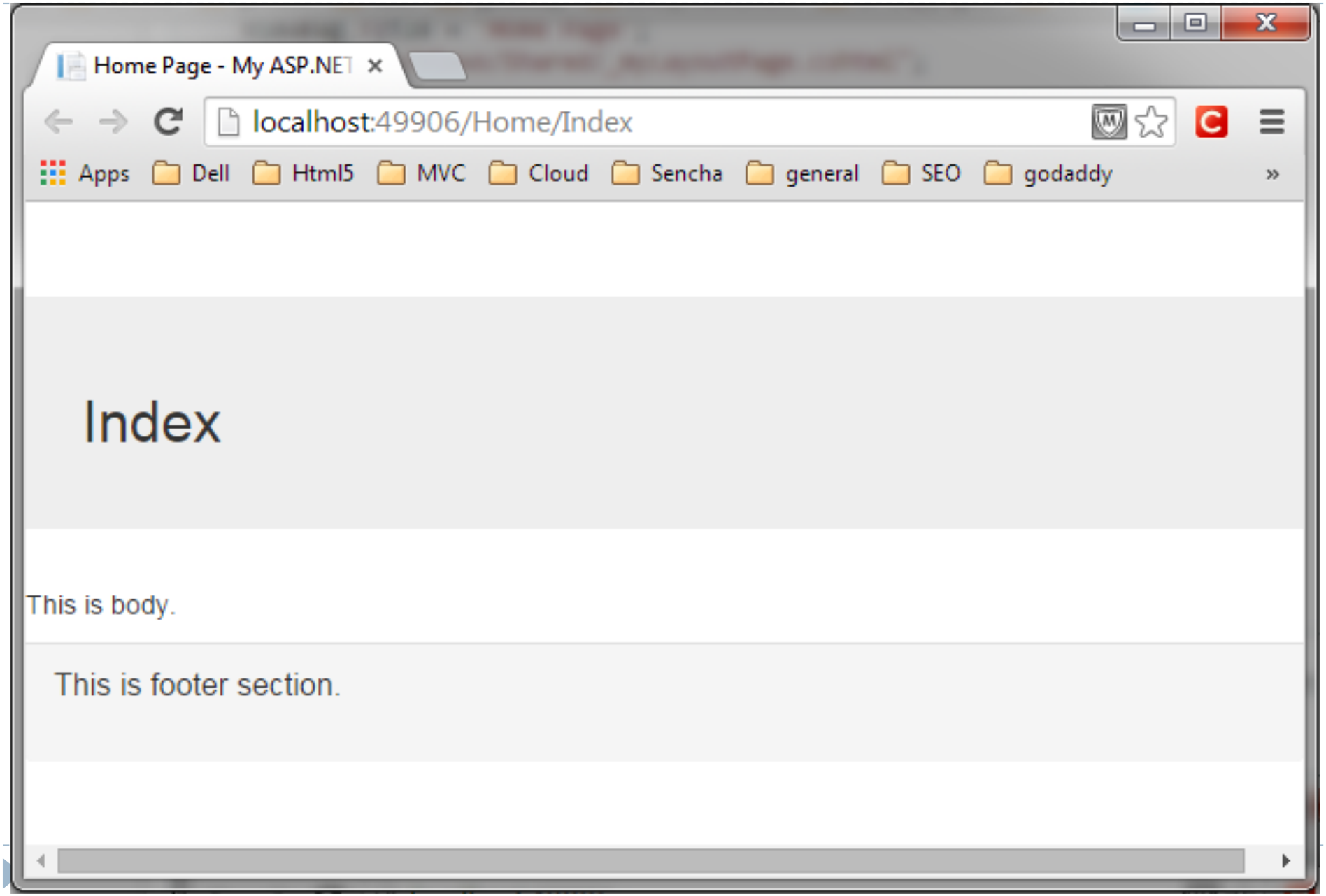
Use this _myLayoutPage.cshtml with the Index view of HomeController.



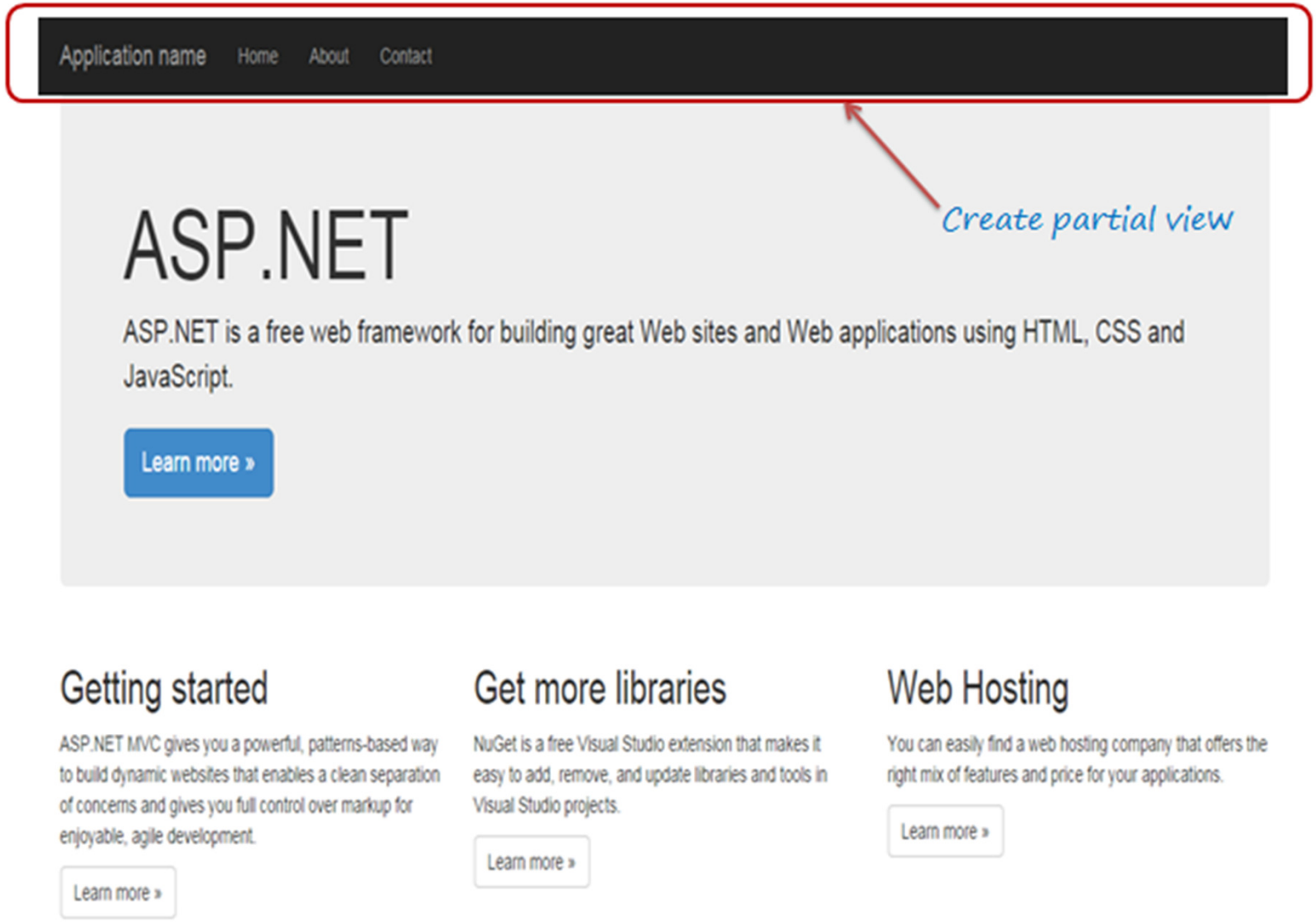
Index view:

```
@{
    ViewBag.Title = "Home Page";
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";
}

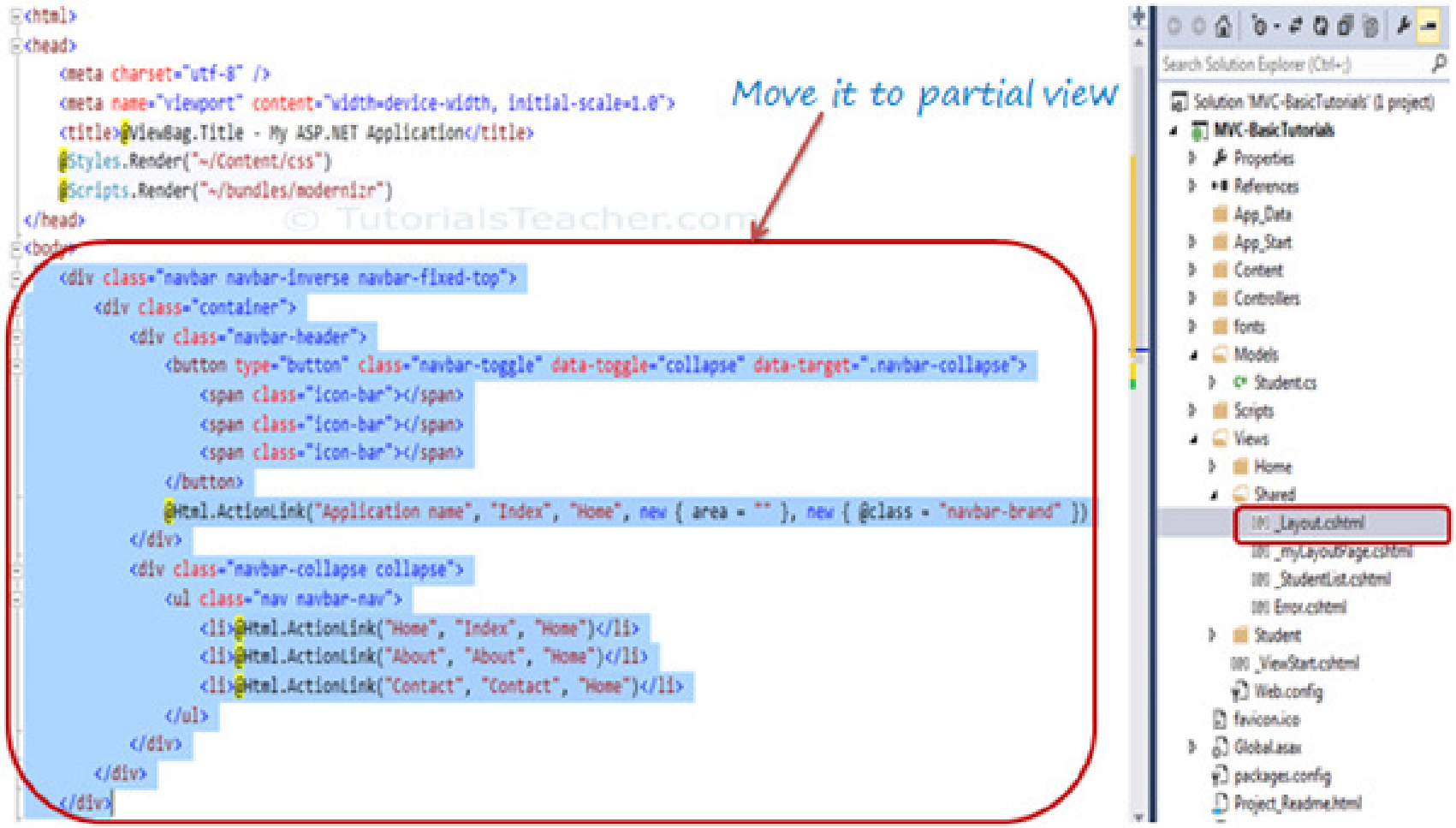
<div class="jumbotron">
    <h2>Index</h2>
</div>
<div class="row">
    <div class="col-md-4">
        <p>This is body.</p>
    </div>
    @section footer{
        <p class="lead">
            This is footer section.
        </p>
    }
</div>
```


Home Page - My ASP.NET
localhost:49906/Home/Index
Apps
Dell
Html5
MVC
Cloud
Sencha
general
SEO
godaddy
Index
This is body.
This is footer section.

Partial View



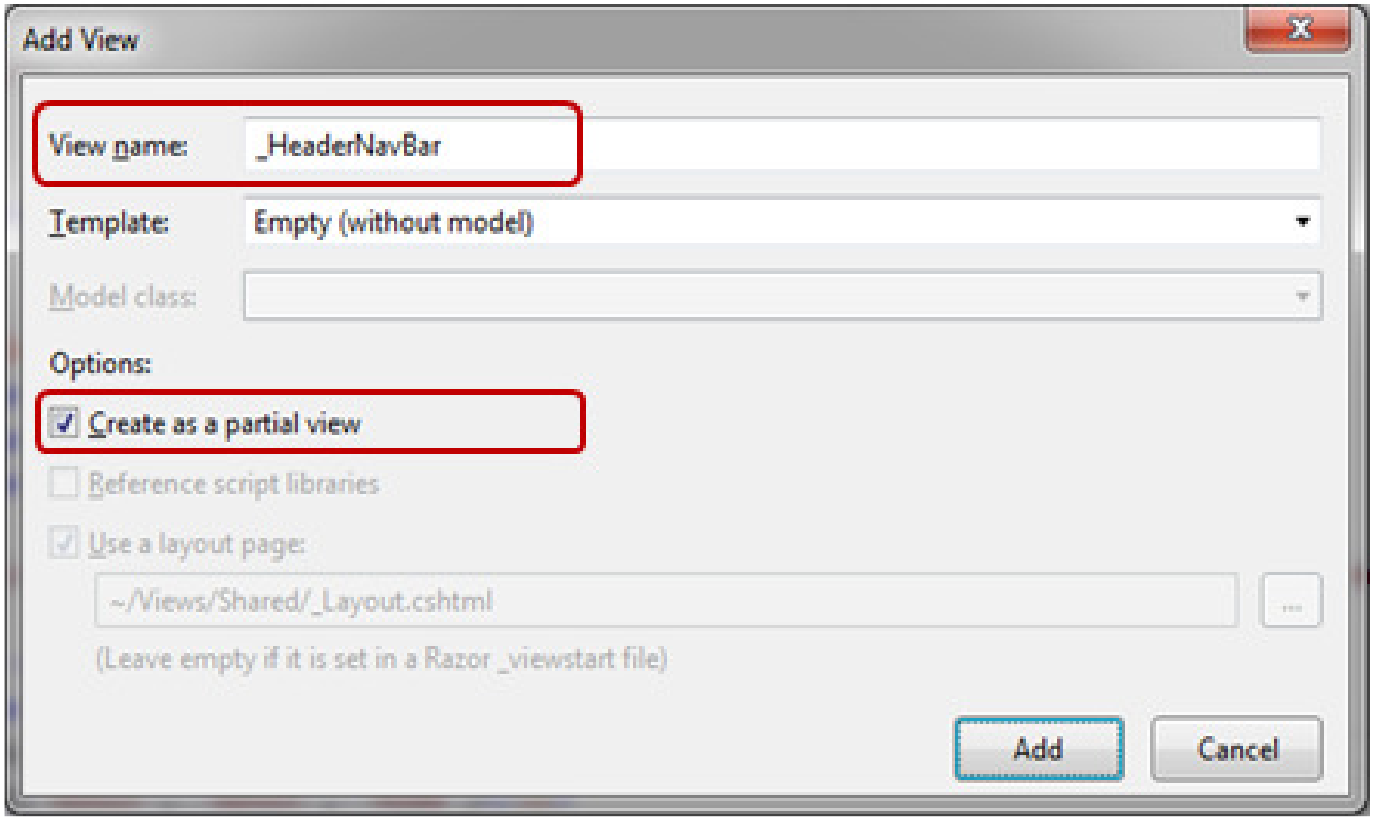
Partial View




```
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
```

Create a New Partial View

- Right click on Shared folder -> select **Add** -> click on **View..**



Example: Partial View _HeaderNavBar.cshtml

```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-c
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = '
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
        </div>
    </div>
</div>
```

Example: Html.RenderPartial()

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    @{
      Html.RenderPartial("_HeaderNavBar");
    }
    <div class="container body-content">
        @RenderBody()

        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>
        @Scripts.Render("~/bundles/jquery")
        @Scripts.Render("~/bundles/bootstrap")
        @RenderSection("scripts", required: false)
</body>
</html>
```

MCSE 541: Web Computing and Mining

Entity Framework Core Architecture

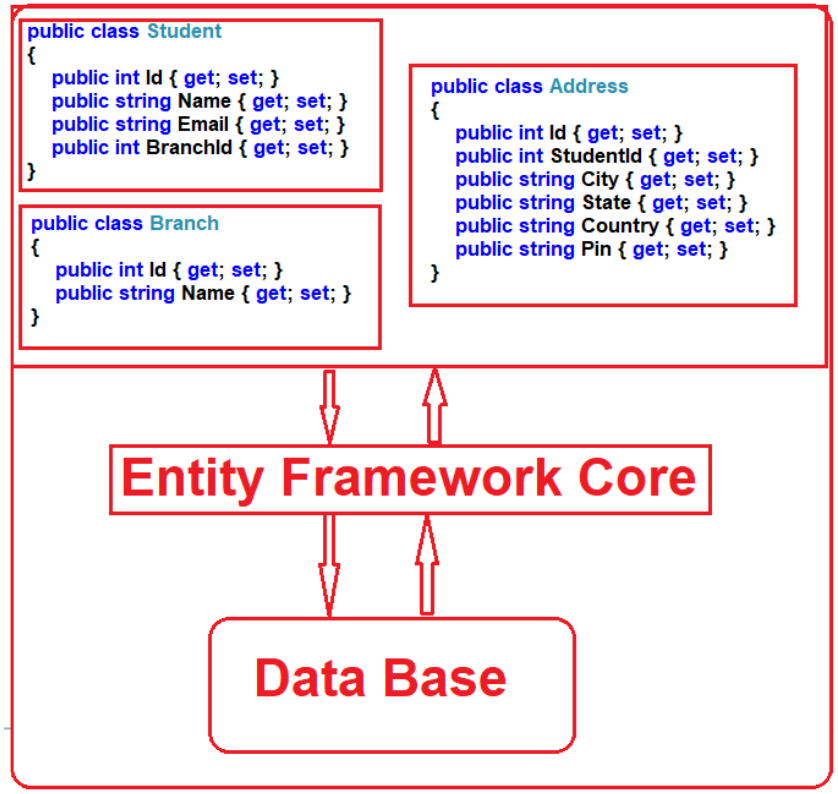
Prof. Dr. Shamim Akhter

EF Core

It is an extensible, lightweight, Open Source, and cross-platform version of Entity Framework data access technology. It works on multiple operating systems like Windows, Mac, and Linus.

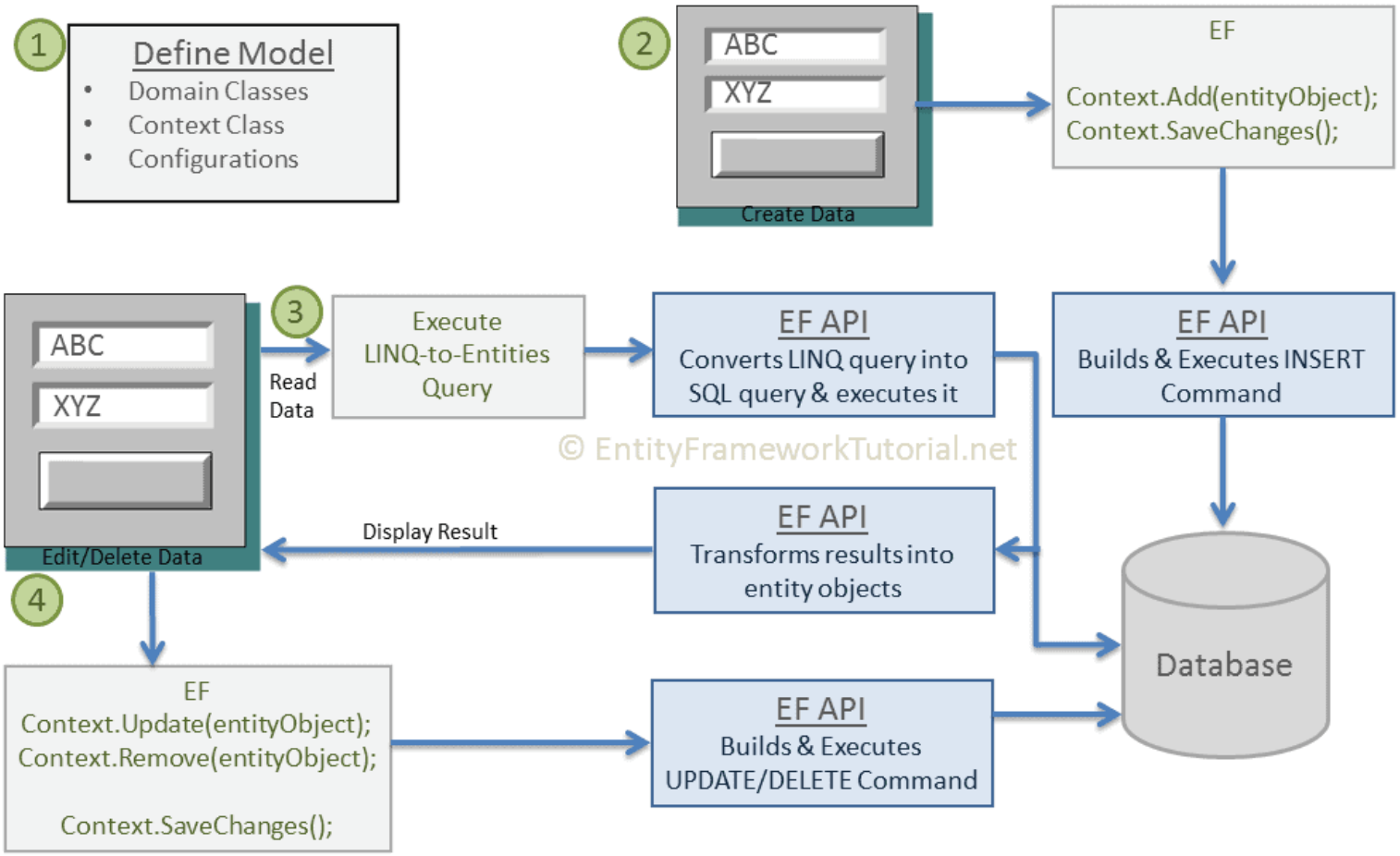
EF is an object-relational mapper (O/RM)

Object-Relational Mapper and it automatically creates classes based on database tables and the vice versa is also true. It can also generate the necessary SQL to create the database based on the classes.



Custom code to map the database data to our model classes like Student, Department, Address, etc. is slow.

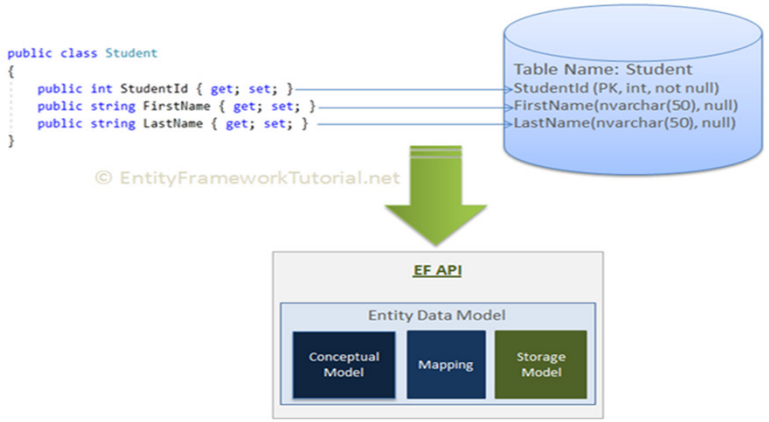
Basic Workflow in Entity Framework

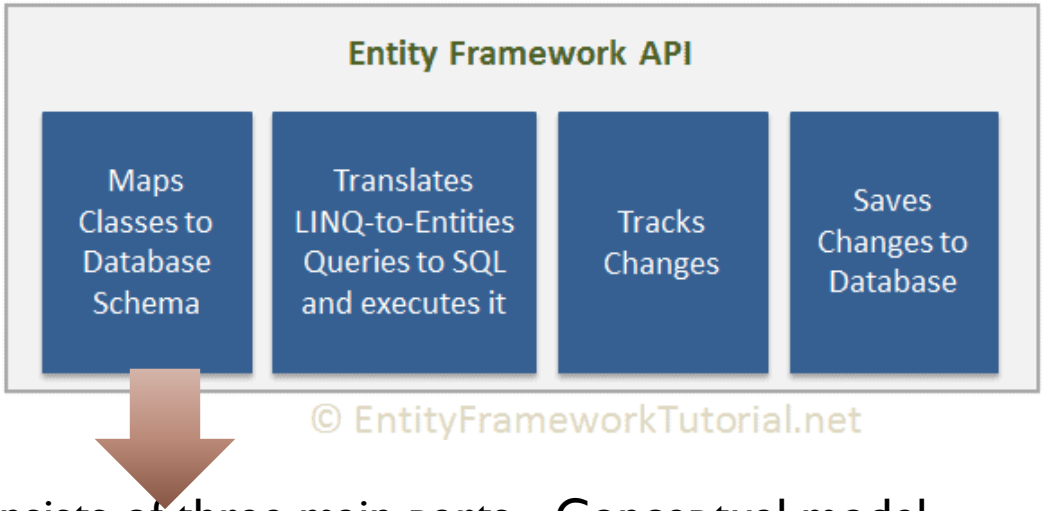


1. First of all, you need to define your model. Defining the model includes defining your domain classes, context class derived from DbContext, and configurations (if any). EF will perform CRUD operations based on your model.
2. To insert data, add a domain object to a context and call the SaveChanges() method. EF API will build an appropriate INSERT command and execute it to the database.
3. To read data, execute the LINQ(Language Integrated Query)-to-Entities query in your preferred language (C#/VB.NET). EF API will convert this query into SQL query for the underlying relational database and execute it. The result will be transformed into domain (entity) objects and displayed on the UI.
4. To edit or delete data, update or remove entity objects from a context and call the SaveChanges() method. EF API will build the appropriate UPDATE or DELETE command and execute it to the database.

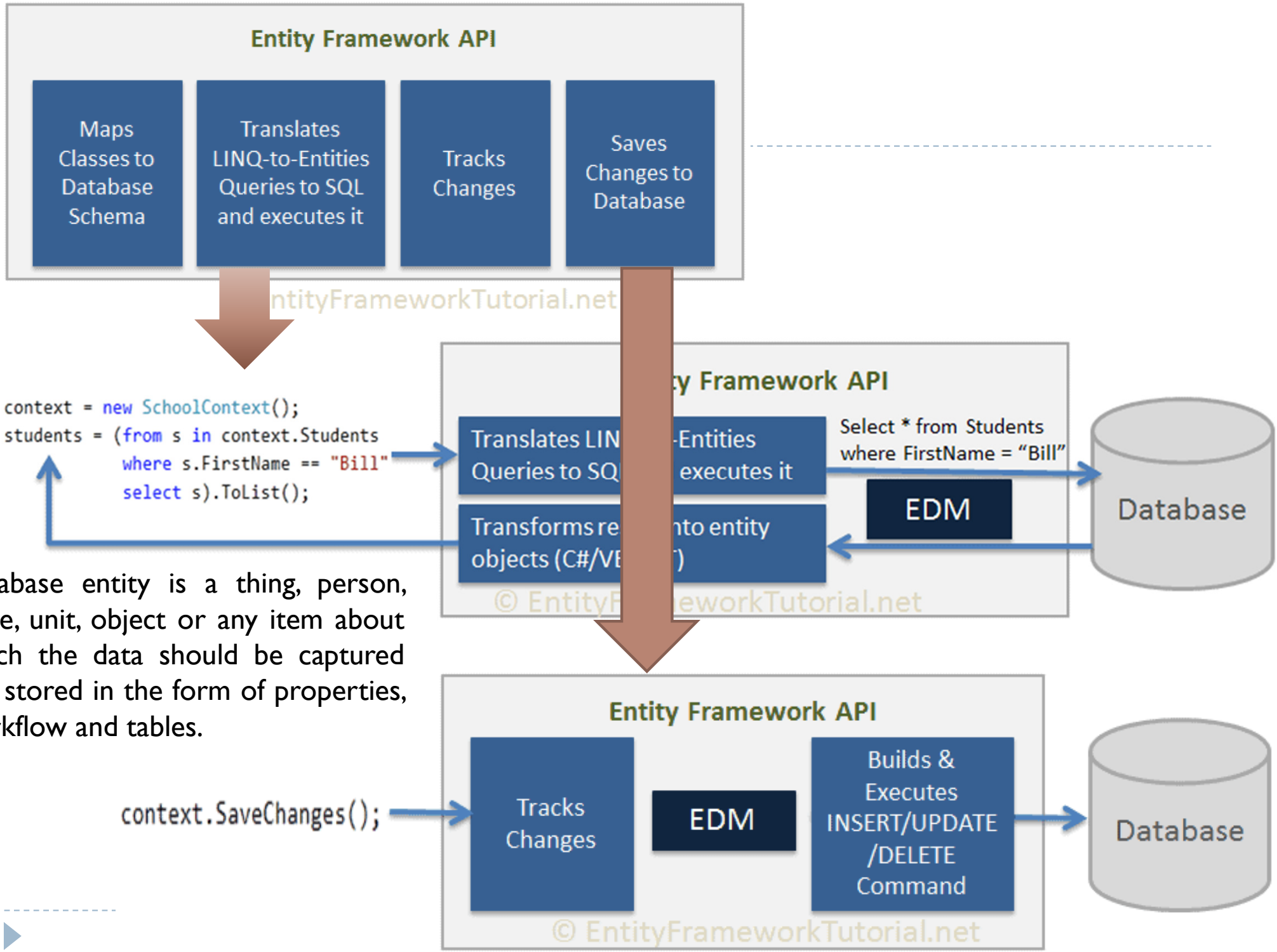
How Does Entity Framework Work?

- EF works on EF API





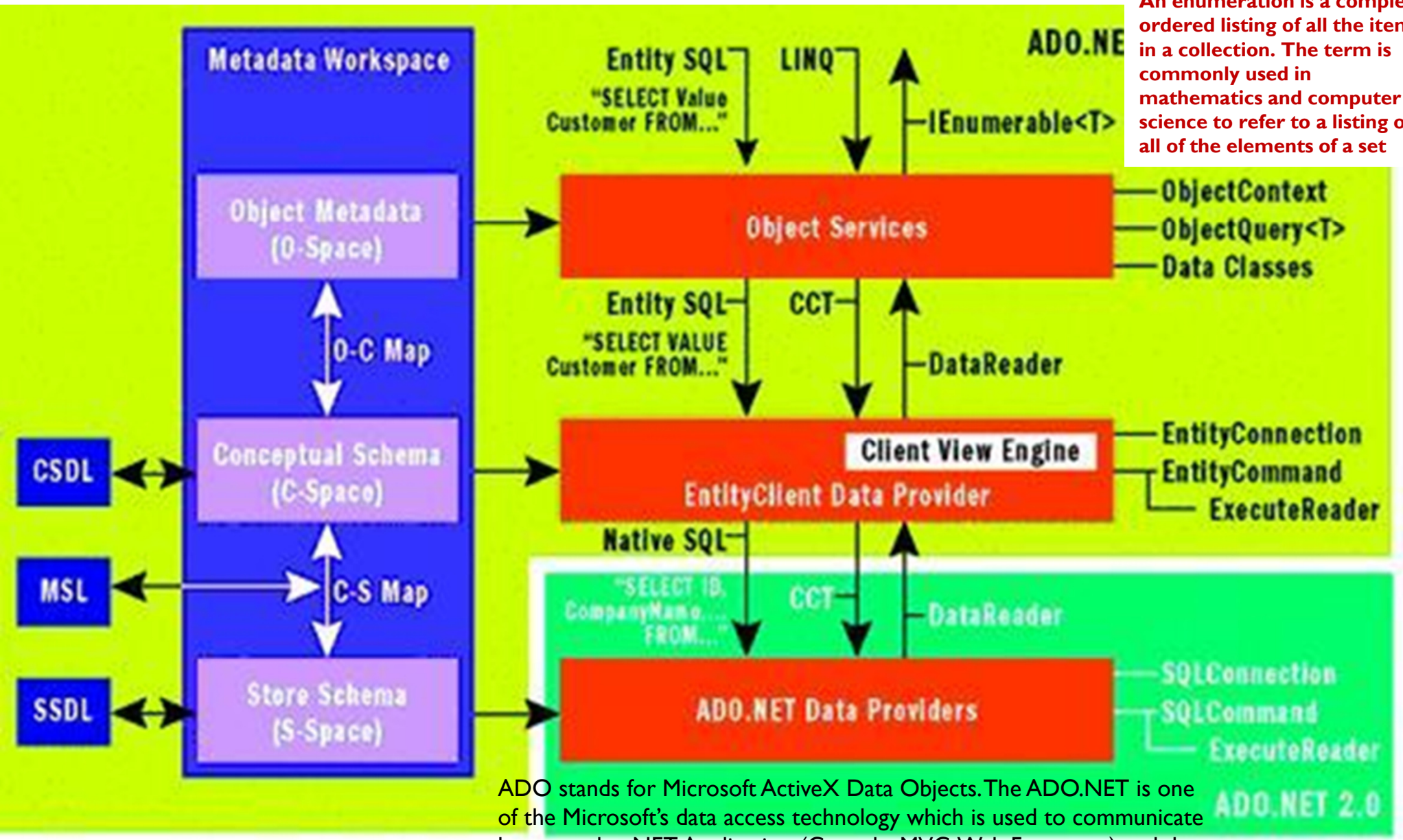
- **EDM (Entity Data Model):** EDM consists of three main parts - Conceptual model, Mapping and Storage model.
  - **Conceptual Model (C-Space)** contains the model classes and their relationships*(In XML). Conceptual Schema Definition Language (.CSDL) is used to map the entity types used in the conceptual model.* This will be independent from your database table design.
  - **Storage Model(S-Space)/Logical Model:** is the database design model which includes tables, views, stored procedures, and their relationships and keys. *Store Schema Definition Language (.SSDL) is used to map the schema information of the logical layer.*
  - In the code-first approach, this will be inferred from the conceptual model. In the database-first approach, this will be inferred from the targeted database.
  - **Mapping(C-S Space)** consists of information about how the conceptual model is mapped to the storage model. *Mapping Schema Language (.MSL) is used to map the logical model to the conceptual model.*


Entity Framework API
Maps Classes to Database Schema
Translates LINQ-to-Entities Queries to SQL and executes it
Tracks Changes
Saves Changes to Database
ntityFrameworkTutorial.net
var context = new SchoolContext();
var students = (from s in context.Students
where s.FirstName == "Bill"
select s).ToList();
y Framework API
Translates LIN -Entities
Queries to SQ executes it
Transforms re nto entity
objects (C#/VB)
Select * from Students
where FirstName = "Bill"
EDM
Database
© EntityF eworkTutorial.net
Database entity is a thing, person, place, unit, object or any item about which the data should be captured and stored in the form of properties, workflow and tables.
Entity Framework API
context.SaveChanges();
Tracks Changes
EDM
Builds & Executes INSERT/UPDATE /DELETE Command
Database
© EntityFrameworkTutorial.net

ADO.Net (ActiveX Data Object.NET (ADO.NET))

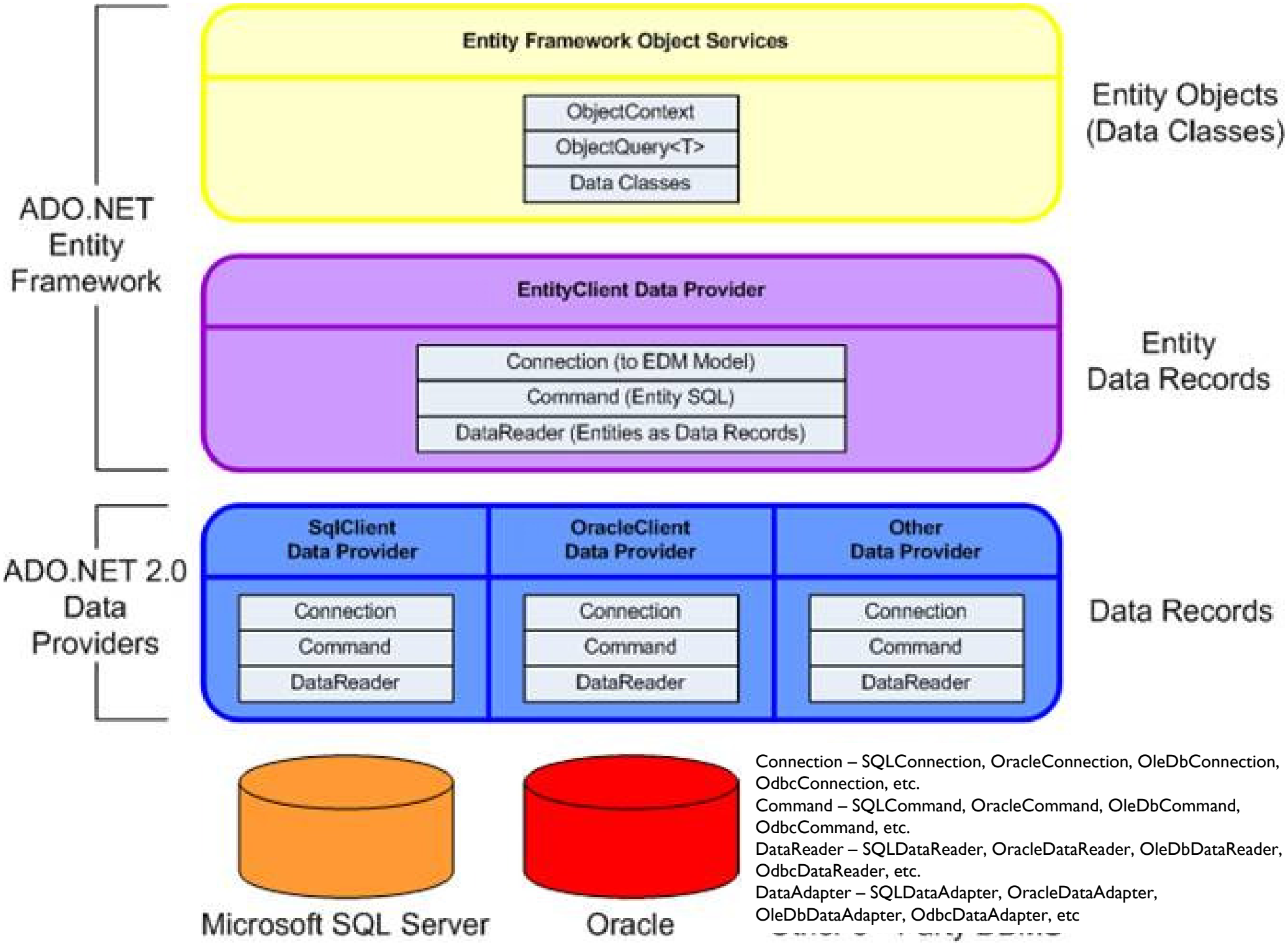
Data Provider

An enumeration is a complete, ordered listing of all the items in a collection. The term is commonly used in mathematics and computer science to refer to a listing of all of the elements of a set



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, MVC, Web Form, etc.) and data sources such as SQL Server, Oracle, MySQL, XML document, etc.

- **LINQ to Entities:** LINQ-to-Entities (L2E) is a query language used to write queries against the object model. It returns entities, which are defined in the conceptual model.
- **Entity SQL:** Entity SQL is another query language (For EF 6 only) just like LINQ to Entities. However, it is a little more difficult than L2E and the developer will have to learn it separately.
- **Object Service:** Object service is a main entry point for accessing data from the database and returning it back. Object service is responsible for materialization, which is the process of converting data returned from an entity client data provider (next layer) to an entity object structure.
- **Entity Client Data Provider:** The main responsibility of this layer is to convert LINQ-to-Entities or Entity SQL queries into a SQL query which is understood by the underlying database. It communicates with the ADO.Net data provider which in turn sends or retrieves data from the database.
- **ADO.Net (**ActiveX Data Object.NET (ADO.NET)**) Data Provider:** This layer communicates with the database using standard ADO.Net.


ADO.NET Entity Framework
Entity Framework Object Services
ObjectContext
ObjectQuery<T>
Data Classes
Entity Objects (Data Classes)
EntityClient Data Provider
Connection (to EDM Model)
Command (Entity SQL)
DataReader (Entities as Data Records)
Entity Data Records
ADO.NET 2.0 Data Providers
SqlClient Data Provider
OracleClient Data Provider
Other Data Provider
Connection
Command
DataReader
Connection
Command
DataReader
Connection
Command
DataReader
Data Records
Microsoft SQL Server
Oracle
Connection – SQLConnection, OracleConnection, OleDbConnection, OdbcConnection, etc.
Command – SQLCommand, OracleCommand, OleDbCommand, OdbcCommand, etc.
DataReader – SQLDataReader, OracleDataReader, OleDbDataReader, OdbcDataReader, etc.
DataAdapter – SQLDataAdapter, OracleDataAdapter, OleDbDataAdapter, OdbcDataAdapter, etc

ADO.NET data provider

```
OracleConnection connection = new OracleConnection("data source=.;
database=TestDB; integrated security=SSPI");

OracleCommand command = new OracleCommand("Select * from Customers",
connection);

connection.Open();

OracleDataReader myReader = command.ExecuteReader();

while (myReader.Read())
{
Console.WriteLine("\t{0}\t{1}", myReader.GetInt32(0), myReader.GetString(1));
}
connection.Close();
```

Entity Client Provider

```
string connectionString = "specify your connection string here…";

EntityConnection entityConnection = new EntityConnection(connectionString);

entityConnection.Open();


String queryString = "Select value a from IDGEntities.Author as a";

EntityCommand entityCommand = new EntityCommand(queryString, entityConnection);



EntityDataReader entityDataReader =
entityCommand.ExecuteReader(CommandBehavior.SequentialAccess);

 while (entityDataReader.Read())
        {
            Console.WriteLine(entityDataReader.GetValue(1));
        }
```

Entity Framework Object Services

The **ObjectContext** appears to have a very similar role to the **DataContext** in the LINQ to SQL world

Constructing an ObjectContext

We can construct an **ObjectContext** either by giving it a connection string, an existing **EntityConnection**

```
1 using (ObjectContext ctx = new ObjectContext("Name=NorthwindEntities"))
```

Starting to Query with an ObjectContext

Once we've got an **ObjectContext**, we can ask it to go ahead and create an **ObjectQuery<T>** for us by feeding it a piece of eSQL. For instance;

```
2 ObjectQuery<DbDataRecord> query = ctx.CreateQuery<DbDataRecord>(
  "select c.CompanyName, c.ContactName from NorthwindContext.Customers as c");
```

```
using System;
using System.Data.Common;
using System.Data.Objects;
using Northwind;

namespace ConsoleApplication4
{
  class Program
  {
    static void Main(string[] args)
    {
      using (ObjectContext ctx = new ObjectContext("Name=NorthwindEntities"))
      {
        ObjectQuery<Customers> query = ctx.CreateQuery<Customers>(
          "select value c from NorthwindContext.Customers as c");

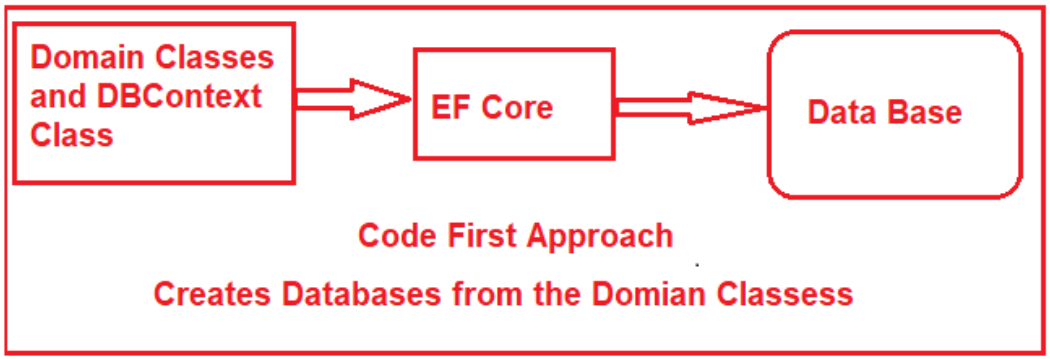
        ObjectResult<Customers> results = query.Execute(MergeOption.NoTracking);

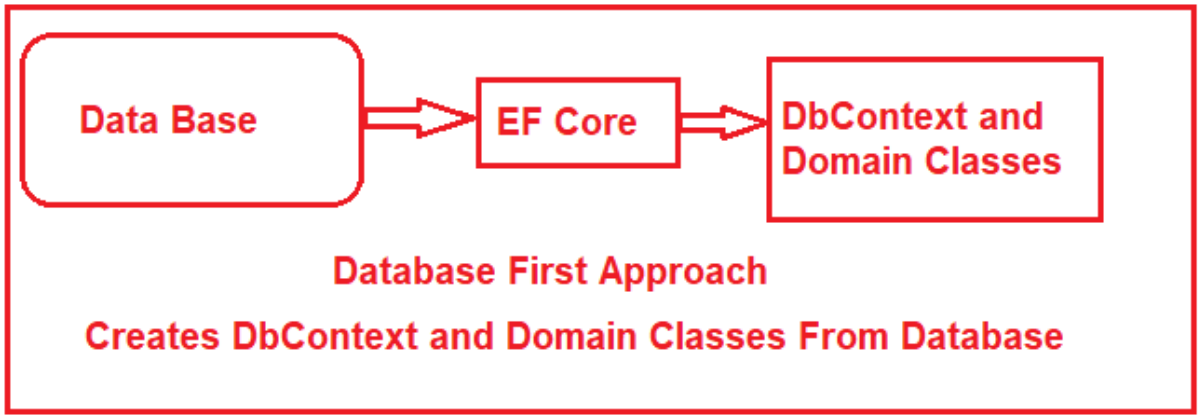
        foreach (Customers c in results)
        {
          Console.WriteLine("Company [{0}], Contact [{1}]",
            c.CompanyName, c.ContactName);
        }
      }
    }
```

EF Core Development Approaches

- Entity Framework Core supports two development approaches. They are as follows:

1. **Code-First Approach**
2. **Database-First Approach**





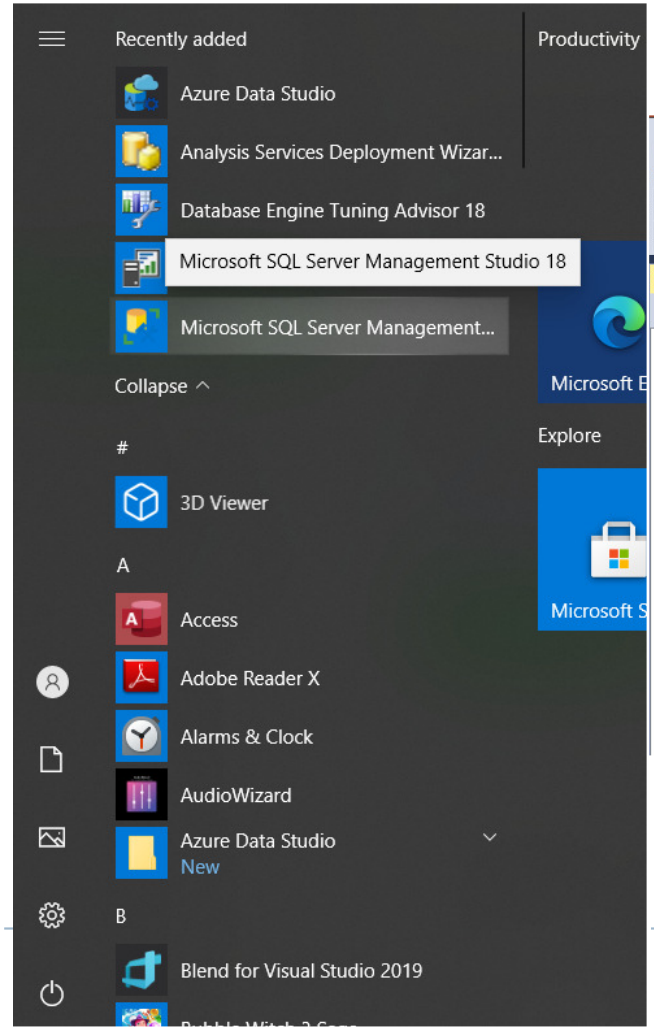
MCSE 541: Web Computing and Mining

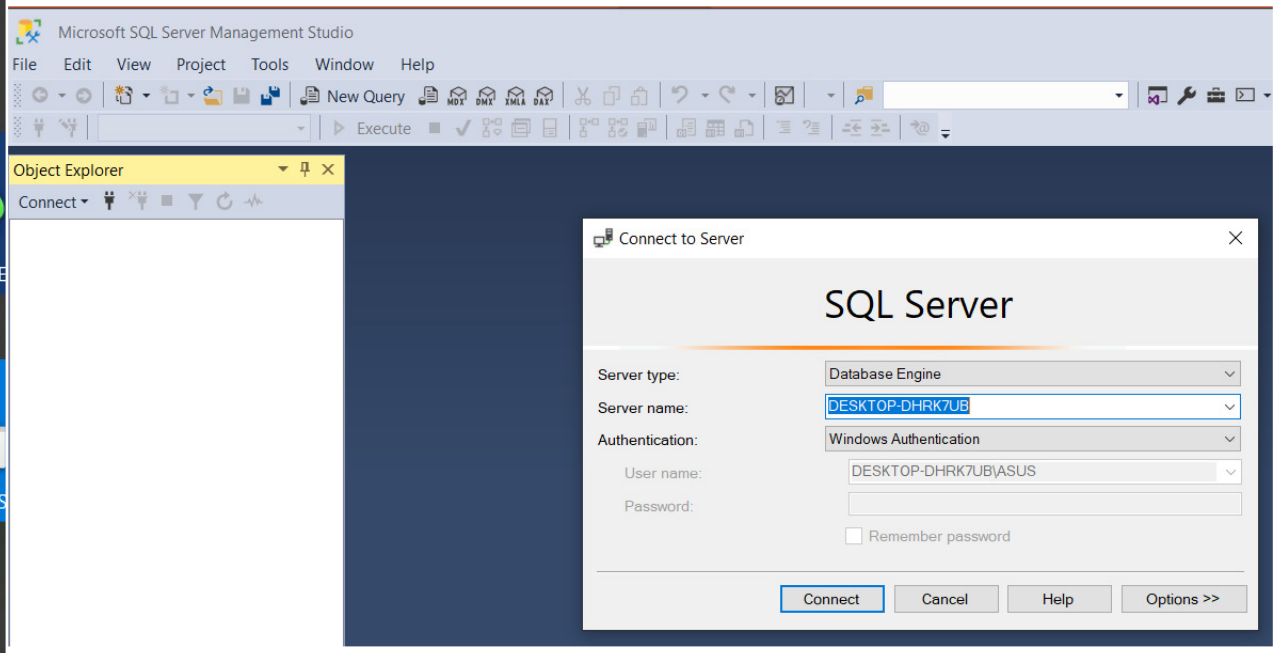
Entity Framework in ASP.NET
DB First EF Convention

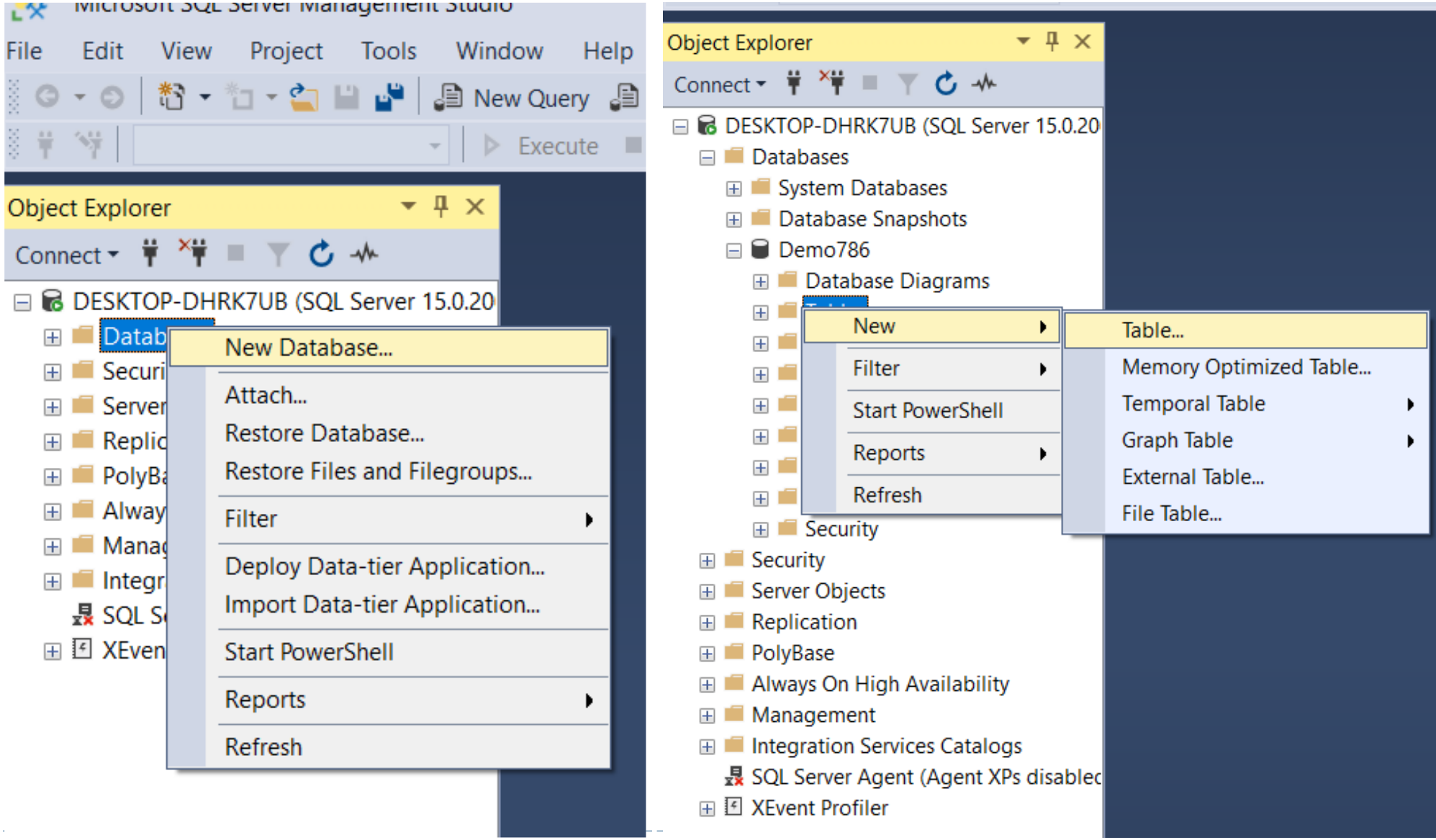
Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

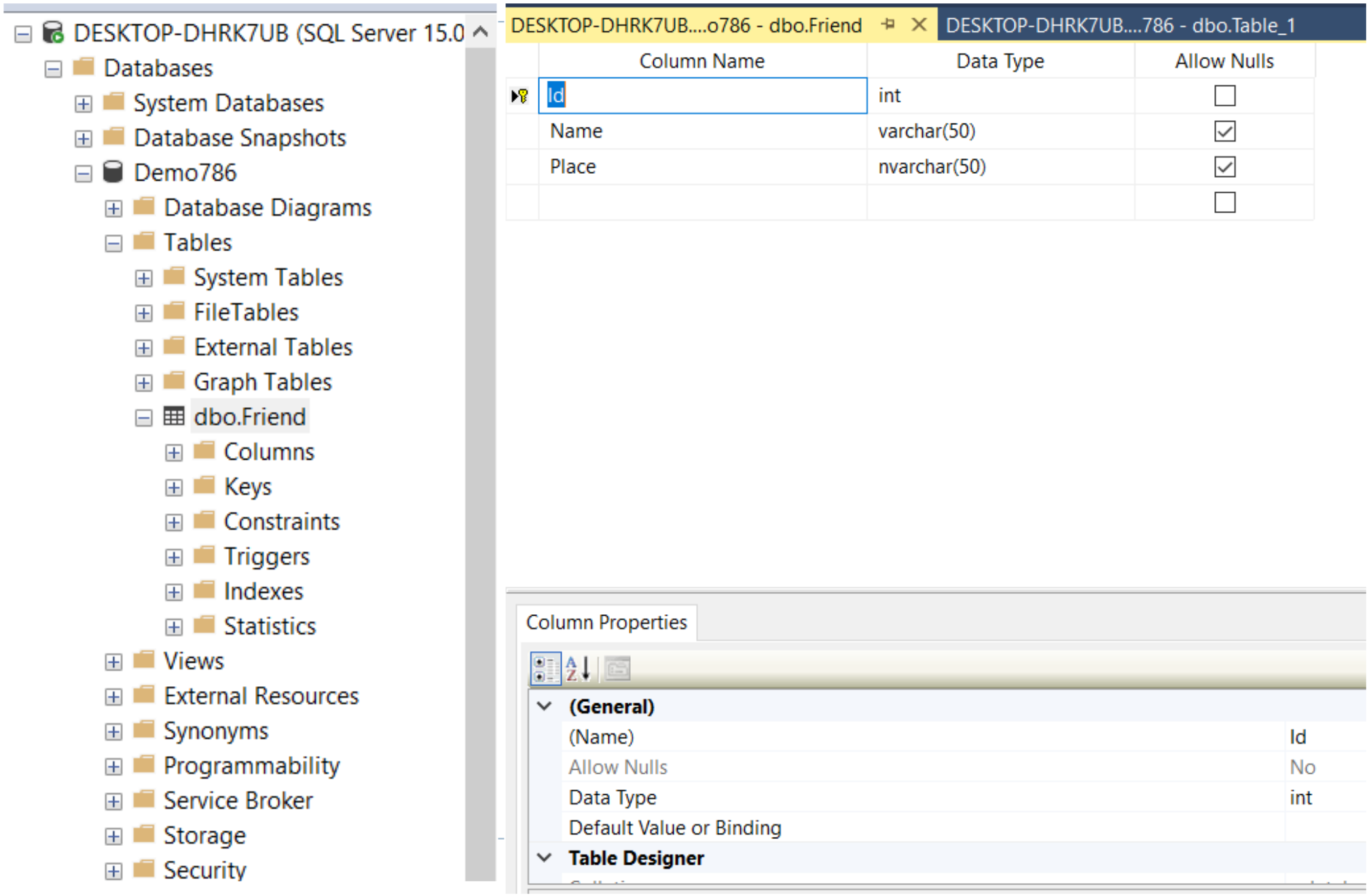
Install SQL Server

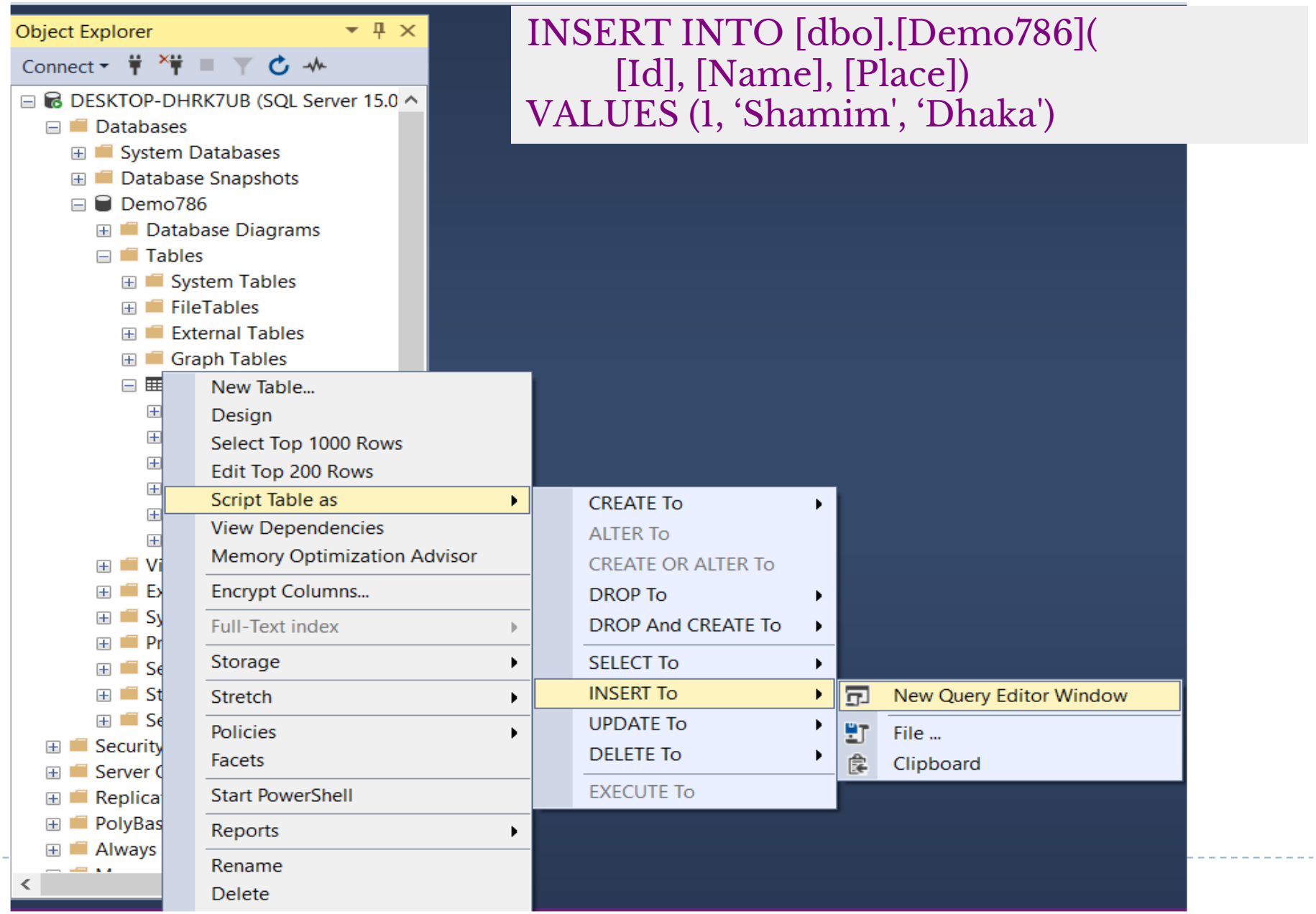
- How to download and install Microsoft SQL Server 2019
  - https://www.youtube.com/watch?v=QsXWszvjMBM

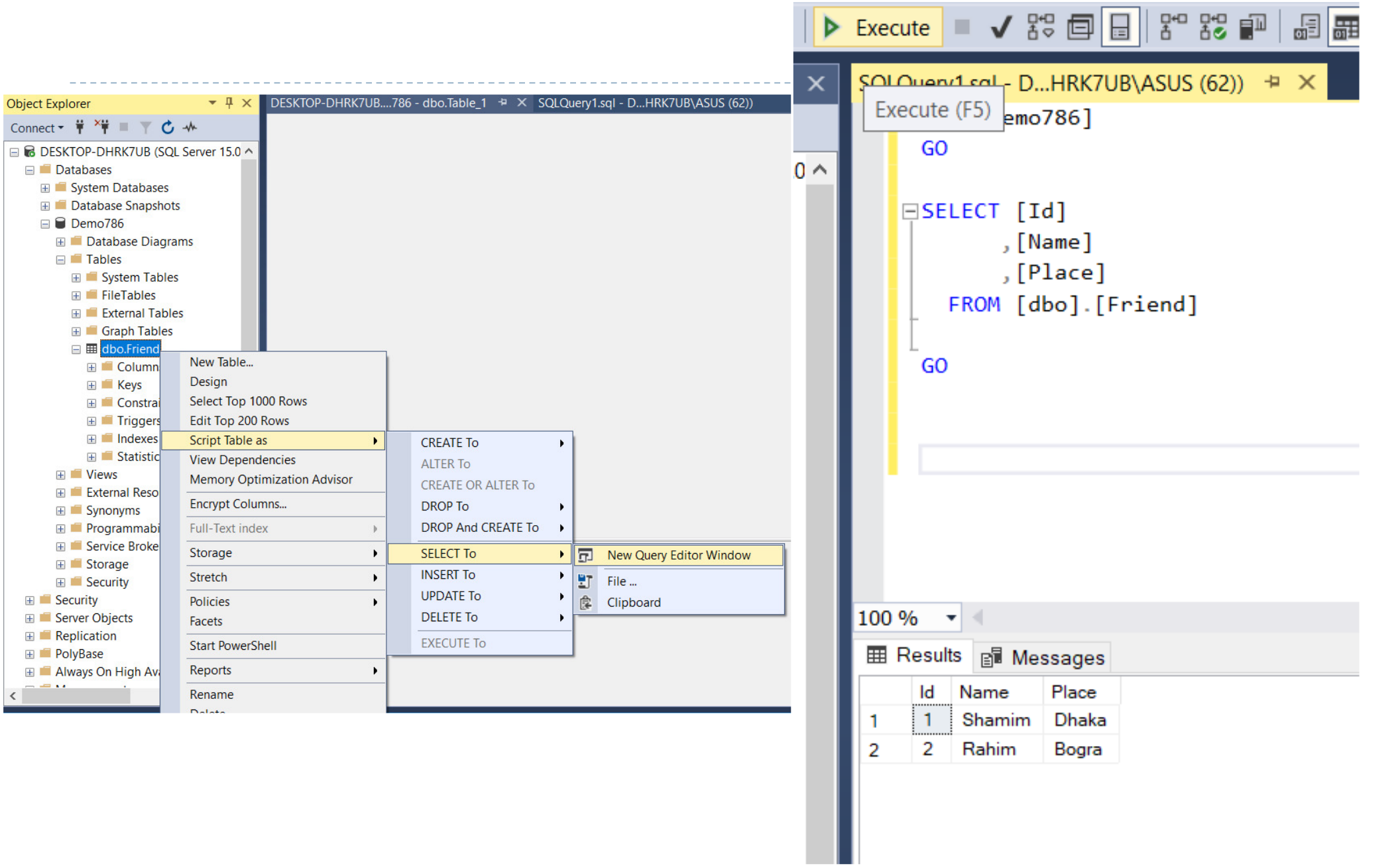




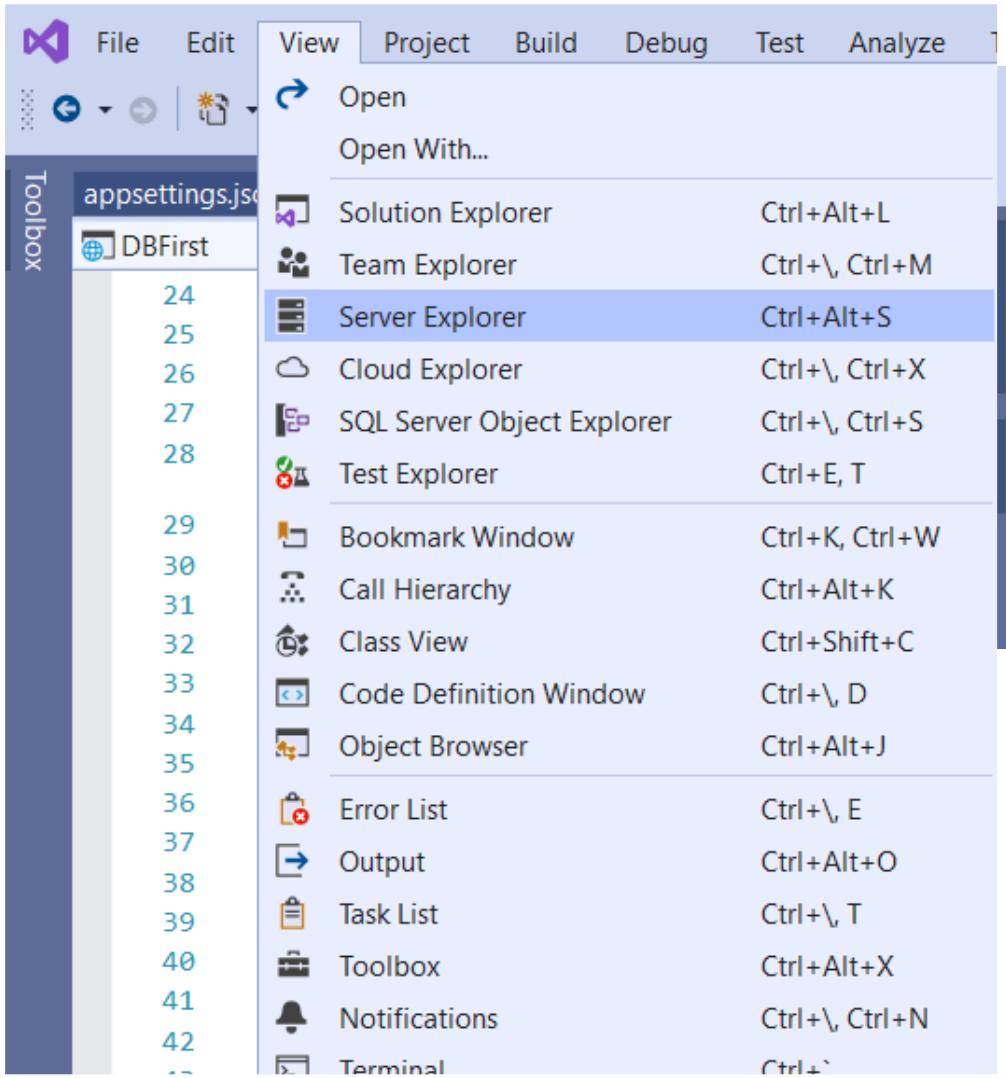

File Edit View Project Tools Window Help
New Query
Execute
Object Explorer
Connect
DESKTOP-DHRK7UB (SQL Server 15.0.20
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Filter
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh
Databases
System Databases
Database Snapshots
Demo786
Database Diagrams
New
Filter
Start PowerShell
Reports
Refresh
Table...
Memory Optimized Table...
Temporal Table
Graph Table
External Table...
File Table...
Security
Server Objects
Replication
PolyBase
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent (Agent XPs disabled
XEvent Profiler

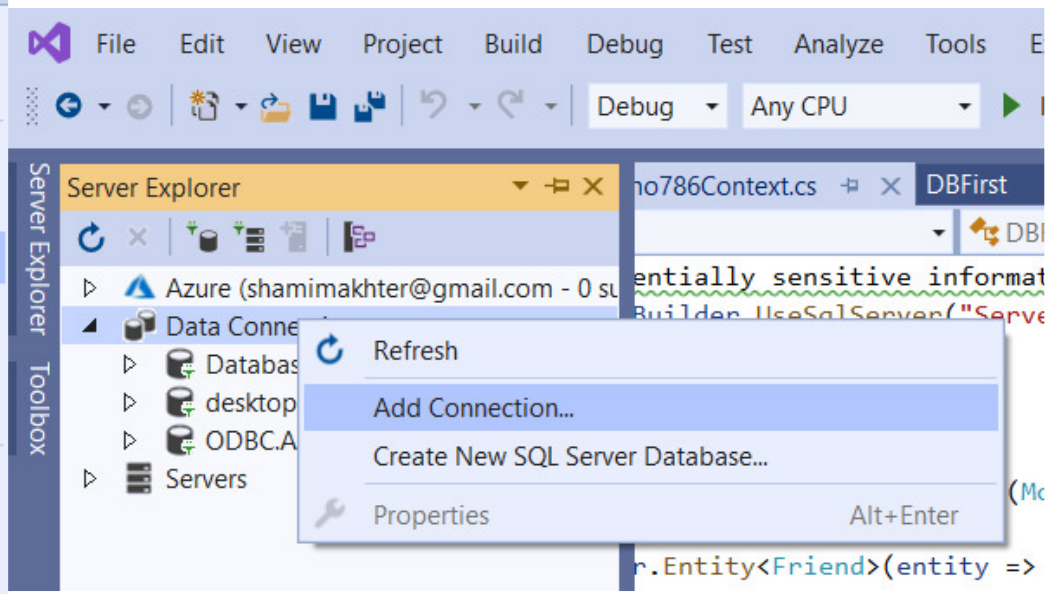

DESKTOP-DHRK7UB (SQL Server 15.0
Databases
System Databases
Database Snapshots
Demo786
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.Friend
Columns
Keys
Constraints
Triggers
Indexes
Statistics
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
DESKTOP-DHRK7UB....o786 - dbo.Friend
DESKTOP-DHRK7UB....786 - dbo.Table_1
Column Name
Data Type
Allow Nulls
Id
int
Name
varchar(50)
Place
nvarchar(50)
Column Properties
(General)
(Name)
Id
Allow Nulls
No
Data Type
int
Default Value or Binding
Table Designer

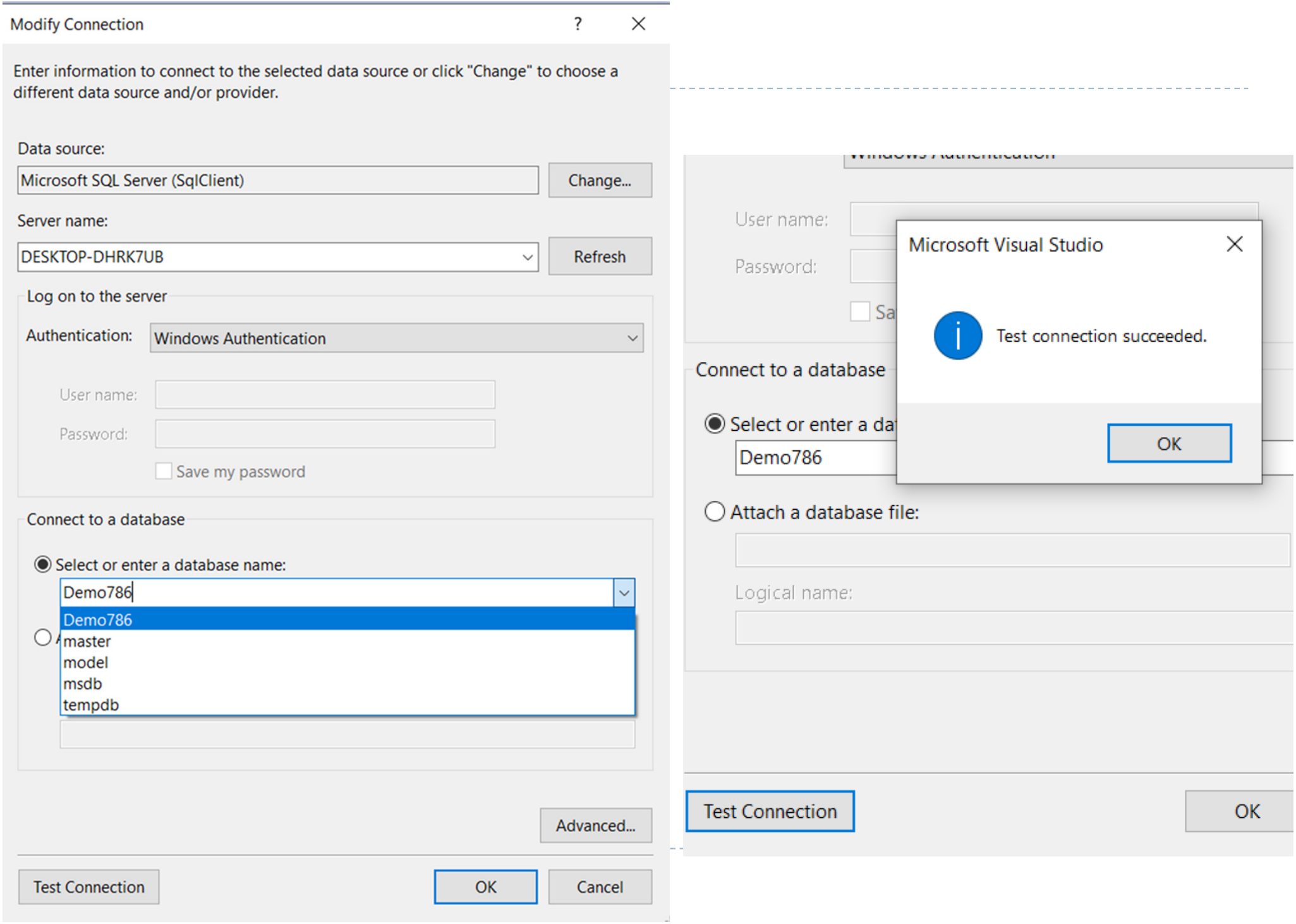

Object Explorer
Connect
DESKTOP-DHRK7UB (SQL Server 15.0
Databases
System Databases
Database Snapshots
Demo786
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
New Table...
Design
Select Top 1000 Rows
Edit Top 200 Rows
Script Table as
View Dependencies
Memory Optimization Advisor
Encrypt Columns...
Full-Text index
Storage
Stretch
Policies
Facets
Start PowerShell
Reports
Rename
Delete
CREATE To
ALTER To
CREATE OR ALTER To
DROP To
DROP And CREATE To
SELECT To
INSERT To
UPDATE To
DELETE To
EXECUTE To
New Query Editor Window
File ...
Clipboard
INSERT INTO [dbo].[Demo786](
[Id], [Name], [Place])
VALUES (1, 'Shamim', 'Dhaka')


Execute
Execute (F5)
SQLQuery1.sql - D...HRK7UB\ASUS (62))
emo786]
GO
SELECT [Id]
,[Name]
,[Place]
FROM [dbo].[Friend]
GO
Object Explorer
Connect
DESKTOP-DHRK7UB (SQL Server 15.0
Databases
System Databases
Database Snapshots
Demo786
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.Friend
Column
Keys
Constrai
Triggers
Indexes
Statistic
Views
External Reso
Synonyms
Programmabi
Service Broke
Storage
Security
Security
Server Objects
Replication
PolyBase
Always On High Av
DESKTOP-DHRK7UB....786 - dbo.Table_1
SQLQuery1.sql - D...HRK7UB\ASUS (62))
New Table...
Design
Select Top 1000 Rows
Edit Top 200 Rows
Script Table as
View Dependencies
Memory Optimization Advisor
Encrypt Columns...
Full-Text index
Storage
Stretch
Policies
Facets
Start PowerShell
Reports
Rename
CREATE To
ALTER To
CREATE OR ALTER To
DROP To
DROP And CREATE To
SELECT To
INSERT To
UPDATE To
DELETE To
EXECUTE To
New Query Editor Window
File ...
Clipboard
100 %
Results
Messages
Id
Name
Place
1
1
Shamim
Dhaka
2
2
Rahim
Bogra

Visual Studio






Modify Connection
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (SqlClient)
Change...
Server name:
DESKTOP-DHRK7UB
Refresh
Log on to the server
Authentication:
Windows Authentication
User name:
Password:
Save my password
Connect to a database
Select or enter a database name:
Demo786
Demo786
master
model
msdb
tempdb
Advanced...
Test Connection
OK
Cancel
Microsoft Visual Studio
Test connection succeeded.
OK
Connect to a database
Select or enter a da
Demo786
Attach a database file:
Logical name:
Test Connection
OK

Scaffold-DbContext Command

- EF Core does not support visual designer for DB model and wizard to create the entity and context classes similar to EF 6.
- So, we need to do reverse engineering using the Scaffold-DbContext command.
  - This reverse engineering command creates model and context classes (by deriving DbContext) based on the schema of the existing database.

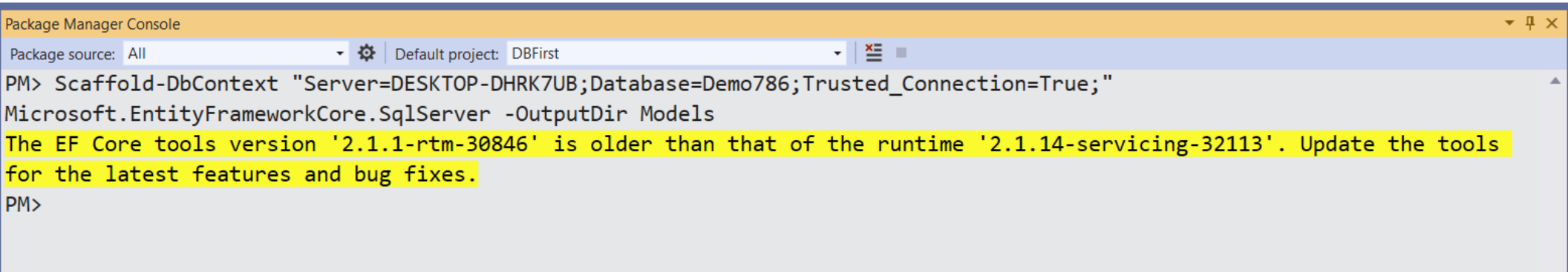
Use Scaffold-DbContext to create a model based on your existing database. The following parameters can be specified with Scaffold-DbContext in Package Manager Console:

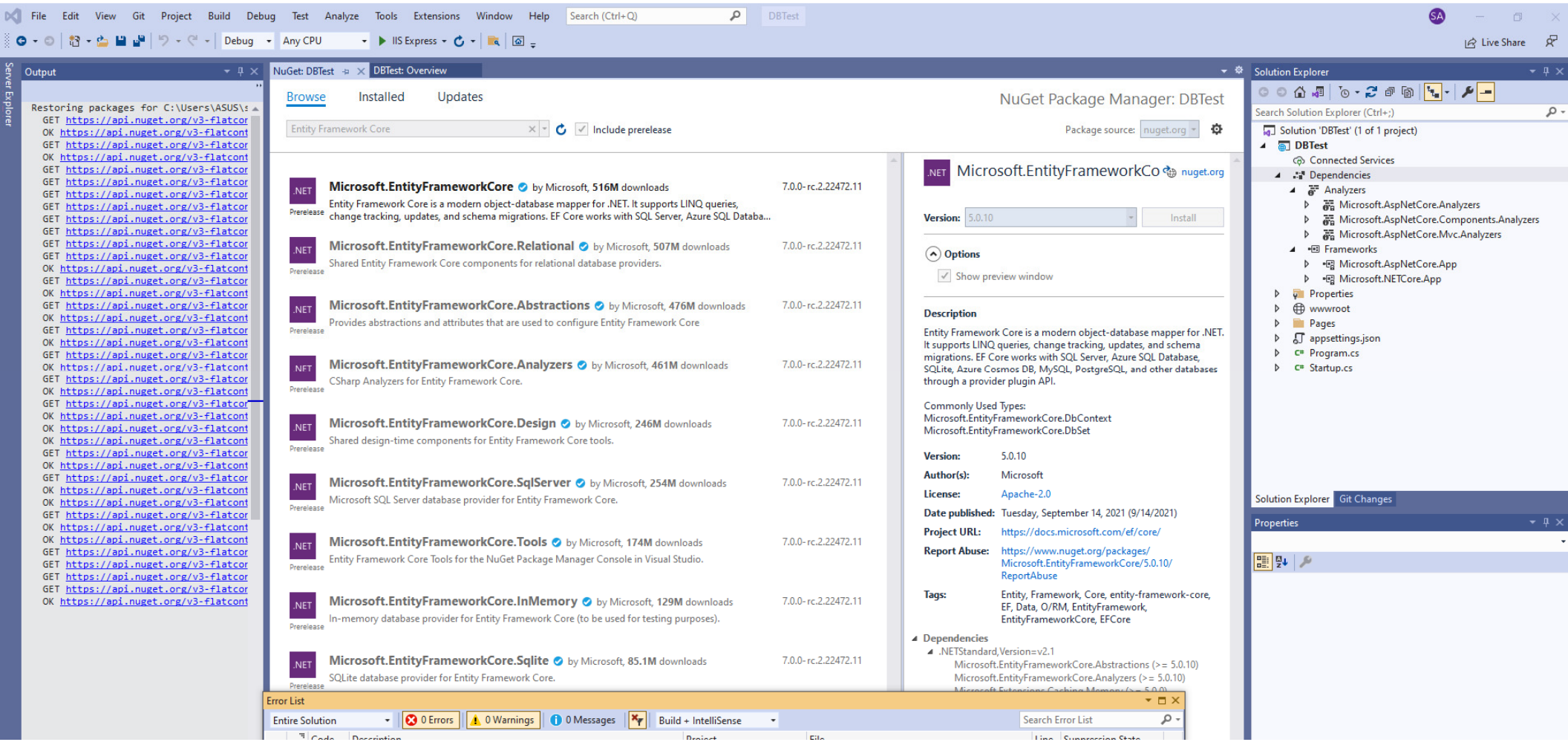
Scaffold-DbContext [-Connection] [-Provider] [-OutputDir] [-Context] [-Schemas>] [-Tables>] [-DataAnnotations] [-Force] [-Project] [-StartupProject] [<CommonParameters>]

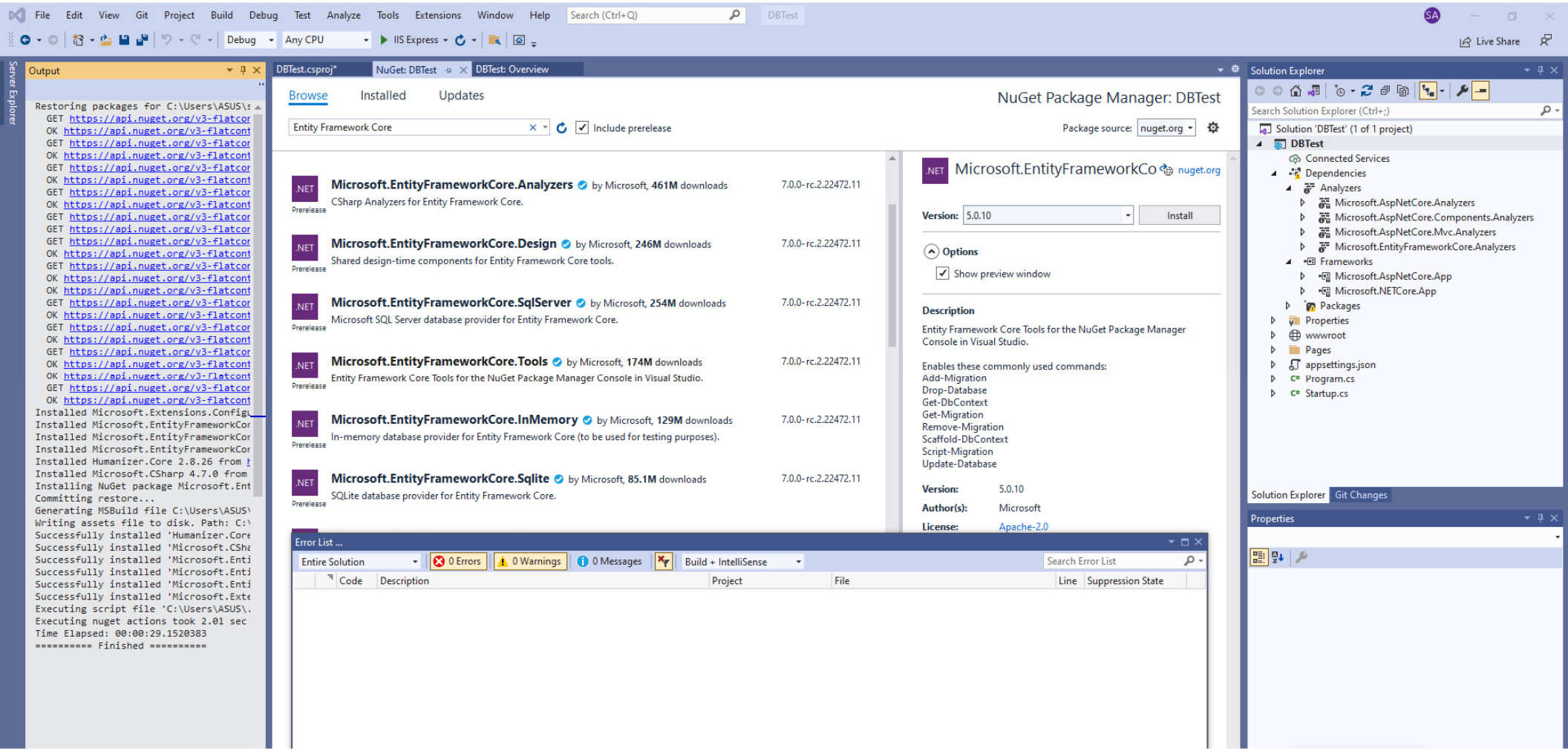
```
Scaffold-DbContext
"Server=.\SQLExpress;Database=SchoolDB;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Model
```

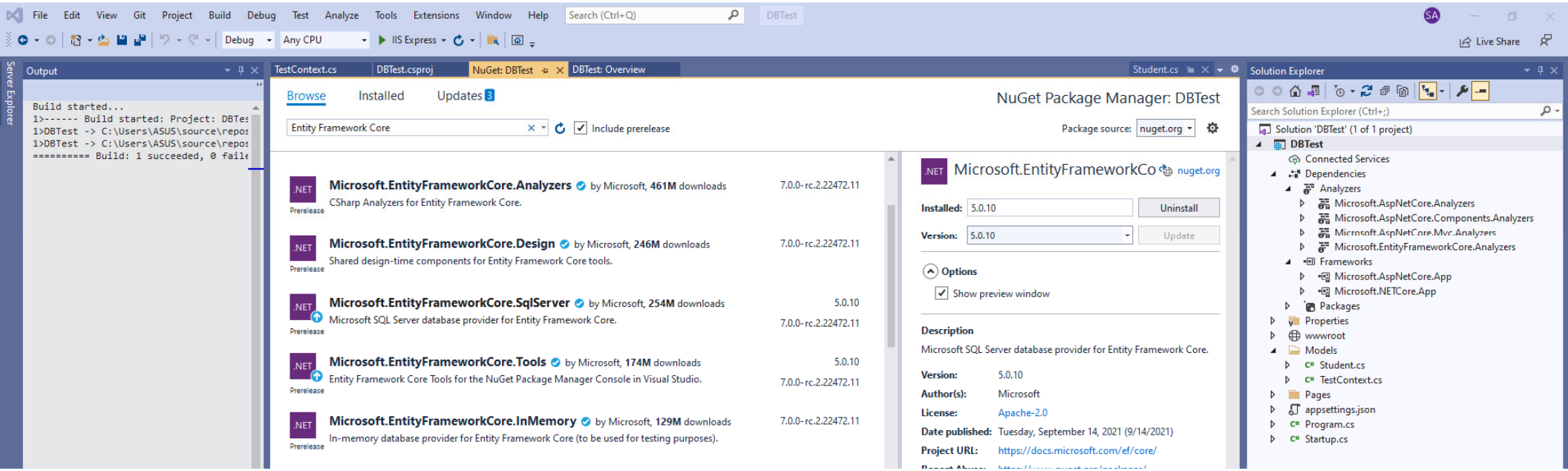
The first parameter is a connection string which includes three parts: DB Server, database name and security info. Here, Server=.\SQLExpress; refers to local SQLEXPRESS database server. Database=SchoolDB; specifies the database name "SchoolDB" for which we are going to create classes. Trusted_Connection=True; specifies the Windows authentication. It will use Windows credentials to connect to the SQL Server. The second parameter is the provider name. We use provider for the SQL Server, so it is Microsoft.EntityFrameworkCore.SqlServer. The -OutputDir parameter specifies the directory where we want to generate all the classes which is the Models folder in this case.

- ```
  Scaffold-DbContext "Server=DESKTOP-
  DHRK7UB;Database=StudentDB;Trusted_Connection=True;"
  Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
  ```

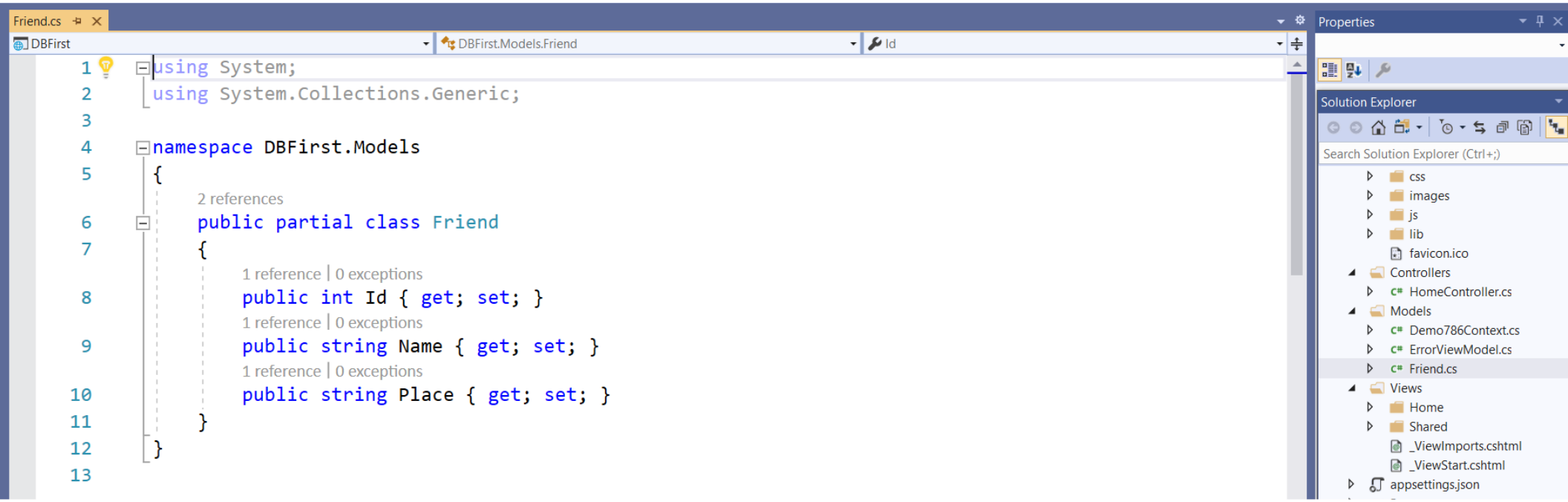

  ```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
DBTest
Debug
Any CPU
IIS Express
Live Share
Output
Restoring packages for C:\Users\ASUS\s
GET https://api.nuget.org/v3-flatcor
OK https://api.nuget.org/v3-flatcont
NuGet: DBTest
DBTest: Overview
Browse
Installed
Updates
NuGet Package Manager: DBTest
Entity Framework Core
Include prerelease
Package source: nuget.org
Microsoft.EntityFrameworkCore by Microsoft, 516M downloads 7.0.0-rc.2.22472.11
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Databa...
Microsoft.EntityFrameworkCore.Relational by Microsoft, 507M downloads 7.0.0-rc.2.22472.11
Shared Entity Framework Core components for relational database providers.
Microsoft.EntityFrameworkCore.Abstractions by Microsoft, 476M downloads 7.0.0-rc.2.22472.11
Provides abstractions and attributes that are used to configure Entity Framework Core
Microsoft.EntityFrameworkCore.Analyzers by Microsoft, 461M downloads 7.0.0-rc.2.22472.11
CSharp Analyzers for Entity Framework Core.
Microsoft.EntityFrameworkCore.Design by Microsoft, 246M downloads 7.0.0-rc.2.22472.11
Shared design-time components for Entity Framework Core tools.
Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 254M downloads 7.0.0-rc.2.22472.11
Microsoft SQL Server database provider for Entity Framework Core.
Microsoft.EntityFrameworkCore.Tools by Microsoft, 174M downloads 7.0.0-rc.2.22472.11
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
Microsoft.EntityFrameworkCore.InMemory by Microsoft, 129M downloads 7.0.0-rc.2.22472.11
In-memory database provider for Entity Framework Core (to be used for testing purposes).
Microsoft.EntityFrameworkCore.Sqlite by Microsoft, 85.1M downloads 7.0.0-rc.2.22472.11
SQLite database provider for Entity Framework Core.
Prerelease
Microsoft.EntityFrameworkCo
nuget.org
Version: 5.0.10
Install
Options
Show preview window
Description
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.
Commonly Used Types:
Microsoft.EntityFrameworkCore.DbContext
Microsoft.EntityFrameworkCore.DbSet
Version: 5.0.10
Author(s): Microsoft
License: Apache-2.0
Date published: Tuesday, September 14, 2021 (9/14/2021)
Project URL: https://docs.microsoft.com/ef/core/
Report Abuse: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore/5.0.10/ReportAbuse
Tags: Entity, Framework, Core, entity-framework-core, EF, Data, O/RM, EntityFramework, EntityFrameworkCore, EFCore
Dependencies
.NETStandard,Version=v2.1
Microsoft.EntityFrameworkCore.Abstractions (>= 5.0.10)
Microsoft.EntityFrameworkCore.Analyzers (>= 5.0.10)
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DBTest' (1 of 1 project)
DBTest
Connected Services
Dependencies
Analyzers
Microsoft.AspNetCore.Analyzers
Microsoft.AspNetCore.Components.Analyzers
Microsoft.AspNetCore.Mvc.Analyzers
Frameworks
Microsoft.AspNetCore.App
Microsoft.NETCore.App
Properties
wwwroot
Pages
appsettings.json
Program.cs
Startup.cs
Solution Explorer
Git Changes
Properties


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
DBTest
Debug
Any CPU
IIS Express
Live Share
Output
Restoring packages for C:\Users\ASUS\s
GET https://api.nuget.org/v3-flatcor
OK https://api.nuget.org/v3-flatcont
Installed Microsoft.Extensions.Configu
Installed Microsoft.EntityFrameworkCor
Installed Microsoft.EntityFrameworkCor
Installed Microsoft.EntityFrameworkCor
Installed Humanizer.Core 2.8.26 from
Installed Microsoft.CSharp 4.7.0 from
Installing NuGet package Microsoft.Ent
Committing restore...
Generating MSBuild file C:\Users\ASUS\
Writing assets file to disk. Path: C:\
Successfully installed 'Humanizer.Core
Successfully installed 'Microsoft.CSha
Successfully installed 'Microsoft.Enti
Successfully installed 'Microsoft.Enti
Successfully installed 'Microsoft.Enti
Successfully installed 'Microsoft.Exte
Executing script file 'C:\Users\ASUS\.
Executing nuget actions took 2.01 sec
Time Elapsed: 00:00:29.1520383
========== Finished ==========
DBTest.csproj*
NuGet: DBTest
DBTest: Overview
Browse
Installed
Updates
NuGet Package Manager: DBTest
Entity Framework Core
Include prerelease
Package source: nuget.org
Microsoft.EntityFrameworkCore.Analyzers by Microsoft, 461M downloads
CSharp Analyzers for Entity Framework Core.
7.0.0-rc.2.22472.11
Microsoft.EntityFrameworkCore.Design by Microsoft, 246M downloads
Shared design-time components for Entity Framework Core tools.
Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 254M downloads
Microsoft SQL Server database provider for Entity Framework Core.
Microsoft.EntityFrameworkCore.Tools by Microsoft, 174M downloads
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
Microsoft.EntityFrameworkCore.InMemory by Microsoft, 129M downloads
In-memory database provider for Entity Framework Core (to be used for testing purposes).
Microsoft.EntityFrameworkCore.Sqlite by Microsoft, 85.1M downloads
SQLite database provider for Entity Framework Core.
Prerelease
Microsoft.EntityFrameworkCo
nuget.org
Version: 5.0.10
Install
Options
Show preview window
Description
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
Enables these commonly used commands:
Add-Migration
Drop-Database
Get-DbContext
Get-Migration
Remove-Migration
Scaffold-DbContext
Script-Migration
Update-Database
Version: 5.0.10
Author(s): Microsoft
License: Apache-2.0
Error List ...
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Code
Description
Project
File
Line
Suppression State
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DBTest' (1 of 1 project)
DBTest
Connected Services
Dependencies
Analyzers
Microsoft.AspNetCore.Analyzers
Microsoft.AspNetCore.Components.Analyzers
Microsoft.AspNetCore.Mvc.Analyzers
Microsoft.EntityFrameworkCore.Analyzers
Frameworks
Microsoft.AspNetCore.App
Microsoft.NETCore.App
Packages
Properties
wwwroot
Pages
appsettings.json
Program.cs
Startup.cs
Solution Explorer
Git Changes
Properties


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
DBTest
Debug
Any CPU
IIS Express
Live Share
Output
Build started...
1>------ Build started: Project: DBTes
1>DBTest -> C:\Users\ASUS\source\repos
1>DBTest -> C:\Users\ASUS\source\repos
========== Build: 1 succeeded, 0 faile
TestContext.cs
DBTest.csproj
NuGet: DBTest
DBTest: Overview
Student.cs
Browse
Installed
Updates 3
NuGet Package Manager: DBTest
Entity Framework Core
Include prerelease
Package source: nuget.org
Microsoft.EntityFrameworkCore.Analyzers by Microsoft, 461M downloads
CSharp Analyzers for Entity Framework Core.
7.0.0-rc.2.22472.11
Microsoft.EntityFrameworkCore.Design by Microsoft, 246M downloads
Shared design-time components for Entity Framework Core tools.
7.0.0-rc.2.22472.11
Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 254M downloads
Microsoft SQL Server database provider for Entity Framework Core.
5.0.10
7.0.0-rc.2.22472.11
Microsoft.EntityFrameworkCore.Tools by Microsoft, 174M downloads
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
5.0.10
7.0.0-rc.2.22472.11
Microsoft.EntityFrameworkCore.InMemory by Microsoft, 129M downloads
In-memory database provider for Entity Framework Core (to be used for testing purposes).
7.0.0-rc.2.22472.11
Prerelease
Microsoft.EntityFrameworkCo
nuget.org
Installed: 5.0.10
Uninstall
Version: 5.0.10
Update
Options
Show preview window
Description
Microsoft SQL Server database provider for Entity Framework Core.
Version: 5.0.10
Author(s): Microsoft
License: Apache-2.0
Date published: Tuesday, September 14, 2021 (9/14/2021)
Project URL: https://docs.microsoft.com/ef/core/
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DBTest' (1 of 1 project)
DBTest
Connected Services
Dependencies
Analyzers
Microsoft.AspNetCore.Analyzers
Microsoft.AspNetCore.Components.Analyzers
Microsoft.AspNetCore.Mvc.Analyzers
Microsoft.EntityFrameworkCore.Analyzers
Frameworks
Microsoft.AspNetCore.App
Microsoft.NETCore.App
Packages
Properties
wwwroot
Models
Student.cs
TestContext.cs
Pages
appsettings.json
Program.cs
Startup.cs

Friend.cs


Friend.cs
DBFirst
DBFirst.Models.Friend
Id
using System;
using System.Collections.Generic;
namespace DBFirst.Models
{
2 references
public partial class Friend
{
1 reference | 0 exceptions
public int Id { get; set; }
1 reference | 0 exceptions
public string Name { get; set; }
1 reference | 0 exceptions
public string Place { get; set; }
}
}
Properties
Solution Explorer
Search Solution Explorer (Ctrl+;)
css
images
js
lib
favicon.ico
Controllers
HomeController.cs
Models
Demo786Context.cs
ErrorViewModel.cs
Friend.cs
Views
Home
Shared
_ViewImports.cshtml
_ViewStart.cshtml
appsettings.json

Demo786Context.cs

Demo786Context.cs* | Friend.cs

DBFirst | DBFirst.Models.Demo786Context | OnModelCreating(ModelBu

```
7   public partial class Demo786Context : DbContext
8   {
        0 references | 0 exceptions
9       public Demo786Context()...
        0 references | 0 exceptions
12      public Demo786Context(DbContextOptions<Demo786Context> options)...
        0 references | 0 exceptions
16      public virtual DbSet<Friend> Friend { get; set; }
        0 references | 0 exceptions
17      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
18      {
19          if (!optionsBuilder.IsConfigured)
20          {
21  #warning To protect potentially sensitive information in your connection string, you should move it out of source code. See
22              optionsBuilder.UseSqlServer("Server=DESKTOP-DHRK7UB;Database=Demo786;Trusted_Connection=True;");
23          }
24      }
        0 references | 0 exceptions
25      protected override void OnModelCreating(ModelBuilder modelBuilder)
26      {
27          modelBuilder.Entity<Friend>(entity =>
28          {
29              entity.Property(e => e.Id).ValueGeneratedNever();
30
31              entity.Property(e => e.Name)
32                  .HasMaxLength(50)
33                  .IsUnicode(false);
34
35              entity.Property(e => e.Place).HasMaxLength(50);
36          });
37      }
38  }
```

Context class connects Models with DB

```
namespace MVC_SQL_Server.Models
{
    public class AppDBContext: DbContext //connecting to DB
    {
        public AppDBContext(DbContextOptions<AppDBContext> options): base(options) {
            //injecting service collection as constructor parameter
        }

        public DbSet<FriendModel> FriendModels { get; set; } //mirrors as table in DB

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
           //overridden to build entity model and initialized data

            modelBuilder.Entity<FriendModel>().HasData(
                new FriendModel { Id =1, Name="Faysal", Place="Sydney"},
                new FriendModel { Id =2, Name = "Sumon", Place = "Rajshahi" }
                );
        }
    }
}
```

LinQ

- An innovative concept in C#
- Encompasses a set of features (query) that lets you retrieve information from a data source.

- LINQ gives C# the ability to generate queries
  - For any compatible data source
  - Unique query syntax for any data source (RDBMS, Array, XML file. ADO.NET datasets)

- Two fundamentals steps are involved with LINQ
  - Query formulation, defines WHAT to retrieve from a data source
  - Executing the query, and obtains the results

LinQ Simple Example with Array

```
using System;
using System.Linq;  // LINQ Library namespace

class SimpQuery{
    static void Main(){
        int [] num={1, 2, -3 , -5, 3, 2};   //C# arrays are convertible to IEnumerable<T>

        var PosNum= from n in num   //PosNum is a query variable
                            where n> 0  // from range-variable in data source
                                select n;  // where boolean-expression

         foreach (int i in PosNum) Console.Write(i+ " "); // execute query

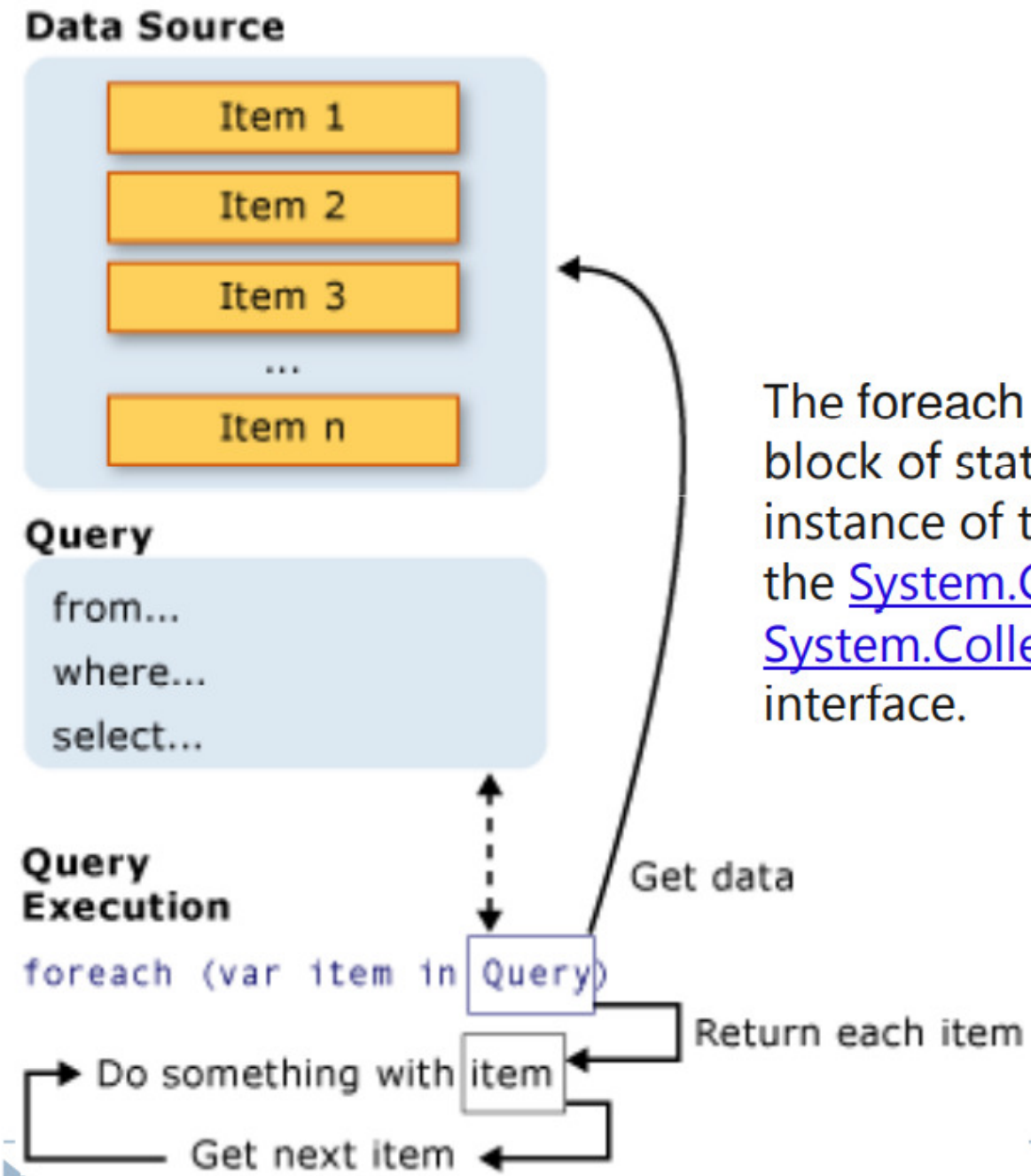
        Console.WriteLine();
    }
}                                                          int evenNumCount = PosNum.Count();
```

WHY var type PosNum?

NB: I) A Query can be executed more than once; num[2]=99;

NB: II) where n>0 && n<10 III) orderby n Iv) orderby n descending

https://www.tutorialspoint.com/compile_csharp_online.php



The foreach statement executes a statement or a block of statements for each element in an instance of the type that implements the System.Collections.IEnumerable or System.Collections.Generic.IEnumerable<T> interface.

LinQ in Database

```
private Demo786Context ctx = new Demo786Context();
        public IActionResult ShowAll()
        {
            IEnumerable<Friend> friends = (from s in ctx.Friend
                                           select s);

            return View(friends);
        }

        public IActionResult SearchFriend( )
        {
            var friend = (from s in ctx.Friend
                          where s.Name == "Shamim"
                          select s).FirstOrDefault<Friend>();

            return View(friend);
        }
```

Add and Remove APIs

```
public IActionResult Index()
{
    ctx.Add(new Participant { Id=4, Name="Maisa", Address="Norshingdi" });
    ctx.SaveChanges();


   /* IEnumerable<Participant> ps = (from s in ctx.Participants
                                     select s);*/


    ctx.Remove(ctx.Participants.Single(a => a.Id == 2));
    ctx.SaveChanges();

    IEnumerable<Participant> ps = (from s in ctx.Participants
                                     select s);


    return View(ps);
}
```

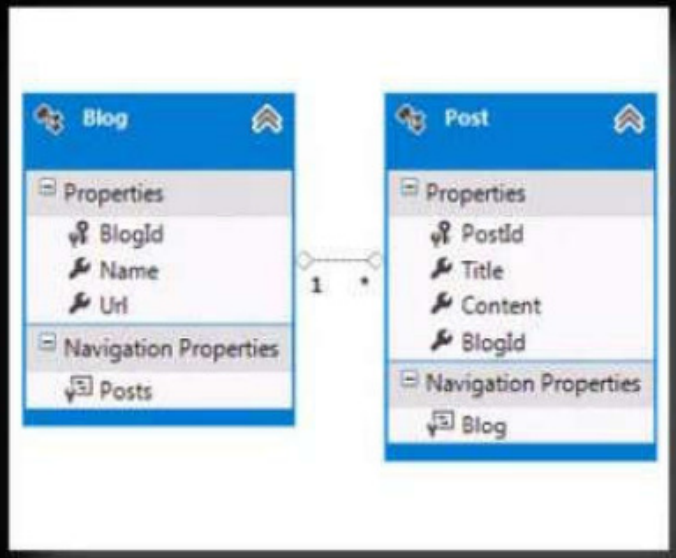
MCSE 541: Web Computing and Data Mining

Entity Framework Core in ASP.NET
Code First EF Convention

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

Creating a Model

- An EF model
  - stores the details about how application classes and properties map to database tables and columns.
- There are two main ways to create an EF model:
  - **Using Code First**:
    - Developer writes code to specify the model.
    - EF generates the models and mappings at runtime based on entity classes and additional model configuration provided by the developer.
  - **Using the EF Designer**:
    - Developer draws boxes and lines to specify the model.
    - The resulting model is stored as XML in a file with the EDMX (Entity data model XML) extension.
    - The application's domain objects are typically generated automatically from the conceptual model.




```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public List<Post> Posts { get; set; }
}

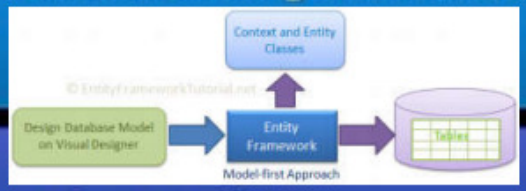
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

New Database

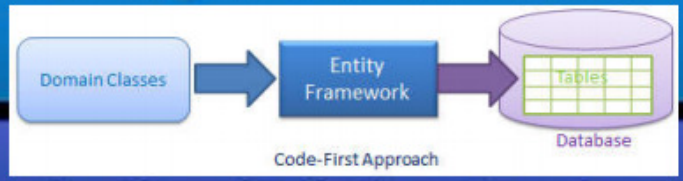
Model First

- Create model in designer
- Database created from model
- Classes auto-generated from model



Code First (New Database)

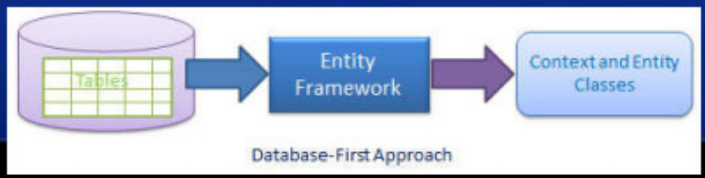
- Define classes & mapping in code
- Database created from model
- Use Migrations to evolve database



Existing Database

Database First

- Reverse engineer model in designer
- Classes auto-generated from model



Code First (Existing Database)

- Define classes & mapping in code
- Reverse engineer tools available

Code First Model-Simple1

Model Classes

```
1 reference
public class Student
{
    0 references | 0 exceptions
    public int StudentId { get; set; }
    0 references | 0 exceptions
    public string StudentName { get; set; }

    0 references | 0 exceptions
    public DateTime DateOfBirth { get; set; }

    0 references | 0 exceptions
    public byte[] Photo { get; set; }

    0 references | 0 exceptions
    public decimal Height { get; set; }

    0 references | 0 exceptions
    public float Weight { get; set; }

}
```

Domain Classes

Entity

- a class that maps to a database table.
- must be included as a DbSet<TEntity> type property in the DbContext class.
- EF API maps each entity to a table and each property of an entity to a column in the database.

Grade Model Class

```
1 reference
public class Grade
{
    0 references | 0 exceptions
    public int GradeId { get; set; }

    0 references | 0 exceptions
    public string GradeName { get; set; }

    0 references | 0 exceptions
    public string Section { get; set; }
}
```

Context Class in Entity Framework

- In order to use Entity Framework to query, insert, update, and delete data using .NET objects,
  - first need to Create a Model
  - maps the entities and relationships that are defined in the model to tables in a database.

- After model class, the primary class your application interacts with is context class.
  - It represents a session with the underlying database to perform CRUD (Create, Read, Update, Delete) operations.
  - Derives from System.Data.Entity.DbContext.
  - It is also used to configure domain classes, database related mappings, change tracking settings, caching, transaction etc.

- Model Context Class:

```
3 references
public class StudentContext:DbContext
{
    0 references | 0 exceptions
    public StudentContext(DbContextOptions<StudentContext> options)
        : base(options)
    {

    }
    0 references | 0 exceptions
    public DbSet<Student> Students { get; set; }
    0 references | 0 exceptions
    public DbSet<Grade> Grades { get; set; }


}
```

Configuration in the Connection String

AppDBContext.cs | Program.cs | Startup.cs | UpdateFriend.cshtml | SearchFriend.cshtml | RemoveFriend.cshtml | AddFriend.cshtml

Schema: https://json.schemastore.org/appsettings

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "MyKey":  "Value of myKey from appsettings.json",
  "ConnectionStrings": {
    "FriendDBConnection":  "server=(localdb)\\MSSQLLocalDB; database=FriendDB;Trusted_Connection=true"
  }
}
```

Configuration in the Startup class

- Working with SQL Server LocalDB

StudentContext

Use AddDb contextPool type AppDBContext method on services collection to add the DB context and the EF services at the beginning of the ConfigureService method.
Call the UseSqlServer method on the options action in its constructor to specify that you want to use SQL Server database provider. UseSqlServer @ Microsoft.EntityFrameworkCore namespace.
At first the connection string must be read.

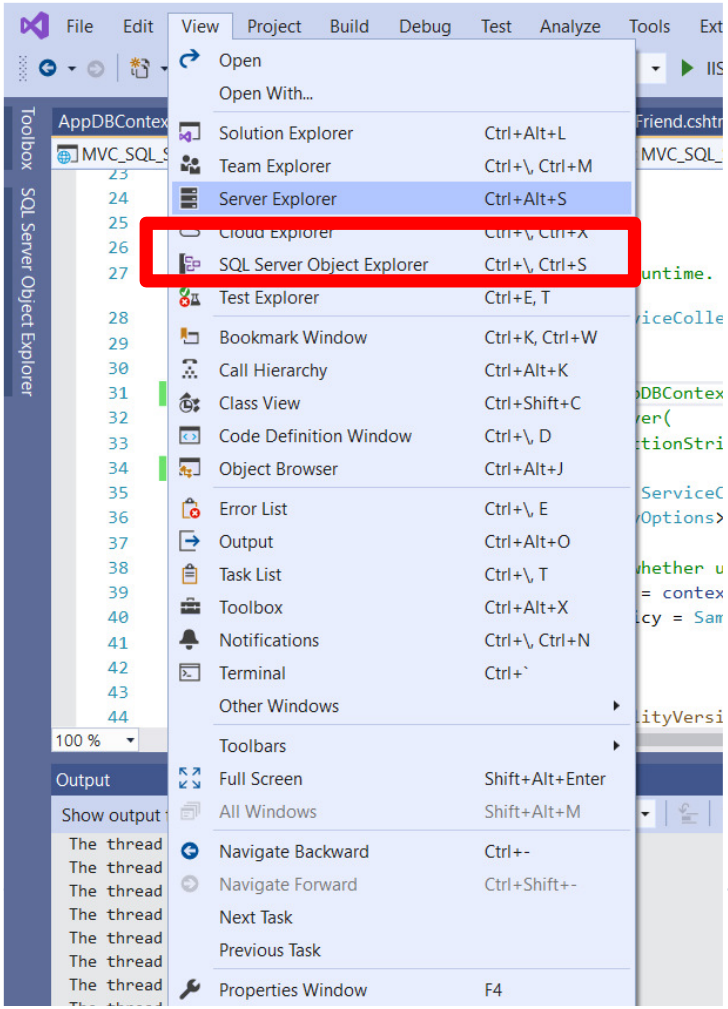
```
0 references | 0 exceptions
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

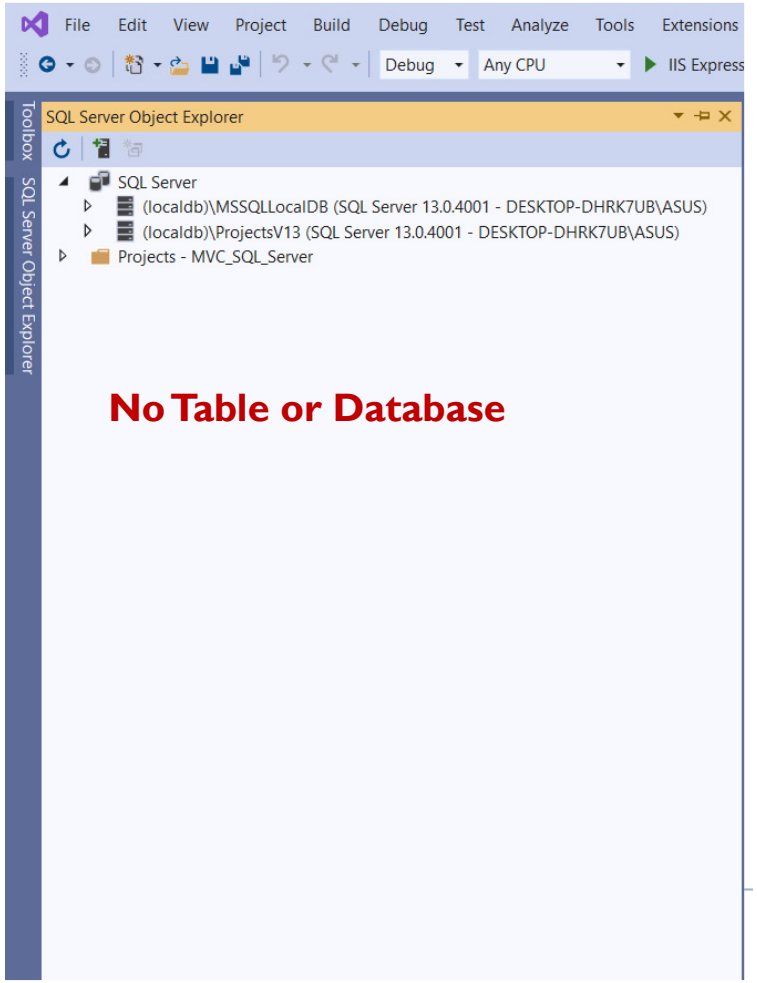
// This method gets called by the runtime. Use this method to add services to the container.
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{

    /*services.AddDbContextPool<AppDBContext>(
        options=>options.UseSqlServer(
            Configuration.GetConnectionString("FriendDBConnection")));
    */
    services.AddSingleton<IFriend, ServiceClass>();
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });
```

SQL Server Express LocalDB

- LocalDB is a lightweight version of the SQL Server Express Database Engine that is targeted for program development. LocalDB starts on demand and runs in user mode, so there is no complex configuration. By default, LocalDB database creates "*.mdf" files in the *C:/Users/<user>* directory.





Adding Migration

Migration is a way to keep the database schema in sync with the EF Core model by preserving data.

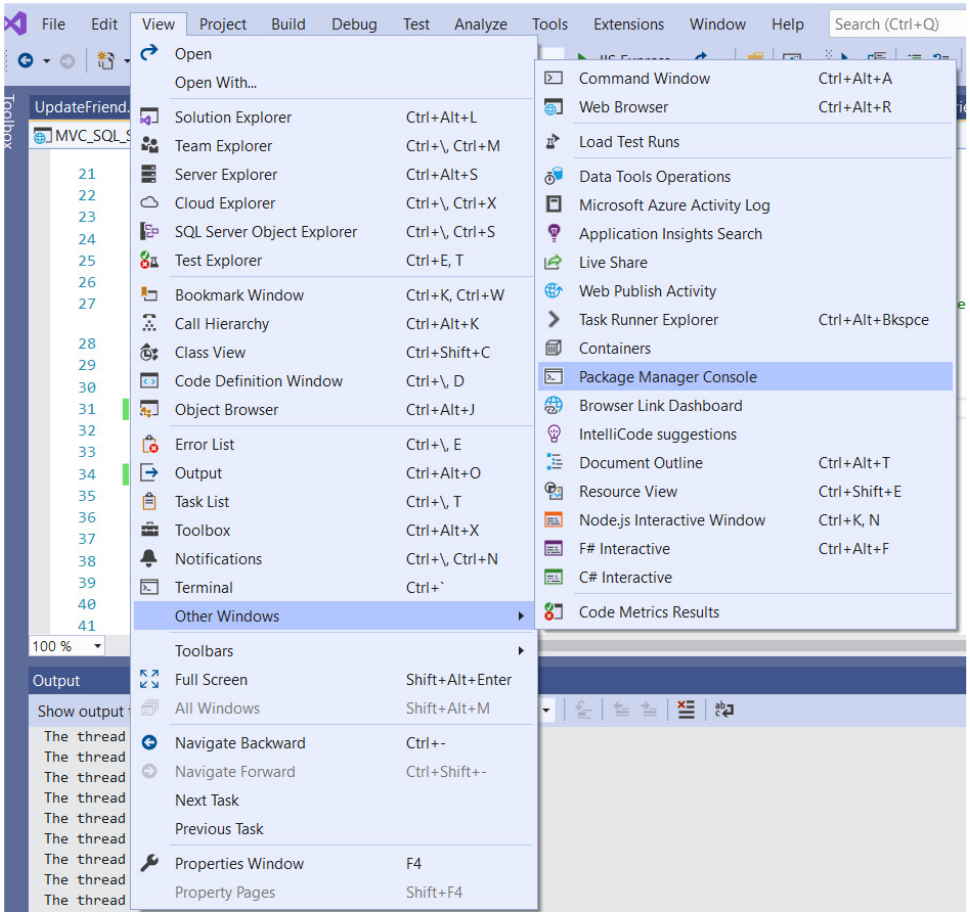
As per the above figure, EF Core API builds the EF Core model from the domain (entity) classes and EF Core migrations will create or update the database schema based on the EF Core model. Whenever you change the domain classes, you need to run migration to keep the database schema up to date.

EF Core migrations are a set of commands which you can execute in NuGet Package Manager Console or in dotnet Command Line Interface (CLI).

The following table lists important migration commands in EF Core.

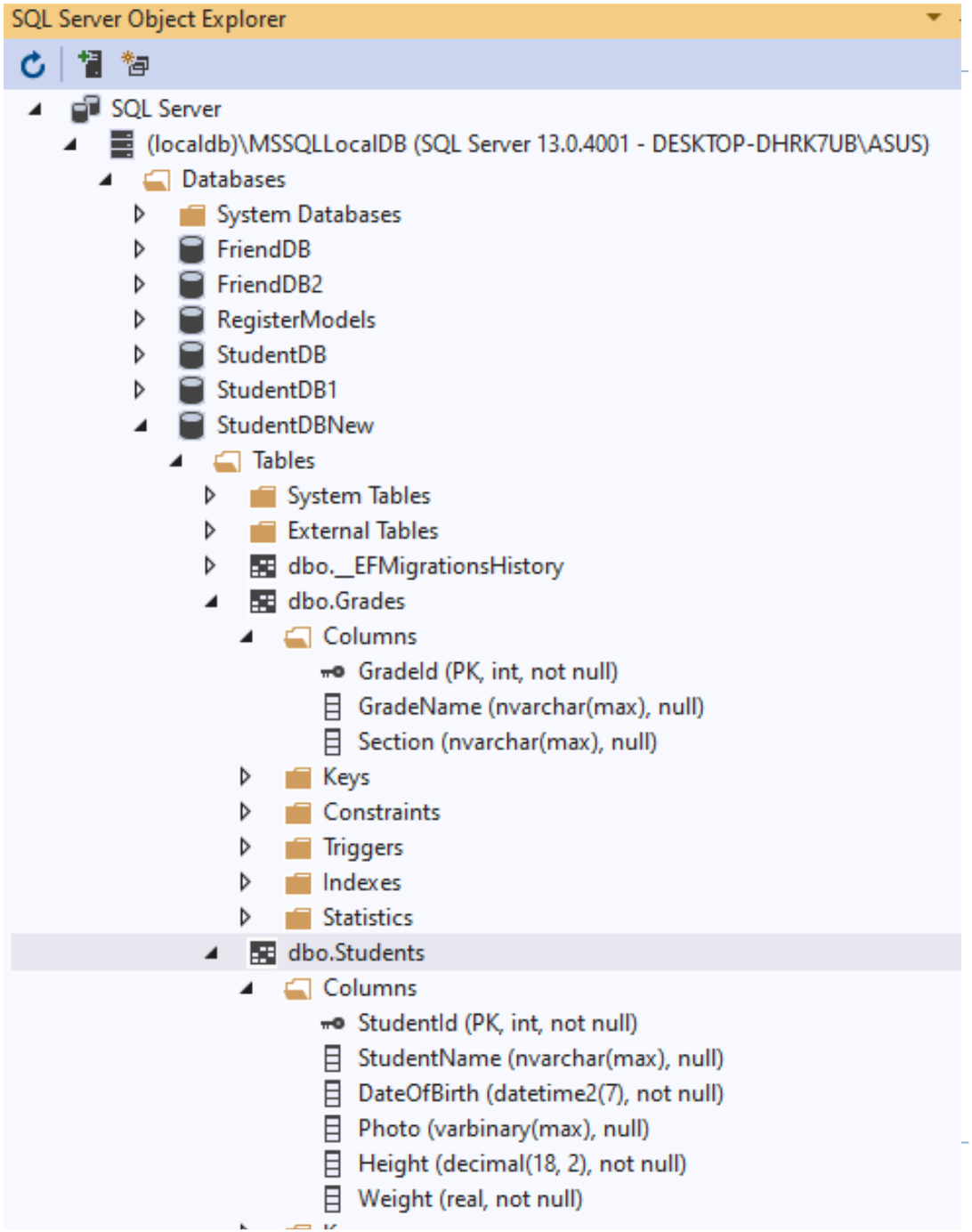
PMC Command	dotnet CLI command	Usage
add-migration <migration name>	Add <migration name>	Creates a migration by adding a migration snapshot.
Remove-migration	Remove	Removes the last migration snapshot.
Update-database	Update	Updates the database schema based on the last migration snapshot.
Script-migration	Script	Generates a SQL script using all the migration snapshots.

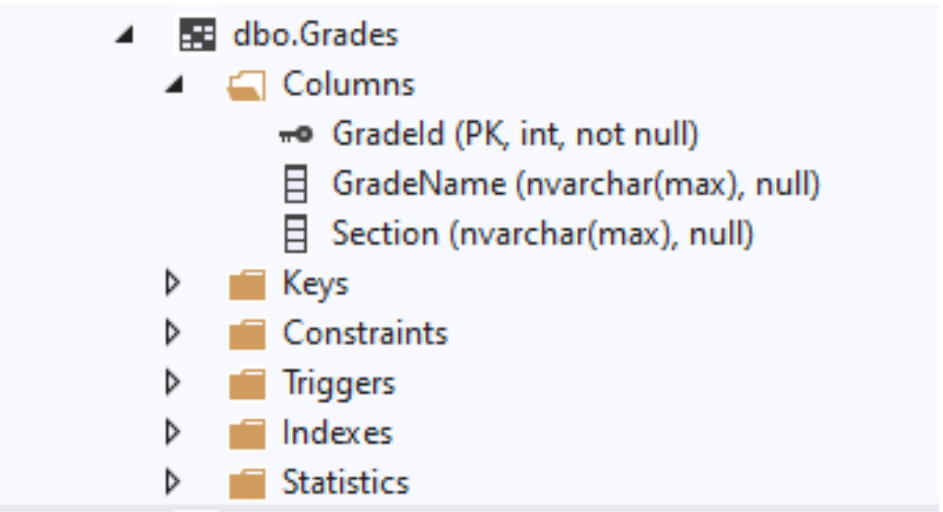
Adding Migration

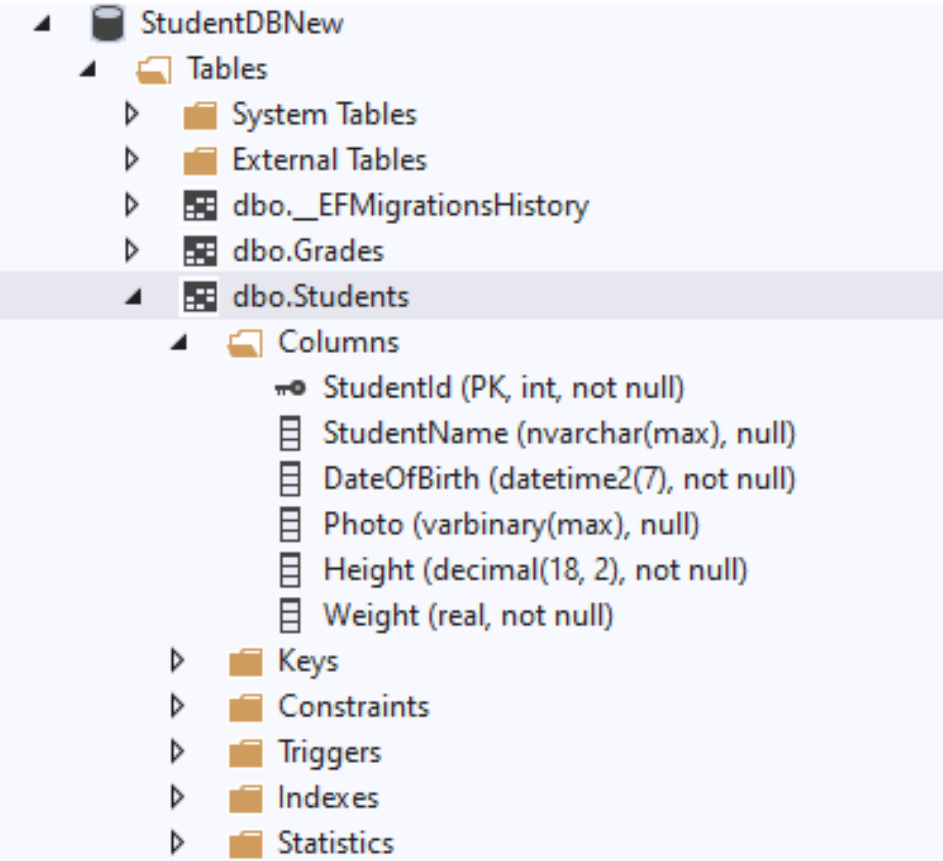


- Add-Migration Initialize
  - A folder will appear
  - Current and future migration will appear there.
- Update-Database

SQL Express Server







Code First Model-Simple1

Model Classes to Database Tables

```
1 reference
public class Student
{
    0 references | 0 exceptions
    public int StudentId { get; set; }
    0 references | 0 exceptions
    public string StudentName { get; set; }

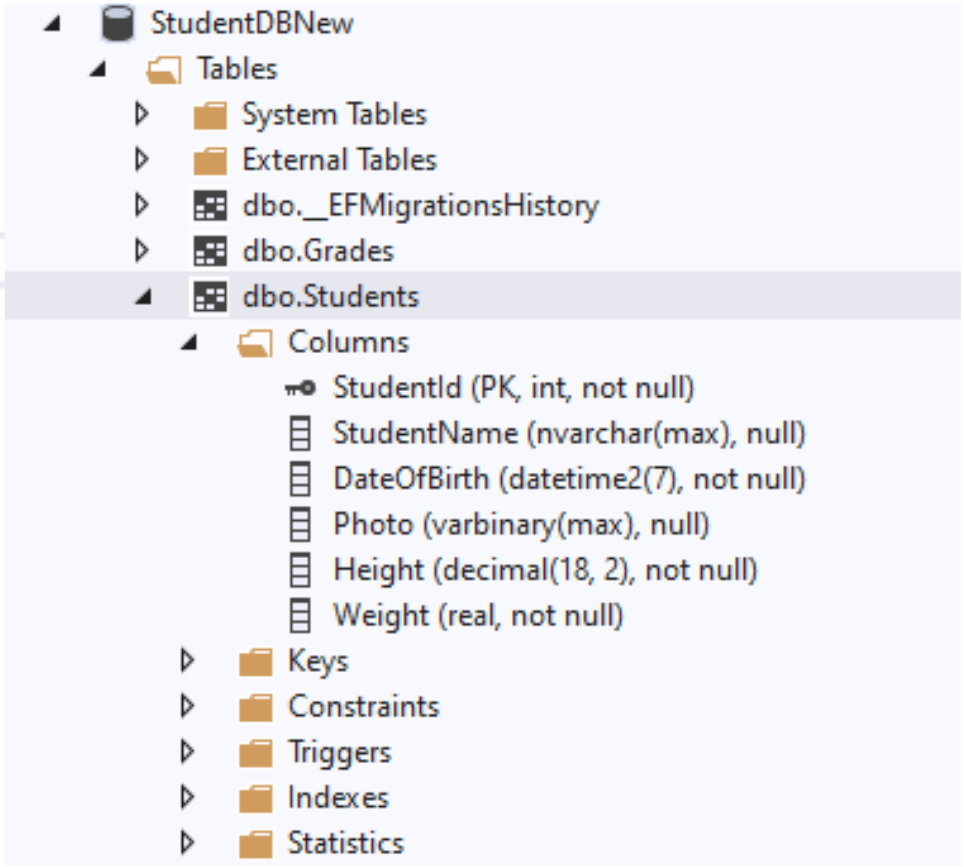
    0 references | 0 exceptions
    public DateTime DateOfBirth { get; set; }

    0 references | 0 exceptions
    public byte[] Photo { get; set; }

    0 references | 0 exceptions
    public decimal Height { get; set; }

    0 references | 0 exceptions
    public float Weight { get; set; }

}
```



Code First Convention

Default Convention For	Description
Schema	By default, EF creates all the DB objects into the **dbo** schema.
Table Name	<Entity Class Name> + 's' EF will create a DB table with the entity class name suffixed by 's' e.g. `Student` domain class (entity) would map to the `Students` table.
Primary key Name	1) Id 2) <Entity Class Name> + "Id" (case insensitive) EF will create a primary key column for the property named Id or <Entity Class Name> + "Id" (case insensitive).
Foreign key property Name	By default EF will look for the foreign key property with the same name as the principal entity primary key name. If the foreign key property does not exist, then EF will create an FK column in the Db table with <Dependent Navigation Property Name> + "_" + <Principal Entity Primary Key Property Name> e.g. EF will create `Grade_GradeId` foreign key column in the `Students` table if the `Student` entity does not contain foreignkey property for `Grade`.

Null column	EF creates a null column for all reference type properties and nullable primitive properties e.g. string, Nullable<int>, Student, Grade (all class type properties)
Not Null Column	EF creates NotNull columns for Primary Key properties and non-nullable value type properties e.g. int, float, decimal, datetime etc.
DB Columns order	EF will create DB columns in the same order like the properties in an entity class. However, primary key columns would be moved first.
Properties mapping to DB	By default, all properties will map to the database. Use the `[NotMapped]` attribute to exclude property or class from DB mapping.
Cascade delete	Enabled by default for all types of relationships.

Entity Framework Core (EF Core) represents relationships using foreign keys. An entity with a foreign key is the child or dependent entity in the relationship. This entity's foreign key value must match the primary key value (or an alternate key value) of the related principal/parent entity.

If the principal/parent entity is deleted, then the foreign key values of the dependents/children will no longer match the primary or alternate key of *any* principal/parent. This is an invalid state, and will cause a referential constraint violation in most databases.

There are two options to avoid this referential constraint violation:

1. Set the FK values to null
2. Also delete the dependent/child entities

The first option is only valid for optional relationships where the foreign key property (and the database column to which it is mapped) must be nullable.

The second option is valid for any kind of relationship and is known as "cascade delete".

Domain Classes

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[]  Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public Grade Grade { get; set; }
}
```

```
public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

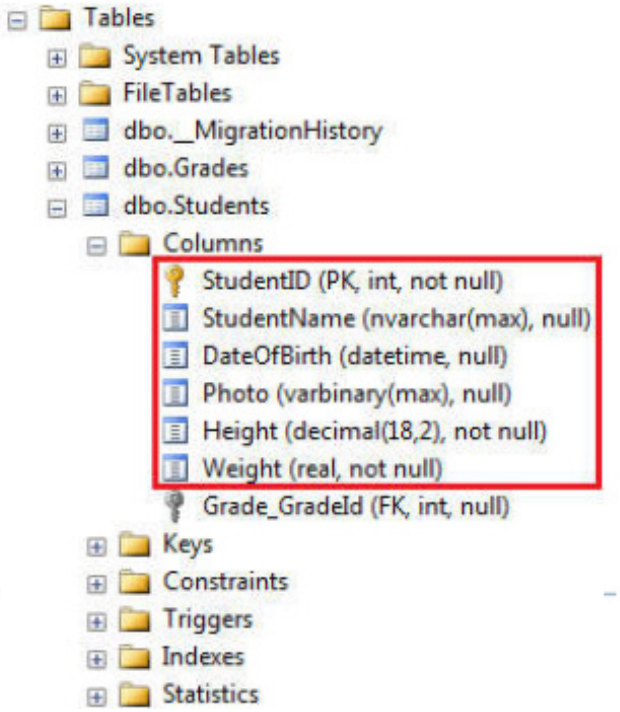
Entity

- a class that maps to a database table.
- must be included as a DbSet<TEntity> type property in the DbContext class.
- EF API maps each entity to a table and each property of an entity to a column in the database.

Entity can include two types of properties: Scalar Properties and Navigation Properties.

- Scalar Property
  - The primitive type properties are called scalar properties.
  - Each scalar property maps to a column in the database table which stores an actual data.
  - For example, StudentID, StudentName, DateOfBirth, Photo, Height, Weight are the scalar properties in the Student entity class.

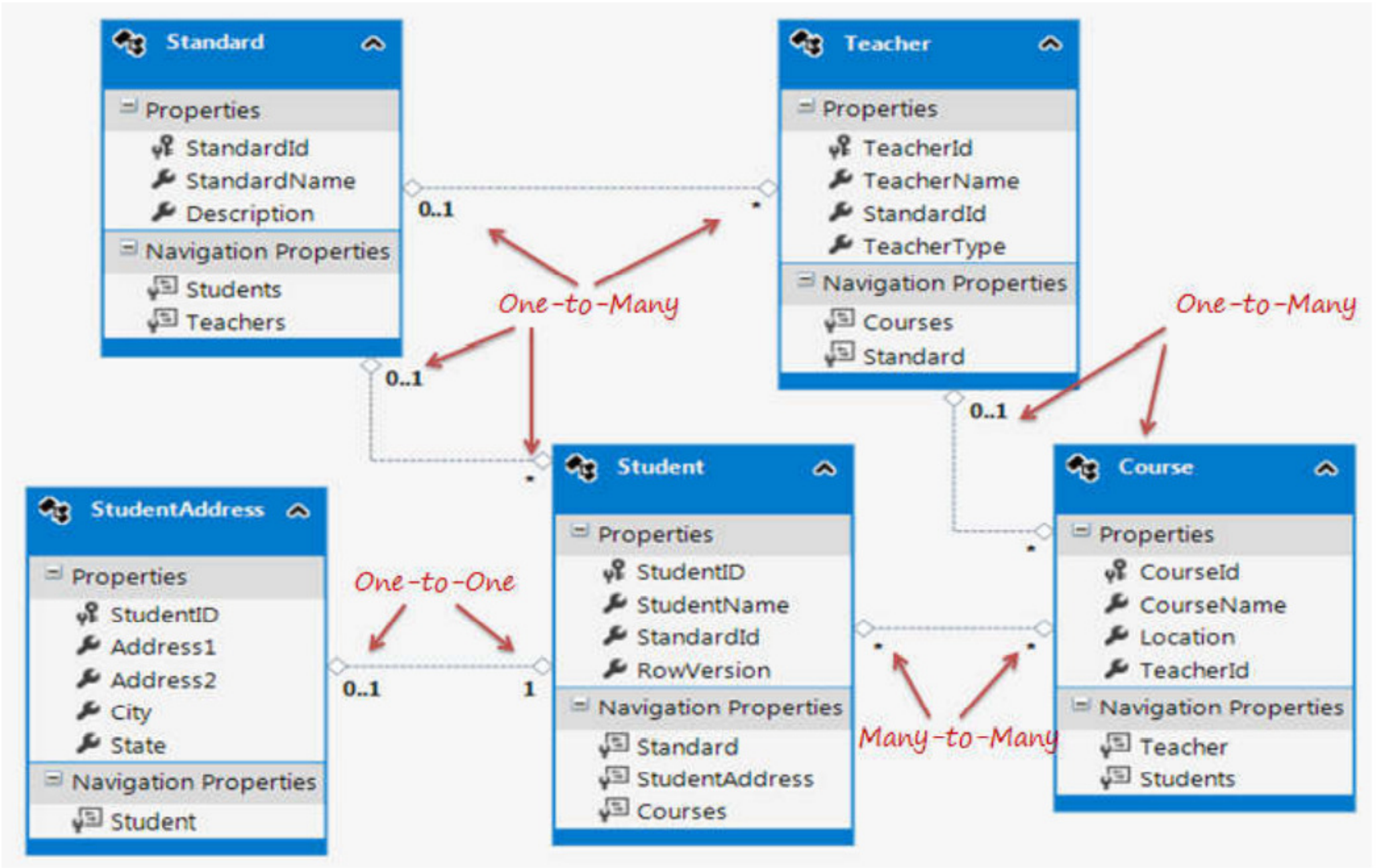
```
public class Student {
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set;}
    public byte[] Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }
   // public Grade Grade { get; set; }
}
```



Introduction to Relationships

- Relational databases store data based on the relation between items of data.
- A key benefit to taking a relational view of data is to reduce duplication.
- In a relational database, each *entity* such as the person, the school, the place of work is stored in separate tables and unique instances if the entity are identified by a Primary Key value.
- Relationships or associations between entities are defined in a database by the existence of Foreign Keys.

Entity Relationship (association) Convention



Navigation Property

- The navigation property represents a relationship to another entity.
- There are two types of navigation properties: Reference Navigation and Collection Navigation

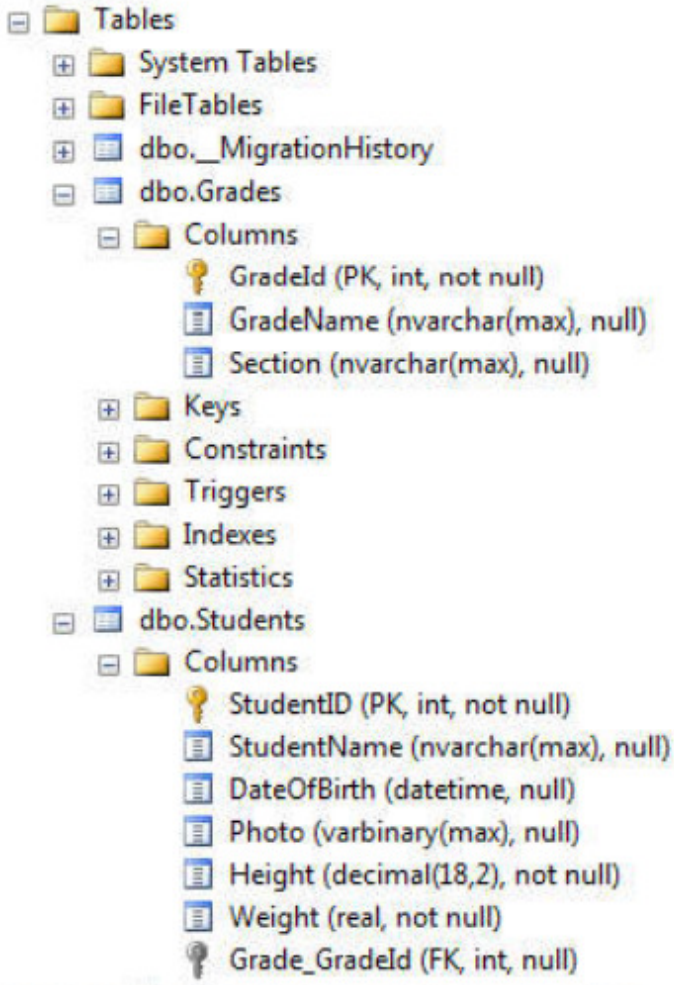
```
public class Student {
    //public int StudentID { get; set; }
    //public string StudentName { get; set; }
    //public DateTime? DateOfBirth { get; set;}
    //public byte[] Photo { get; set; }
    //public decimal Height { get; set; }
    //public float Weight { get; set; }

    public Grade Grade { get; set; }
}
```

Reference Navigation Property

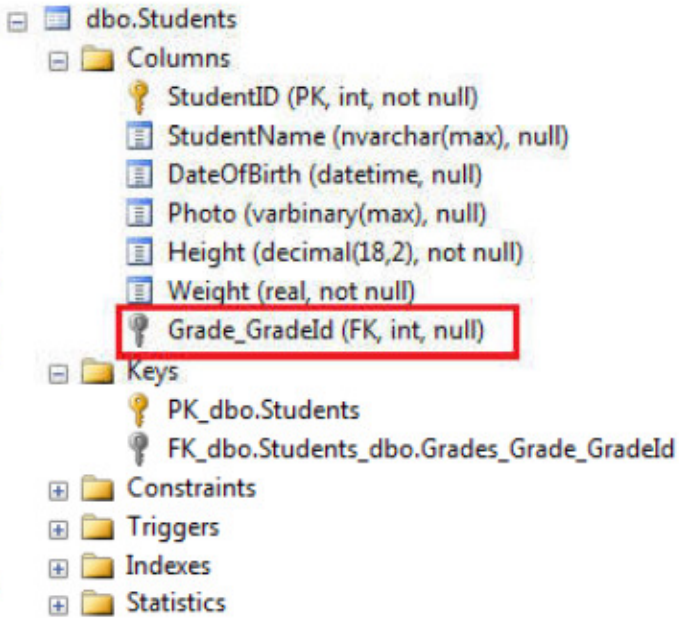
- If an entity includes a property of another entity type, it is called a Reference Navigation Property.
- It points to a single entity and represents multiplicity of one (1) in the entity relationships.
- EF API will create a ForeignKey column in the table for the navigation properties that points to a PrimaryKey of another table in the database.
- For example, Grade are reference navigation properties in the Student entity class.

```
public class Student {
…
    public Grade Grade { get; set; }
}
```



Collection Navigation Property

- If an entity includes a property of generic collection of an entity type, it is called a collection navigation property.
- It represents multiplicity of many (*).
- EF API does not create any column for the collection navigation property in the related table of an entity, but it creates a column in the table of an entity of generic collection.
- For example, the Grade entity contains a generic collection navigation property ICollection<Student>. Here, the Student entity is specified as generic type, so EF API will create a column Grade_GradeId in the Students table in the database.




```
public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

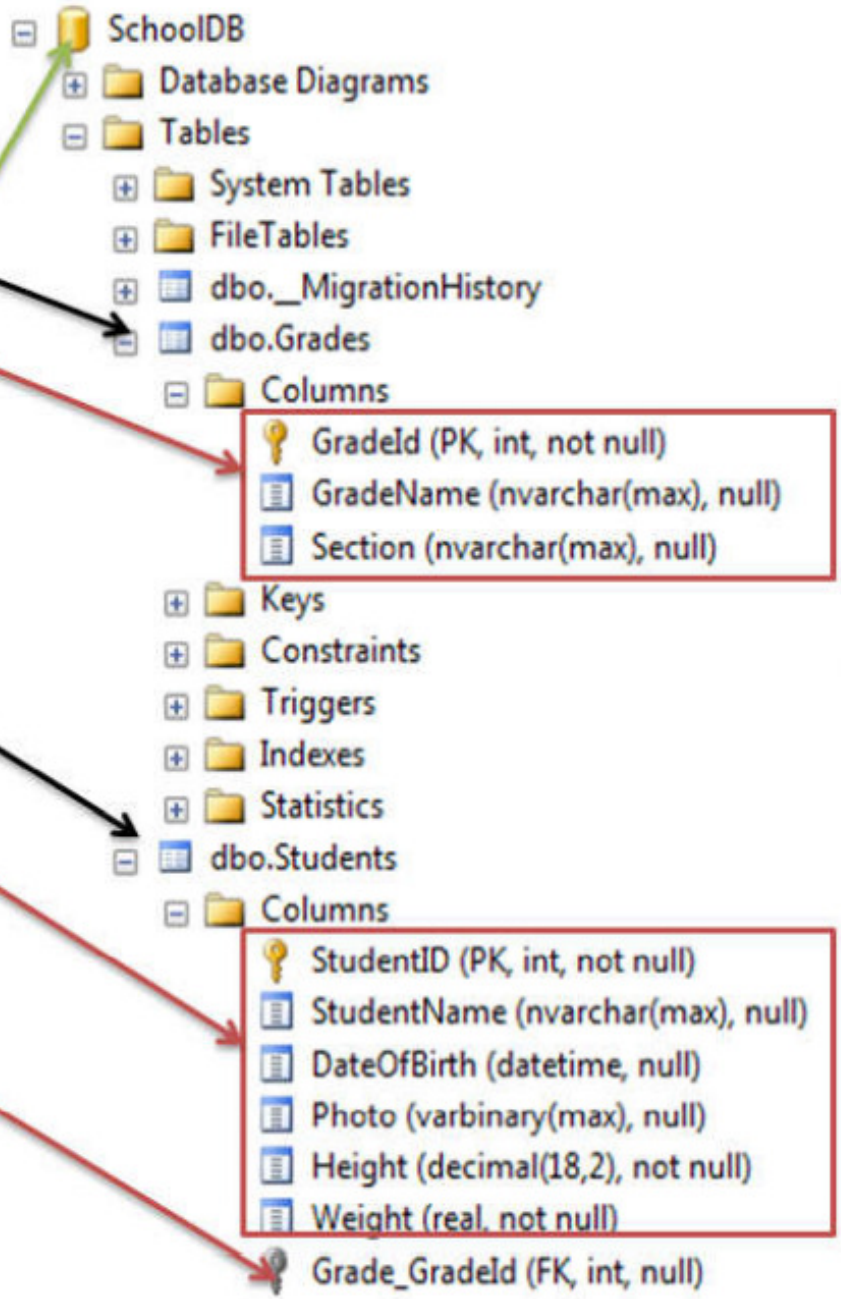
```
public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[] Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public Grade Grade { get; set; }
}
public class SchoolContext : DbContext
{
    public SchoolContext() : base("SchoolDB")
    {

    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Grade> Grades { get; set; }
}
```



Convention 4

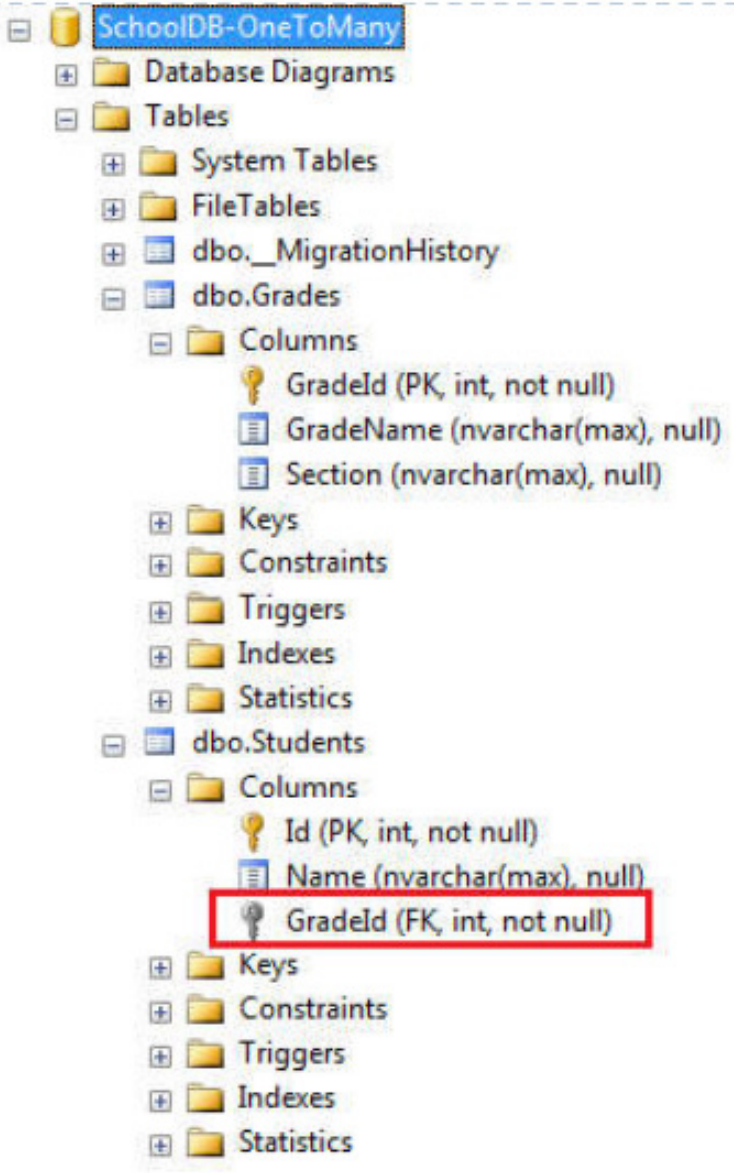
```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int GradeId { get; set; }
    public Grade Grade { get; set; }
}

public class Grade
{

    public int GradeId { get; set; }
    public string GradeName { get; set; }

    public ICollection<Student> Student { get; set; }
}
```



public int? GradeId { get; set; }

GradeId is nullable integer, and create a null foreign key.

One-to-Zero-or-One relationships

- A student can have only one or zero addresses.

```
public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }

    public virtual StudentAddress Address { get; set; }
}

public class StudentAddress
{
    public int StudentAddressId { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public int Zipcode { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public virtual Student Student { get; set; }
}
```

A one to one (or more usually a one to zero or one) relationship exists when only one row of data in the principal table is linked to zero or one row in a dependent table.

A one-to-zero-or-one relationship happens when a primary key of one table becomes PK & FK in another table in a relational database such as SQL Server.

One-to-Zero-or-One Relationship using Data Annotation Attributes

```
public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }

    public virtual StudentAddress Address { get; set; }
}

public class StudentAddress
{
    [ForeignKey("Student")]
    public int StudentAddressId { get; set; }

    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public int Zipcode { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public virtual Student Student { get; set; }
}
```

By convention StudentId and StudentAddressId are PK

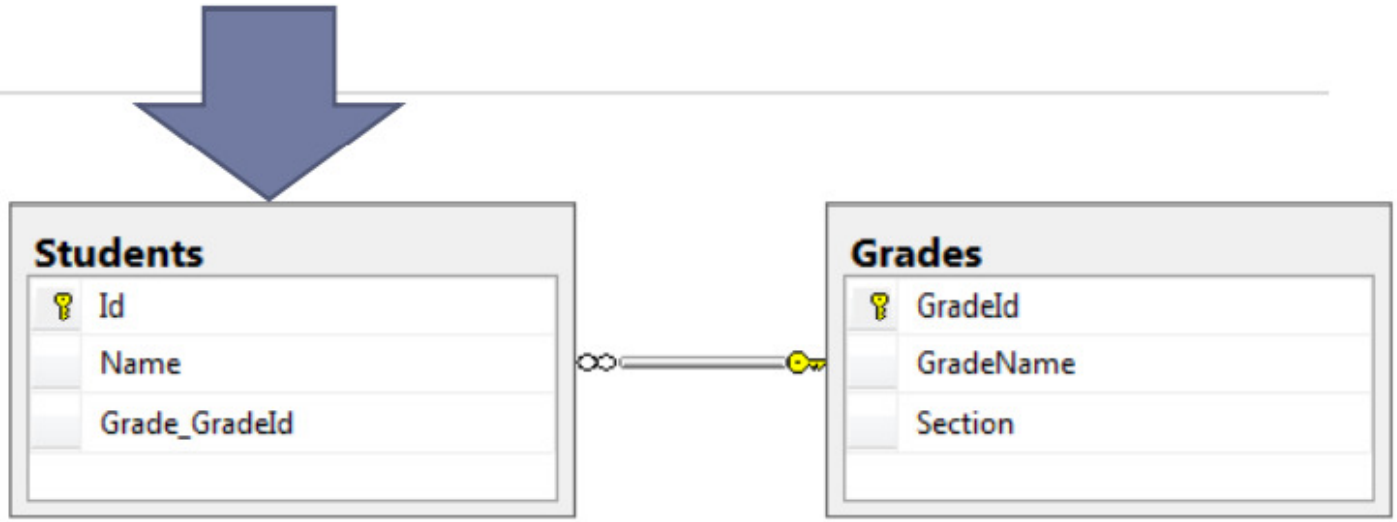
But FK we define Data Annotation Attributes

Student includes the StudentAddress navigation property and StudentAddress includes the Student navigation property. With the one-to-zero-or-one relationship, a Student can be saved without StudentAddress but the StudentAddress entity cannot be saved without the Student entity. EF will throw an exception if you try to save the StudentAddress entity without the Student entity.

One-to-Many relationships

```
public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }
}

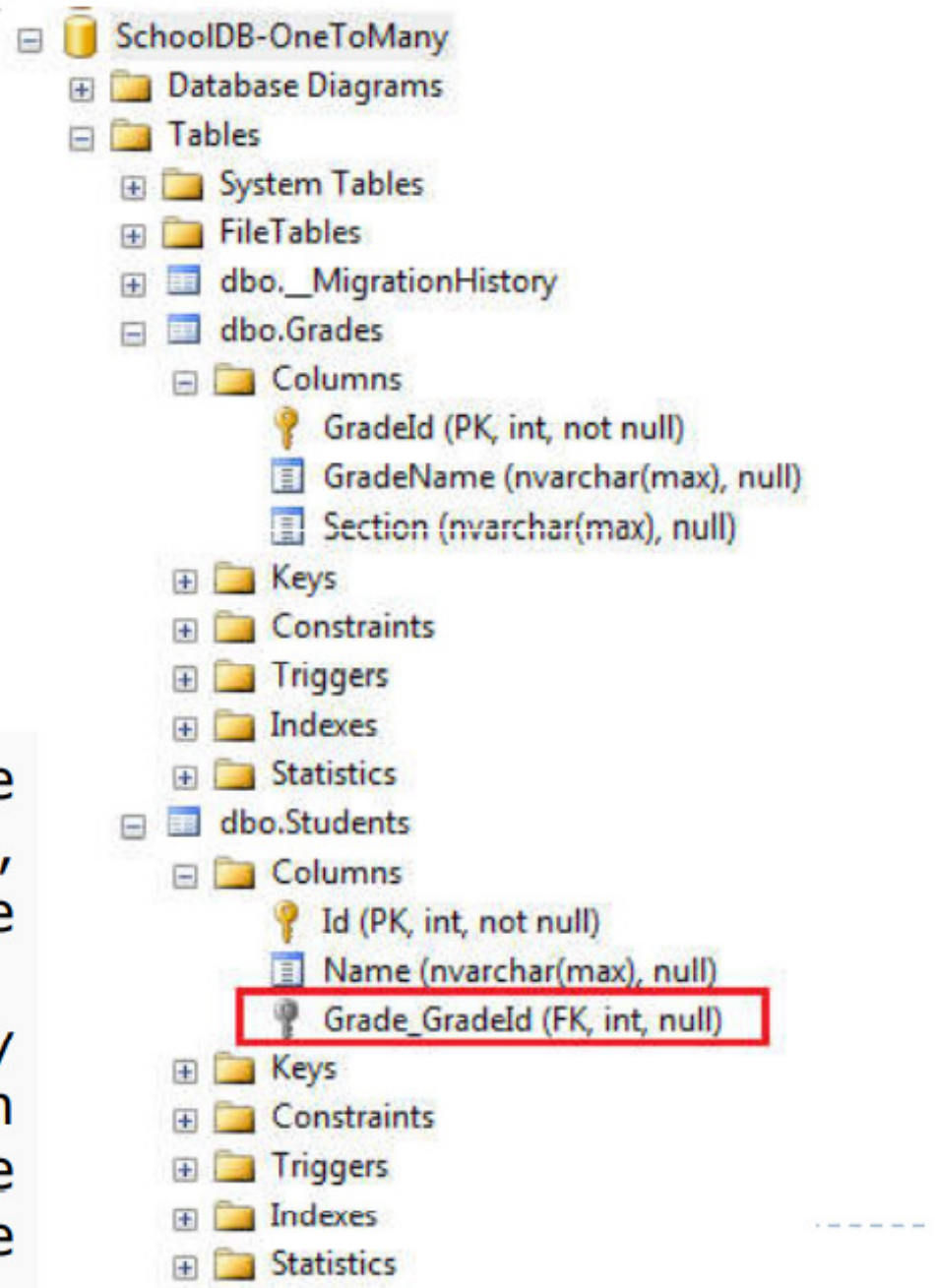
public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }
}
```



Convention 1

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public Grade Grade { get; set; }
}

public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }
}
```



Student class includes a reference navigation property of Grade class. So, there can be many students in a single grade.

This will result in a one-to-many relationship between the Students and Grades table in the database, where the Students table includes foreign key Grade_GradeId

Convention 2

```
public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }
}

public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

- Same result in the database as convention 1.

Convention 4

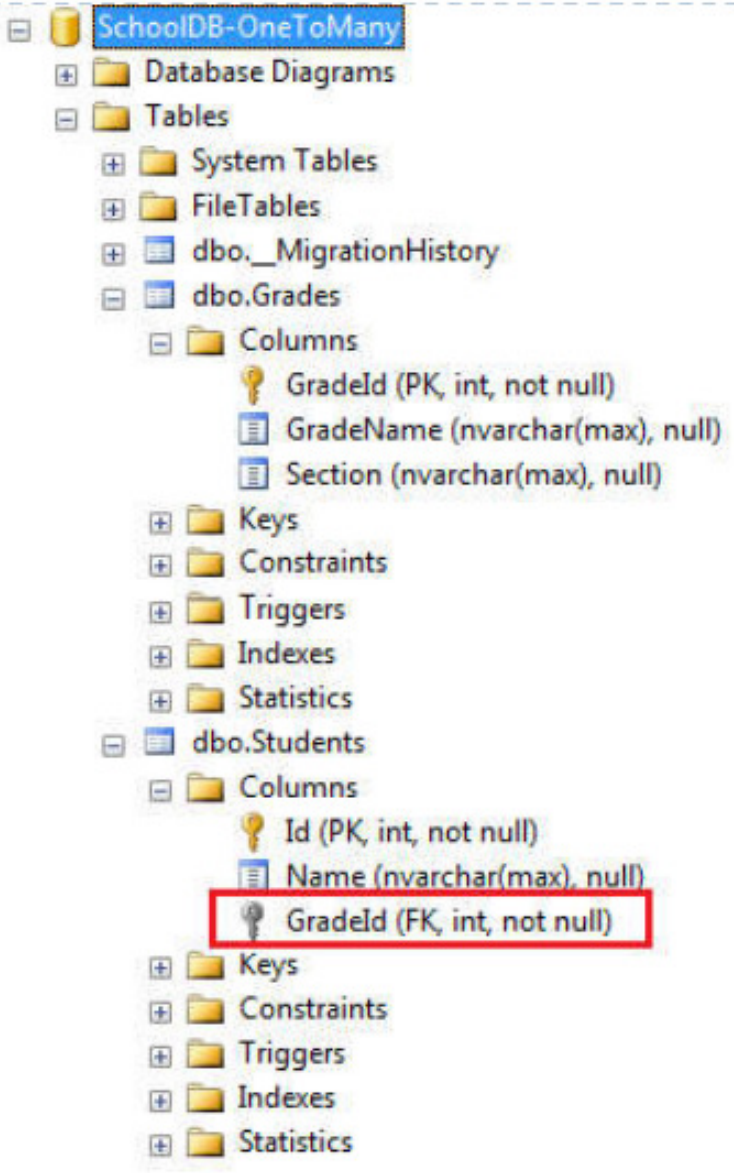
```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int GradeId { get; set; }
    public Grade Grade { get; set; }
}

public class Grade
{

    public int GradeId { get; set; }
    public string GradeName { get; set; }

    public ICollection<Student> Student { get; set; }
}
```



public int? GradeId { get; set; }

GradeId is nullable integer, and create a null foreign key.

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

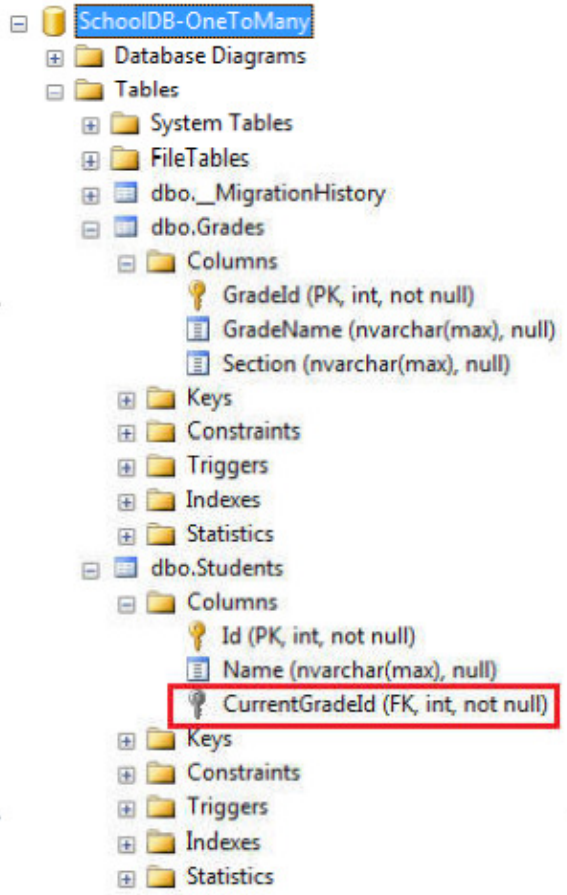
    public int CurrentGradeId { get; set; }
    public Grade CurrentGrade { get; set; }
}

public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

```
public class SchoolContext : DbContext
{
    public DbSet<Student> Students { get; set; }
    public DbSet<Grade> Grades { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // configures one-to-many relationship
        modelBuilder.Entity<Student>()
            .HasRequired<Grade>(s => s.CurrentGrade)
            .WithMany(g => g.Students)
            .HasForeignKey<int>(s => s.CurrentGradeId);          }
    }
}
```




```
modelBuilder.Entity<Student>()
        .HasOne(s => s.CurrentGrade)
        .WithMany(g => g.Students)
        .HasForeignKey(s => s.CurrentGradeId);
```

Many-to-Many relationship

```
public class Student
{
    public Student()
    {
        this.Courses = new HashSet<Course>();
    }

    public int StudentId { get; set; }
    [Required]
    public string StudentName { get; set; }

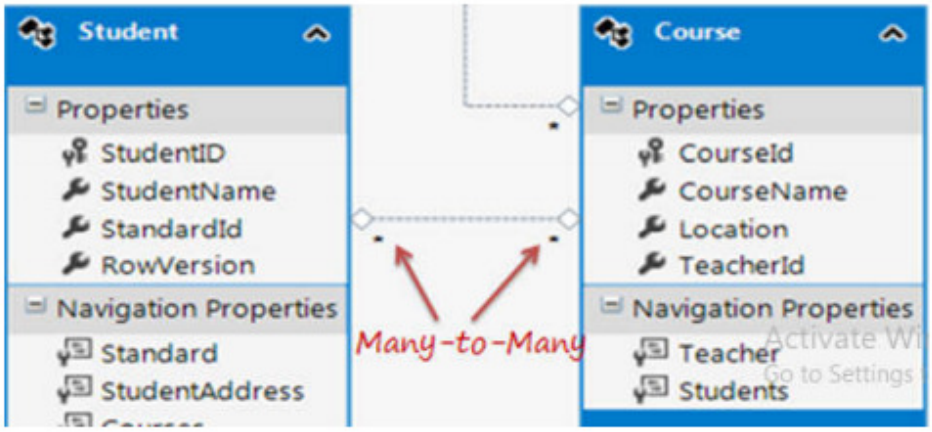
    public virtual ICollection<Course> Courses { get; set; }
}

public class Course
{
    public Course()
    {
        this.Students = new HashSet<Student>();
    }

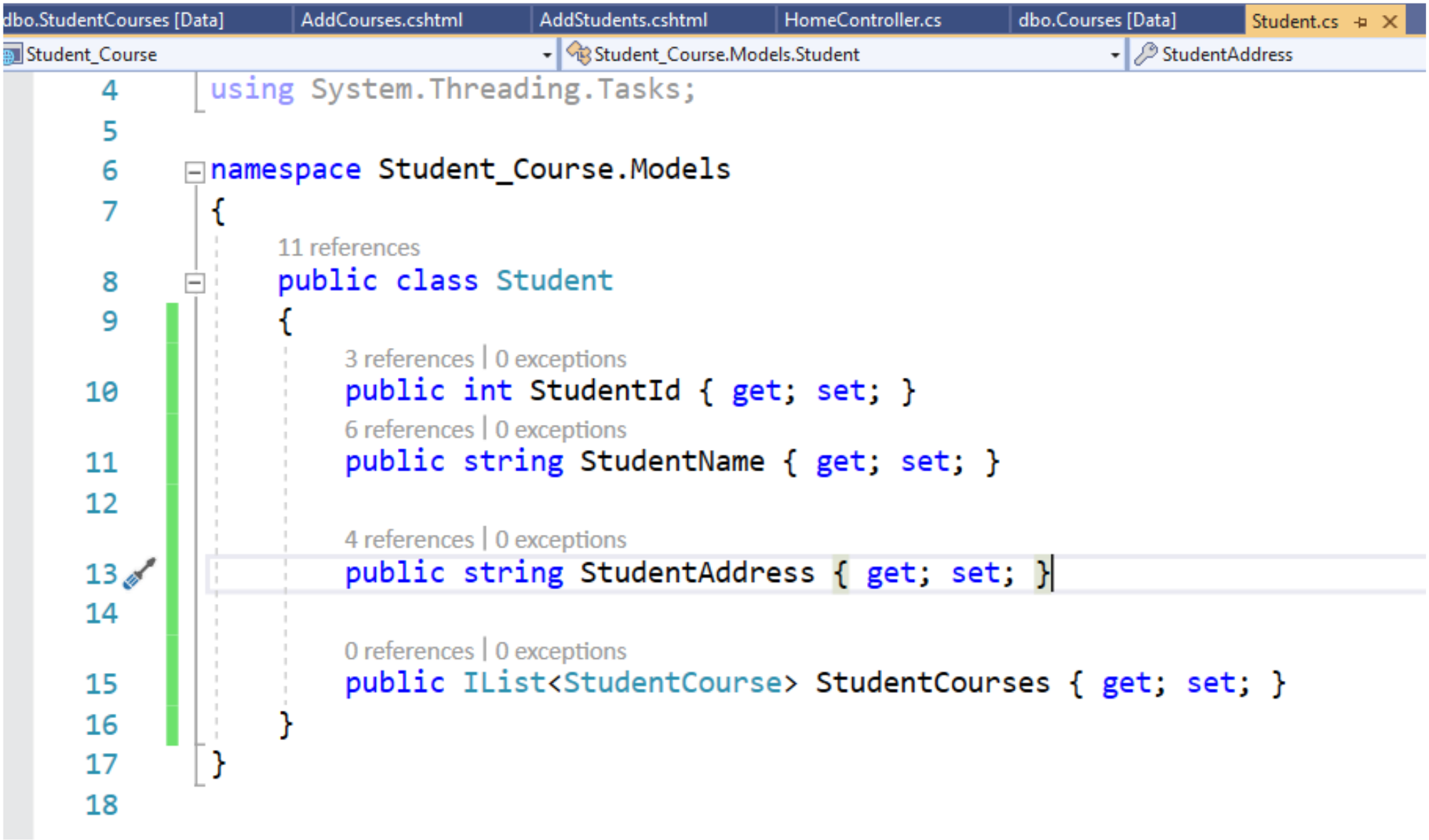
    public int CourseId { get; set; }
    public string CourseName { get; set; }

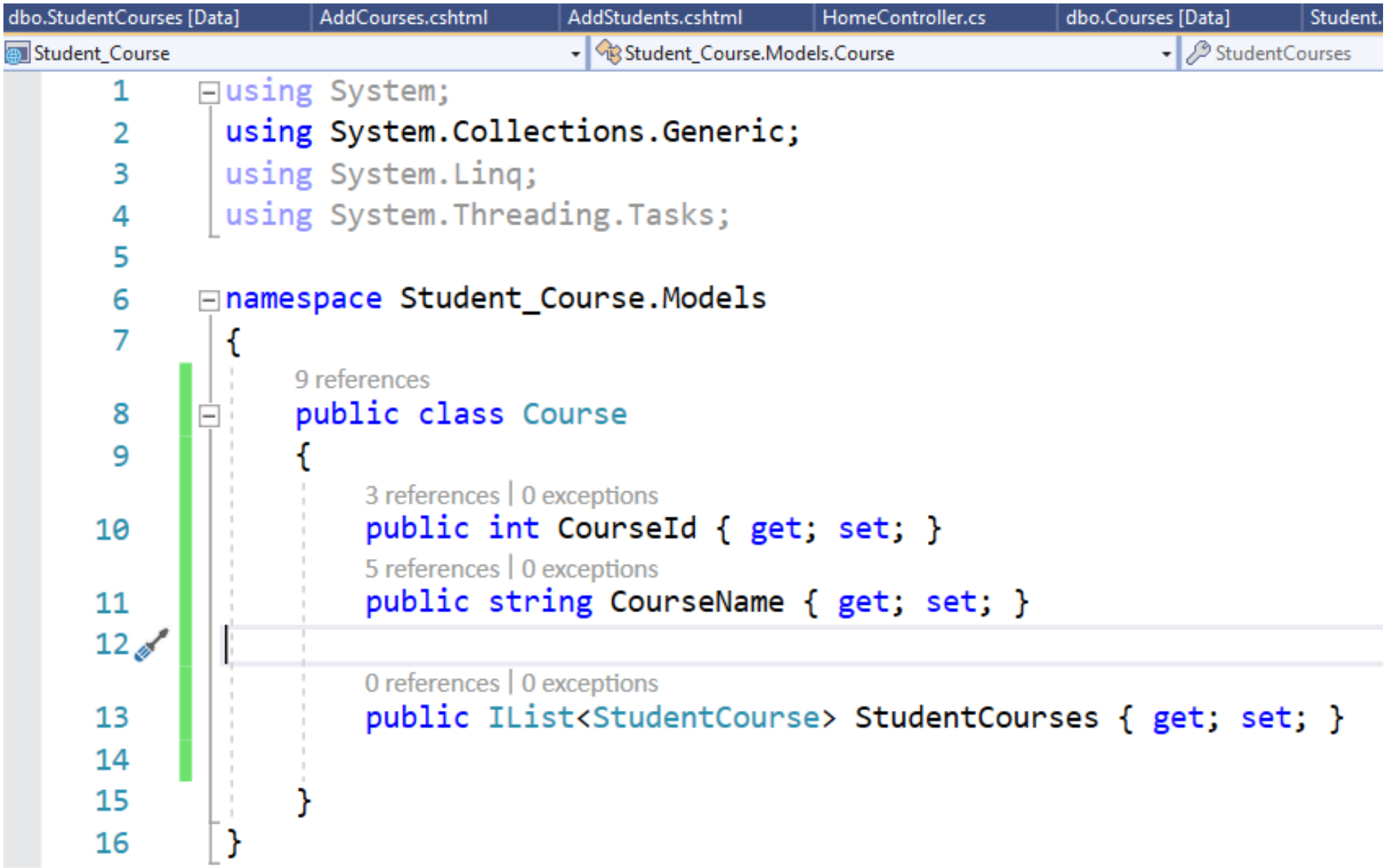
    public virtual ICollection<Student> Students { get; set; }
}
```

Student class should have a collection navigation property of Course type, and the Course class should have a collection navigation property of Student type to create a many-to-many relationship between them without any configuration.



Student can join multiple courses and multiple students can join one Course.


dbo.StudentCourses [Data]
AddCourses.cshtml
AddStudents.cshtml
HomeController.cs
dbo.Courses [Data]
Student.cs
Student_Course
Student_Course.Models.Student
StudentAddress
using System.Threading.Tasks;
namespace Student_Course.Models
{
11 references
public class Student
{
3 references | 0 exceptions
public int StudentId { get; set; }
6 references | 0 exceptions
public string StudentName { get; set; }
4 references | 0 exceptions
public string StudentAddress { get; set; }
0 references | 0 exceptions
public IList<StudentCourse> StudentCourses { get; set; }
}
}


dbo.StudentCourses [Data]
AddCourses.cshtml
AddStudents.cshtml
HomeController.cs
dbo.Courses [Data]
Student_Course
Student_Course.Models.Course
StudentCourses
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
namespace Student_Course.Models
{
9 references
public class Course
{
3 references | 0 exceptions
public int CourseId { get; set; }
5 references | 0 exceptions
public string CourseName { get; set; }
0 references | 0 exceptions
public IList<StudentCourse> StudentCourses { get; set; }
}
}

dbo.StudentCourses [Data] | AddStudents.cshtml | HomeController.cs | dbo.Courses [Data]

Student_Course | Student_Course.Models.StudentCourse

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5
6   namespace Student_Course.Models
7   {
        10 references
8       public class StudentCourse
9       {
            6 references | 0 exceptions
10          public int StudentId { get; set; }
            0 references | 0 exceptions
11          public Student Student { get; set; }
12
            6 references | 0 exceptions
13          public int CourseId { get; set; }
14
            0 references | 0 exceptions
15          public Course Course { get; set; }
16
17      }
18  }
19
```

dbo.StudentCourses [Data] | HomeController.cs | dbo.Courses [Data] | Student.cs | Course.cs | StudentCourse.cs | StudentContext.cs

Student_Course | Student_Course.Models.StudentContext | Courses

```
7   namespace Student_Course.Models
8   {
        7 references
9       public class StudentContext:DbContext
10      {
            0 references | 0 exceptions
11          public StudentContext(DbContextOptions<StudentContext> options) : base(options)
12          {
13
14          }
            0 references | 0 exceptions
15          protected override void OnModelCreating(ModelBuilder modelBuilder)
16          {
17              modelBuilder.Entity<StudentCourse>()
18                  .HasKey(sc => new { sc.StudentId, sc.CourseId });
19          }
20
            6 references | 0 exceptions
21          public DbSet<Student> Students { get; set; }
            6 references | 0 exceptions
22          public DbSet<Course> Courses { get; set; }
            5 references | 0 exceptions
23          public DbSet<StudentCourse> StudentCourses { get; set; }
24      }
25  }
```

Tables

dbo.StudentCourses [Data] | SC.cs | AddStudentCours

Max Rows: 1000

StudentId	StudentName	StudentAddress
1	Rahim	Uttara
2	Rahim	Uttara
3	Mounita	Gazipur
4	Moin	Mirpur
5	Moinul	Mirpur
NULL	NULL	NULL

dbo.StudentCourses [Data] | SC.cs

Max Rows: 1000

CourseId	CourseName
1	OS
2	CA
3	Java
4	C#
NULL	NULL

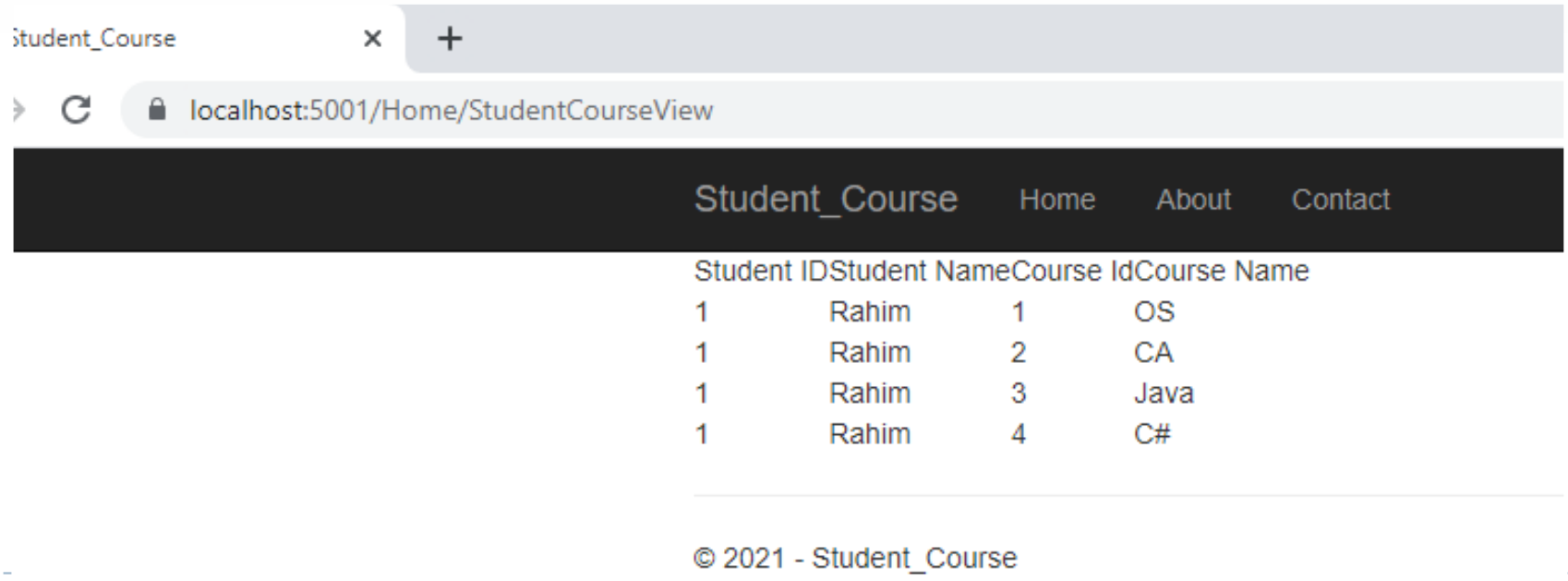
dbo.StudentCourses [Data] | SC.cs

Max Rows: 1000

StudentId	CourseId
1	1
2	1
1	2
2	2
1	3
1	4
NULL	NULL

```
0 references | 0 requests | 0 exceptions
public IActionResult StudentCourseView()
{
    var scv = from s in ctx.Students
              from c in ctx.Courses
              from sc in ctx.StudentCourses
              where s.StudentId == 1 && sc.StudentId==1 && sc.CourseId==c.CourseId
              select new SC {
                  StudentId = s.StudentId,
                  StudentName = s.StudentName,
                  CourseId = c.CourseId,
                  CourseName = c.CourseName
              };

    return View(scv);
}
```



AddStudent

```
public IActionResult AddStudent()
{

    var std = new Student
    {

        StudentName = "Mahin",
        DateOfBirth = new DateTime(2006, 1, 25),
        Height = 25.17m,
        Weight = 100.2F,
        GradeId=1
    };
    ctx.Students.Add(std);
    ctx.SaveChanges();
```

AddGrade

```
0 references | 0 requests | 0 exceptions
public IActionResult AddGrade() {
    var grd1= new Grade {  GradeName = "A", Section = "+" };
    ctx.Grades.Add(grd1);
    object p1 = ctx.Database.ExecuteSqlCommand(@"SET IDENTITY_INSERT [dbo].[Grades] ON");
    ctx.SaveChanges();
    ctx.Database.ExecuteSqlCommand(@"SET IDENTITY_INSERT [dbo].[Grades] OFF");
    var grd2 = new Grade {  GradeName = "A", Section = "-" };
    ctx.Grades.Add(grd2);
    object p2 = ctx.Database.ExecuteSqlCommand(@"SET IDENTITY_INSERT [dbo].[Grades] ON");
    ctx.SaveChanges();
    ctx.Database.ExecuteSqlCommand(@"SET IDENTITY_INSERT [dbo].[Grades] OFF");
    var grd3 = new Grade {  GradeName = "A", Section = " " };
    ctx.Grades.Add(grd3);
    object p3 = ctx.Database.ExecuteSqlCommand(@"SET IDENTITY_INSERT [dbo].[Grades] ON");
    ctx.SaveChanges();
    ctx.Database.ExecuteSqlCommand(@"SET IDENTITY_INSERT [dbo].[Grades] OFF");
```

Delete and Update Queries

```
0 references | 0 requests | 0 exceptions
public IActionResult DeleteStudent()
{
    var st = (from s in ctx.Students
              where (s.StudentId == 2)
              select s).FirstOrDefault();
    ctx.Students.Remove(st);
    ctx.SaveChanges();

    return View(ctx.Students);
}
```

```
public IActionResult UpdateStudent()
{
    var st = (from s in ctx.Students
              where (s.StudentId == 2)
              select s).FirstOrDefault();
    st.StudentName = "Moni";
    ctx.Students.Update(st);
    ctx.SaveChanges();

    return View(ctx.Students);
}
```

ShowGradeStudents

```
0 references | 0 requests | 0 exceptions
public IActionResult ShowGradeStudents()
{
    var sq = from g in ctx.Grades
             from s in ctx.Students
             where g.GradeName == "A" && g.GradeId==s.GradeId
             select new Student_Grade
             {
                 StudentId = s.StudentId,
                 StudentName = s.StudentName,
                 GradeName = g.GradeName,
                 Section = g.Section
             };
    //Student-Grade result= sq.ToList();
    return View(sq);
}
```

ShowStudentGrade

```
Student IdStudentNameStudent Grade
1        Rahim       A+
3        Mahin       A+
4        Rimi        A-

© 2021 - CodeFirstSimple
```

```
0 references | 0 requests | 0 exceptions
public IActionResult ShowStudentGrade()
{
    var sq = from s in ctx.Students
             from g in ctx.Grades
             where g.GradeId == s.GradeId
             select new Student_Grade{ StudentId=s.StudentId, StudentName=s.StudentName,
                 GradeName=g.GradeName, Section=g.Section };
   //Student-Grade result= sq.ToList();
    return View(sq);
}
```

```
namespace CodeFirstSimple.Models
{
    8 references
    public class Student_Grade
    {
        4 references | 0 exceptions
        public int StudentId { get; set; }
        4 references | 0 exceptions
        public string StudentName { get; set; }
        4 references | 0 exceptions
        public string GradeName { get; set; }
        4 references | 0 exceptions
        public string Section { get; set; }
    }
}
```

```
@model IEnumerable<CodeFirstSimple.Models.Student_Grade>
<html>
<body>
    <table>
        <tr>
            <td> Student Id </td>
            <td> StudentName</td>
            <td> Student Grade</td>
        </tr>
        @foreach (var s in Model)
        {
            <tr>
                <td> @s.StudentId </td>
                <td> @s.StudentName </td>
                <td>  @s.GradeName@s.Section</td>
</tr>
        }

    </table>

</body>

</html>
```

MCSE 541: Visual Programming

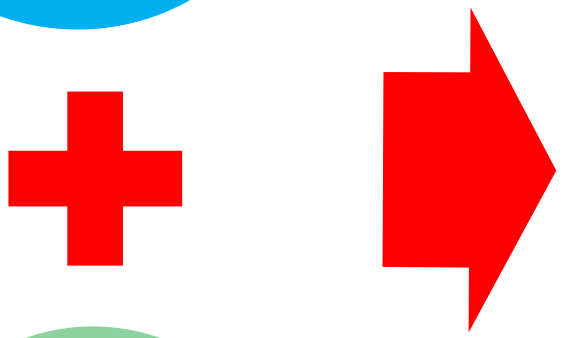
Authentication and Authorization in ASP.NET Core

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

Two Interlinked Concepts

I) Authentication

Authentication is the process of obtaining some sort of credentials from the users and using those credentials to verify the user's identity.
"Who is the users?"



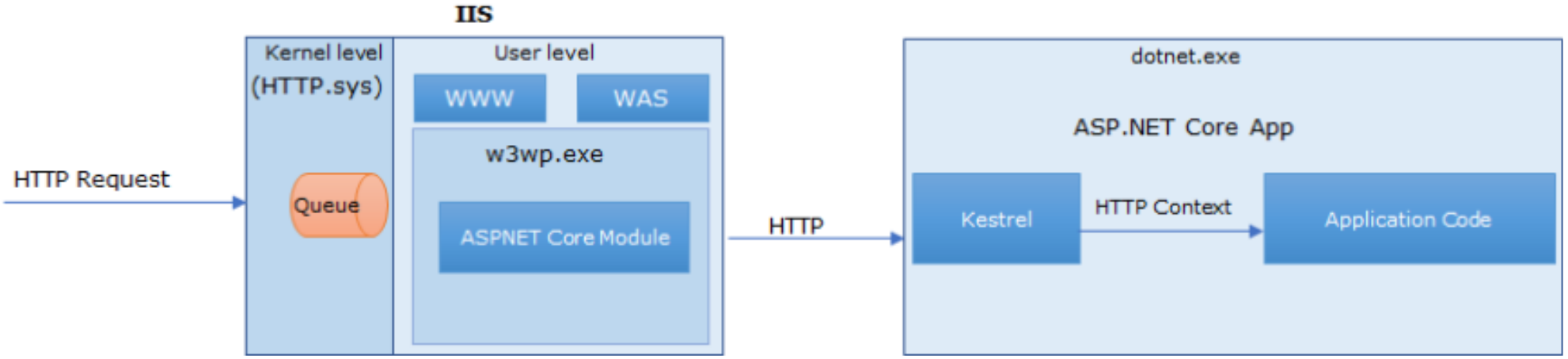
Security for distributed applications

II) Authorization

Authorization is the process of allowing an authenticated user access to resources.
-What right the user has? What resources the user can access

How about **anonymous users** want to connect and use the application

Two Layers of Authentication – IIS and ASP.NET



How a request is served in this scenario:

1. The request is received by the *HTTP.sys* from the network.
2. If response is cached at *HTTP.sys* then it is sent back from there
   else gets a place the corresponding Application Pool's queue.

3. When a thread is available in the thread pool, it picks up the request and start processing it.

4. The request goes through IIS processing pipeline. Including few native IIS modules and once it reaches to **ANCM(ASPNET Core Module)**, it forwards the request to Kestrel (under dotnet.exe).

How a request is served.

5. ANCM has a responsibility to manage the process as well.
   - It (re)starts the process (if not running or crashed) and
   - IIS integration middleware configures the server to listen the request on port defined in environment variable.

   Server only accepts the requests which originates from ANCM.

   Note -Please do note that in ASP.NET Webforms/MVC the application is hosted under the worker process w3wp.exe which is managed by Windows Activation Service (WAS) which was part of IIS.

6. Once the request is received by Kestrel, it creates the ***HTTPContext* object** and request is handed over to ASP.NET Core middleware pipeline.

7. The request is passed to routing middleware which invokes the right controller and action method (model binding, various filters almost similar way as earlier versions).

8. Finally, the response is returned from the action and passed to kestrel via Middlewares and later sent back to client via IIS.

Steps in the joint IIS and ASP.net authentication process

1. IIS first checks to make sure the incoming request comes from an IP address that is allowed access to the domain. If not it denies the request.
2. Next IIS performs its own user authentication if it is configured to do so. By **default IIS allows anonymous access**, so requests are automatically authenticated, but you can change this default on a per - application basis with in IIS.

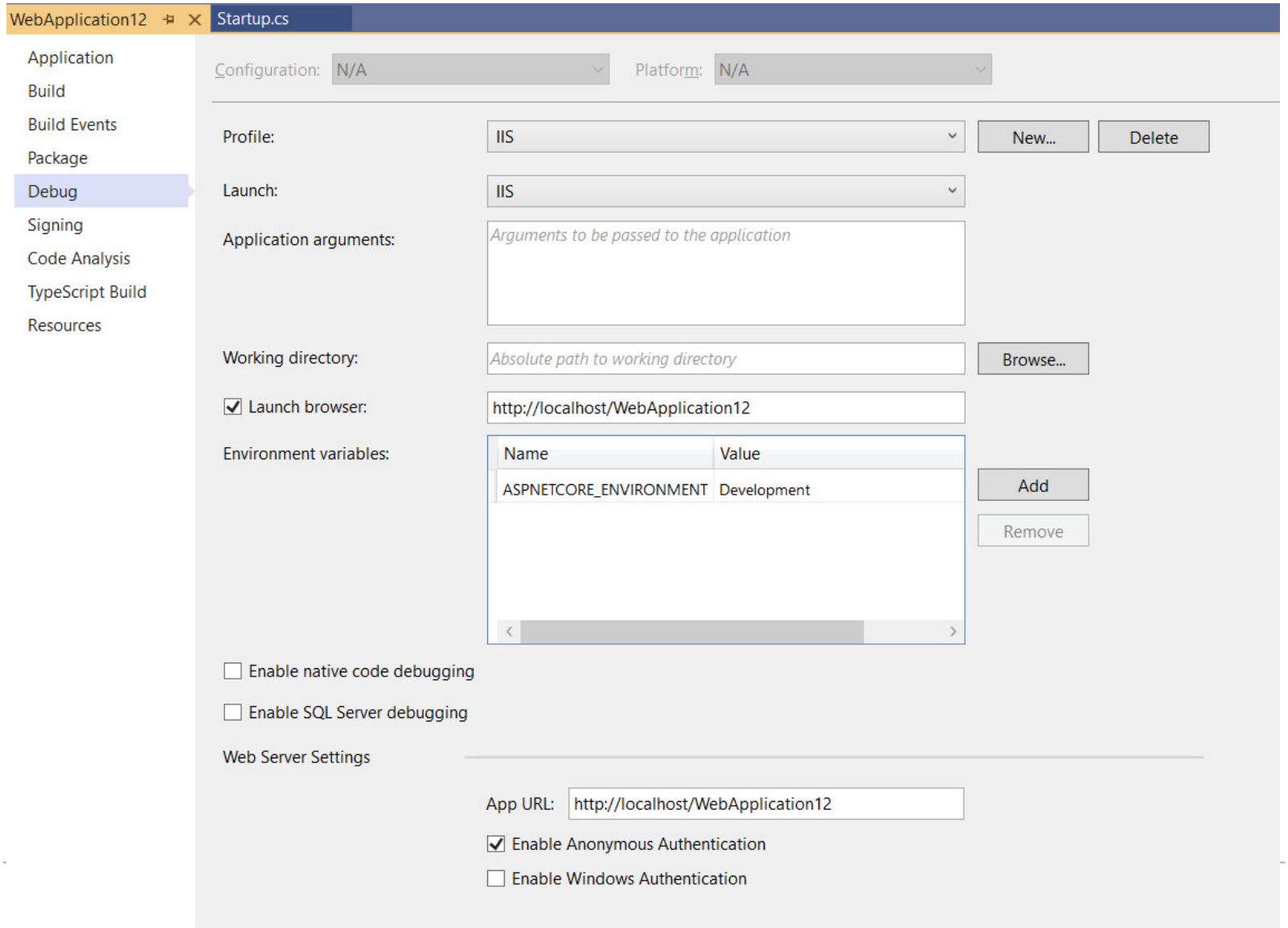
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.web>
      <compilation debug="true" targetFramework="4.0" />
      <identity impersonate="true"/>
    </system.web>
</configuration>
```

3. If the request is passed to ASP.net with an authenticated user, ASP.net checks to see whether impersonation is enable. If impersonation is enabled, ASP.net acts as though it were the authenticated user using IUSR account. If not ASP.net acts with its own configured account using application pool identity .

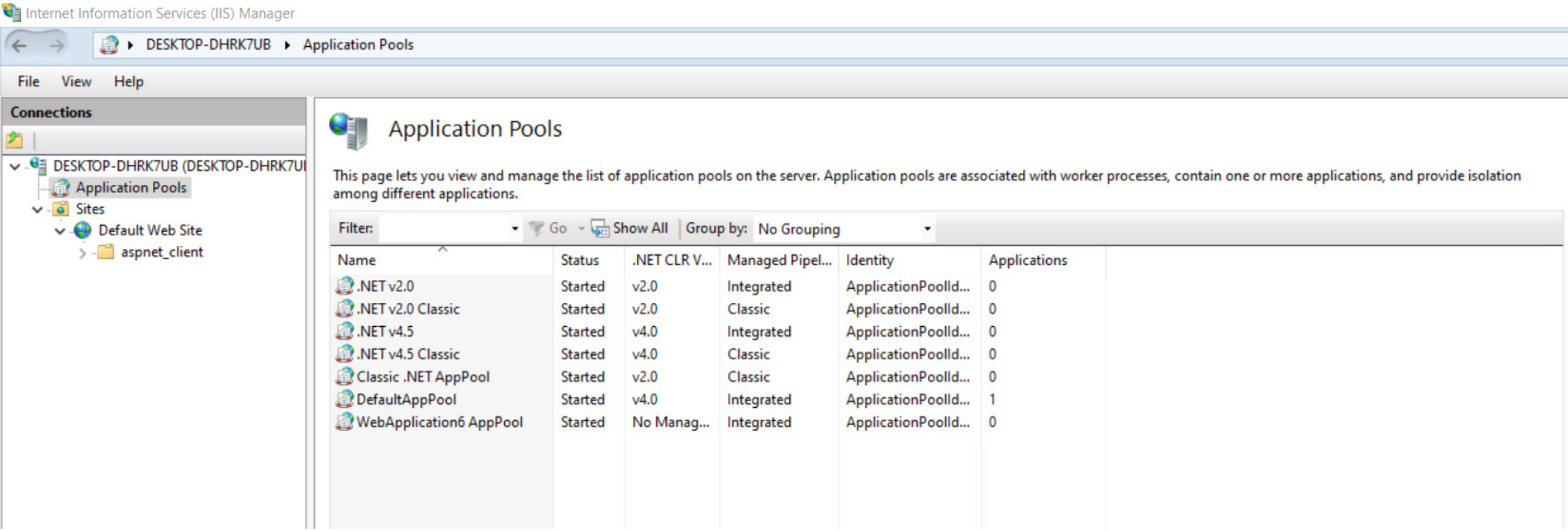
4. Finally the identity from step 3 is used to request resources from the operating system. If ASP.net authentication can obtain all the necessary resources it grants the users request otherwise it is denied. Resources can include much more than just the ASP.net page itself you can also use .Net's code access security features to extend this authorization step to disk files, Registry keys and other resources.

Run A Program In IIS server

- Solution->Properties->Debug

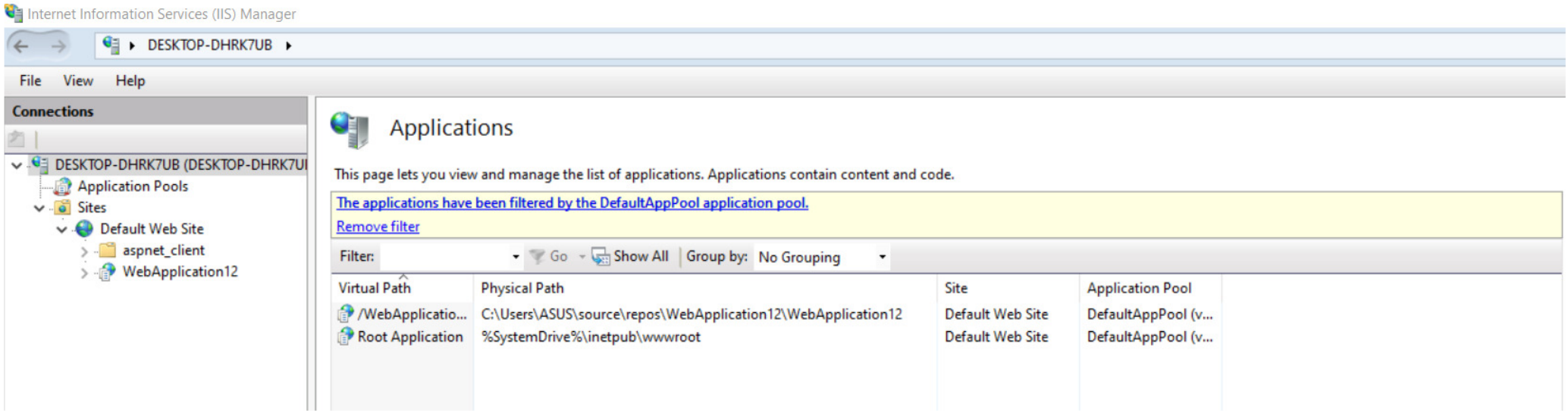


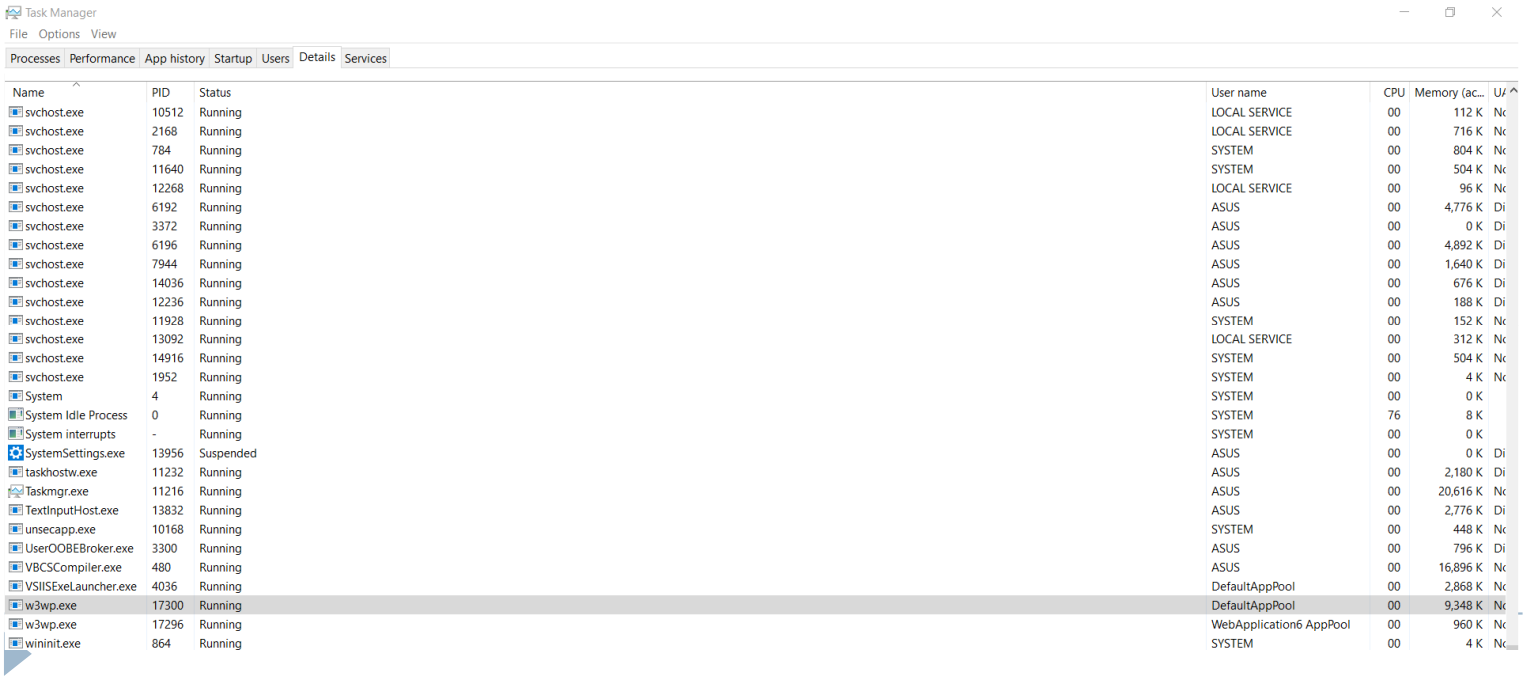
IIS Application Pools



Each Application runs their own processes.

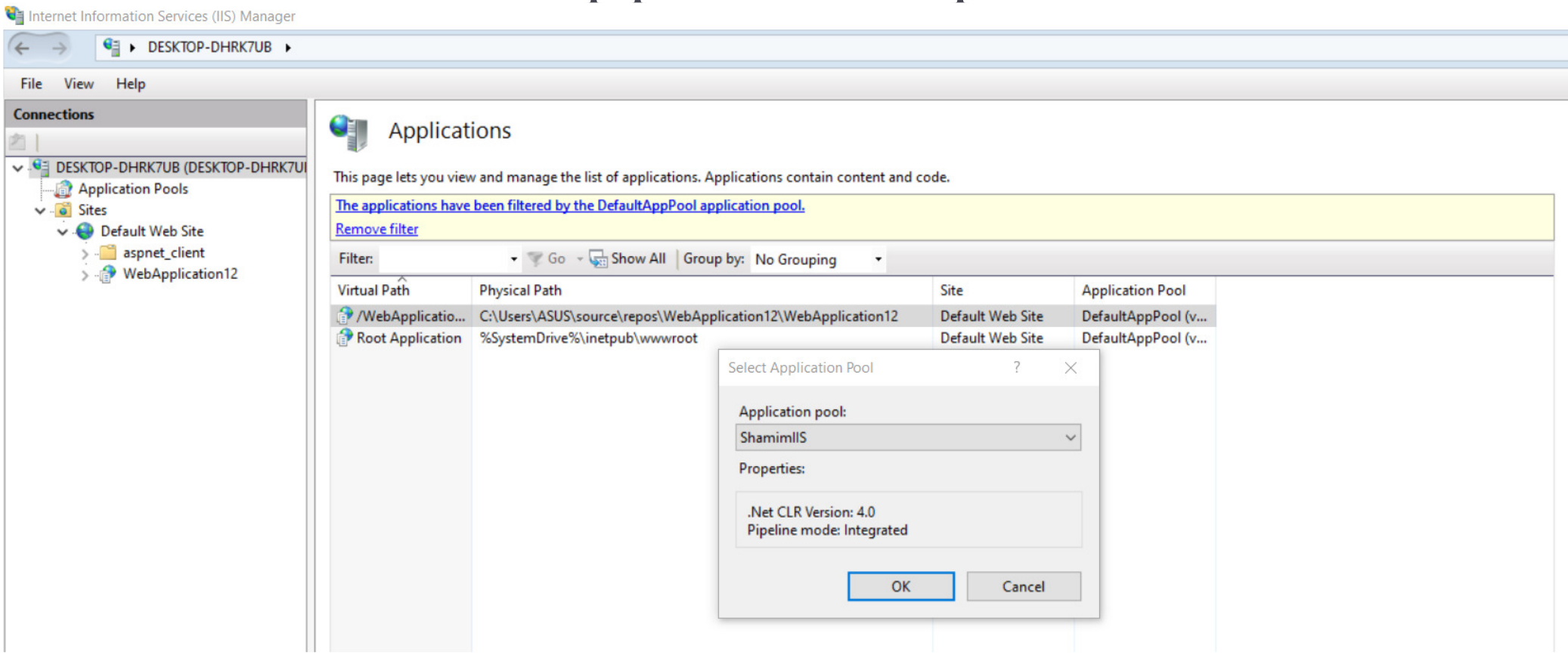
Running Program in DefaultAppPool





End the task from Task Manager. The webpage will be closed also.

Create a new application pool



System Idle Process	0	Running	SYSTEM
System interrupts	-	Running	SYSTEM
SystemSettings.exe	13956	Suspended	ASUS
taskhostw.exe	11232	Running	ASUS
Taskmgr.exe	11216	Running	ASUS
TextInputHost.exe	13832	Running	ASUS
unsecapp.exe	10168	Running	SYSTEM
UserOOBEBroker.exe	3300	Running	ASUS
VBCSCompiler.exe	480	Running	ASUS
VSIISExeLauncher.exe	6864	Running	ShamimIIS
w3wp.exe	17596	Running	ShamimIIS
w3wp.exe	17300	Running	DefaultAppPool
wininit.exe	864	Running	SYSTEM
winlogon.exe	11208	Running	SYSTEM

Anonymous User Enable

launchSettings.json | appsettings.json | web.config | Index.cshtml

Schema: https://json.schemastore.org/launchsettings

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iis": {
      "applicationUrl": "http://localhost/WebApplication12"
      "sslPort": 0
    },
    "iisExpress": {
      "applicationUrl": "http://localhost:7047",
      "sslPort": 44354
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "WebApplication12": {
      "commandName": "Project",
      "launchBrowser": true,
      "environmentVariables": {
```

User Authenticate as PoolIdentity

```
ViewBag.Name=System.Security.Principal.WindowsIdentity.GetCurrent().Name;
ViewBag.Idenity = User.Identity.IsAuthenticated.ToString();
```

ApplicationPoolIdentity

The least privileged one.

Create a Virtual account for each new application pool and run worker processes under this account.

Running an application with low privileged one is good practice

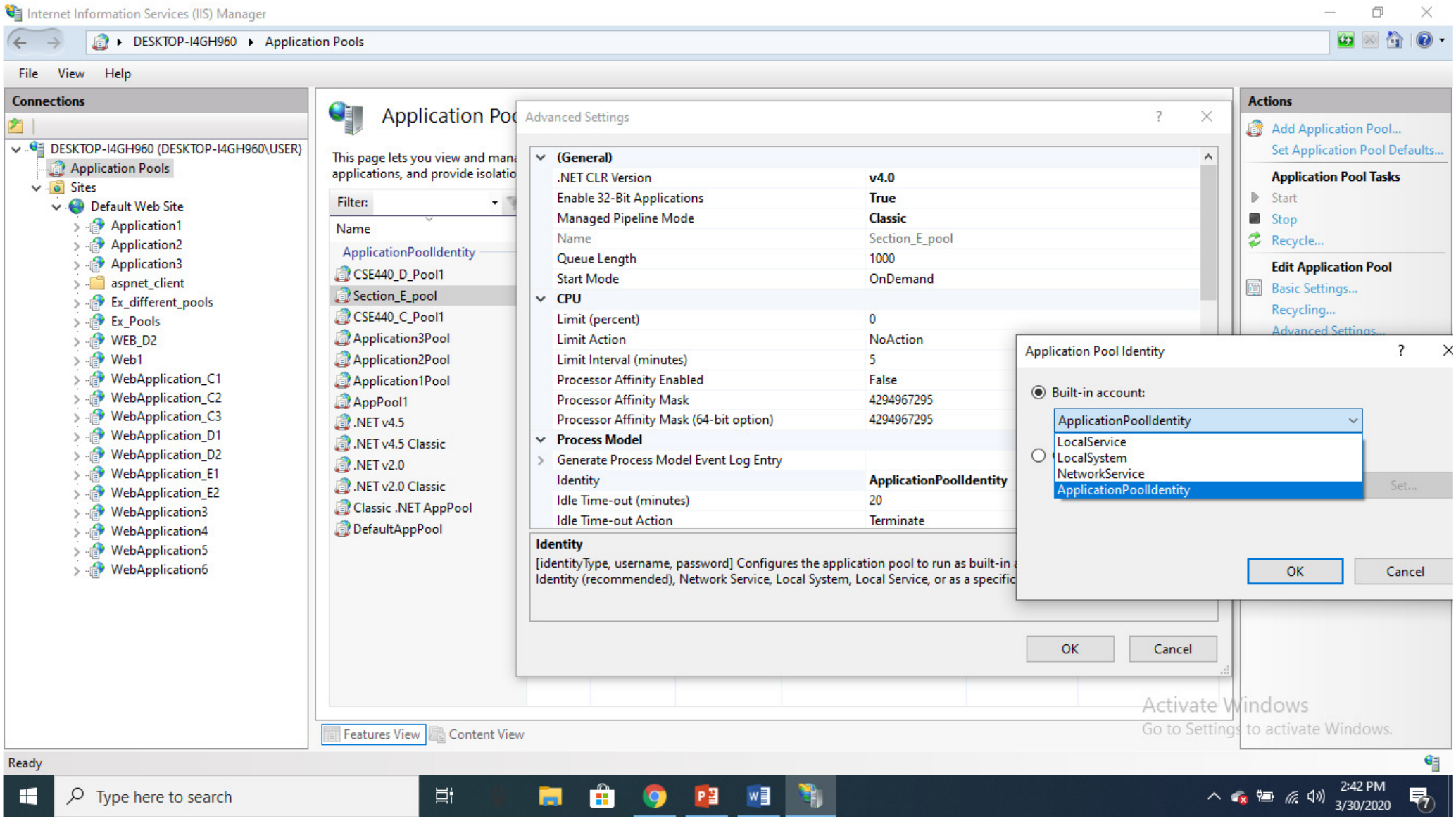
Application uses

- IIS APPPOOL\DefaultAppPool
- Sample pages using ASP.NET Core MVC
- Theming using Bootstrap

Application uses

- IIS APPPOOL\ShamimIIS
- Sample pages using ASP.NET Core MVC
- Theming using Bootstrap

Types of Identity with Pool



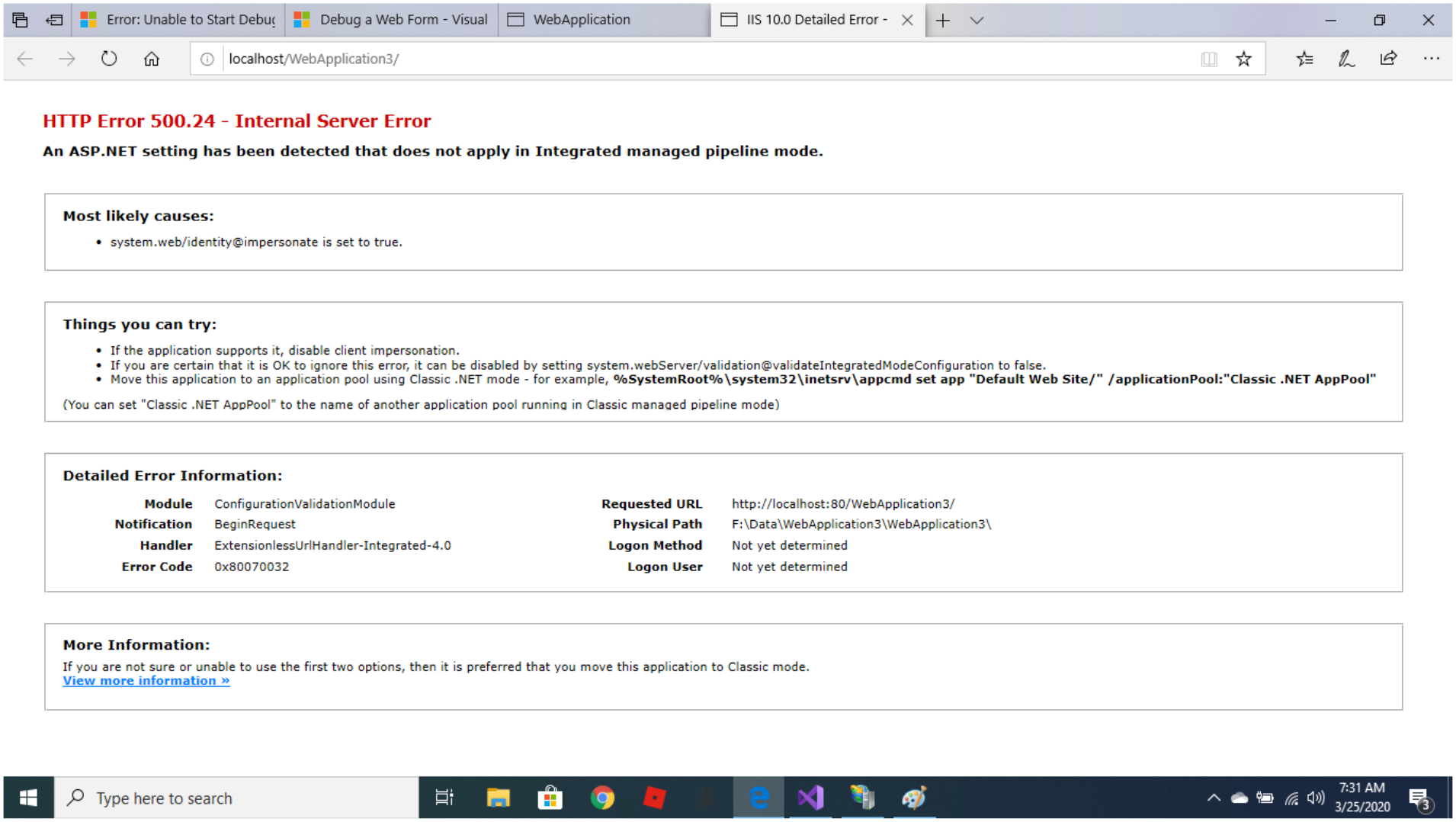
Impersonate enable ASP.NET

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.web>
      <compilation debug="true" targetFramework="4.0" />
      <identity impersonate="true"/>
    </system.web>
</configuration>
```

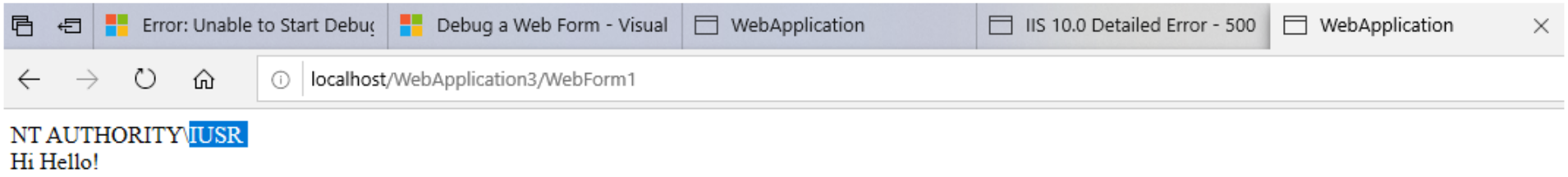
pretend to be (another person) for entertainment or fraud.

Impersonate is not supported in ASP.NET Core

```
<configuration>
  <system.web>
    <identity impersonate="true"/>
    <compilation debug="true" targetFramework="4.7.2" />
    <httpRuntime targetFramework="4.7.2" />
  </system.web>
  <system.codedom>
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs"
type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider
, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=2.0.1.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4"
compilerOptions="/langversion:default /nowarn:1659;1699;1701" />
      <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider,
Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=2.0.1.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4"
compilerOptions="/langversion:default /nowarn:41008
/define:_MYTYPE=\&quot;Web\&quot; /optionInfer+" />
    </compilers>
  </system.codedom>
</configuration>
```



Change pool to classical mode


Error: Unable to Start Debu
Debug a Web Form - Visual
WebApplication
IIS 10.0 Detailed Error - 500
WebApplication
localhost/WebApplication3/WebForm1
NT AUTHORITY\IUSR
Hi Hello!

Authentication providers

- The ASP.net architecture includes
  - two(2) different authentication providers
    - Windows Authentication provider
      - Authenticates users based on their windows accounts.
      - This provider uses IIS to perform the authentication and then passes the authenticated identity to the code.
      - This is the default provided for ASP.net.
    - Forms authentication provider
      - uses custom HTML forms to collect authentication information and lets you use your own logic to authenticate users.
      - The user's credentials are stored in a cookie for use during the session.

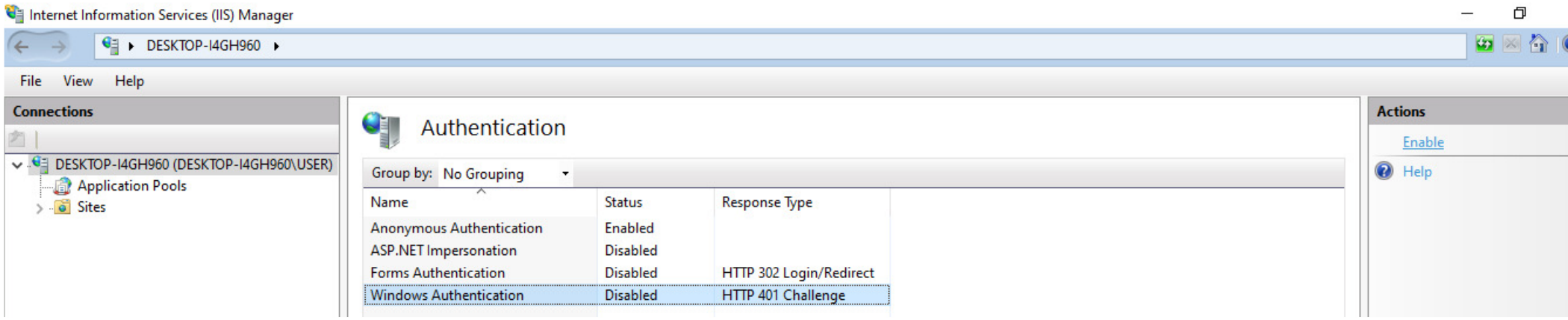
Selecting an authentication provider is as simple as making an entry in the web.config file for the application.

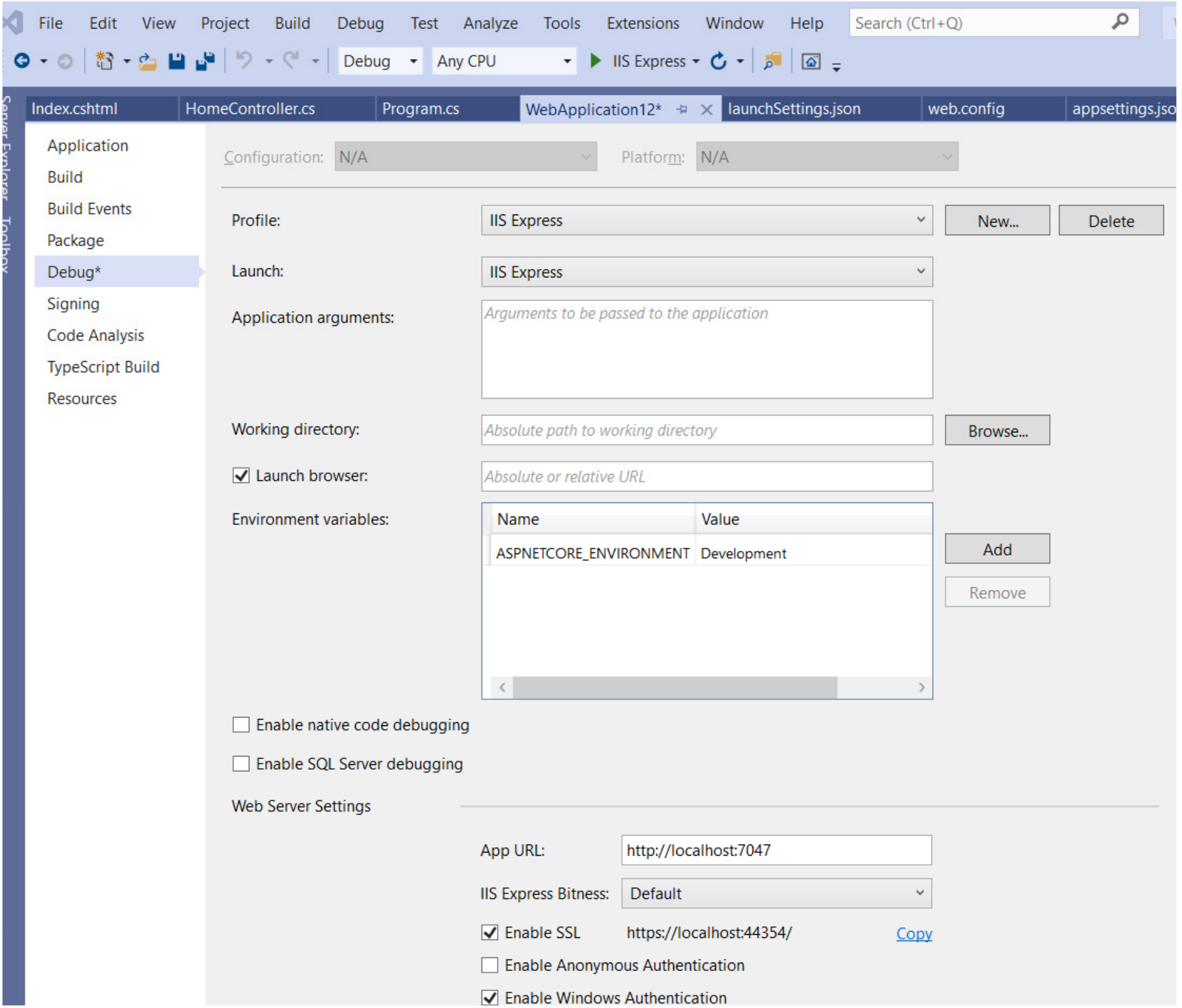
```
<authentication mode="windows">
```

```
<authentication mode="forms">
```

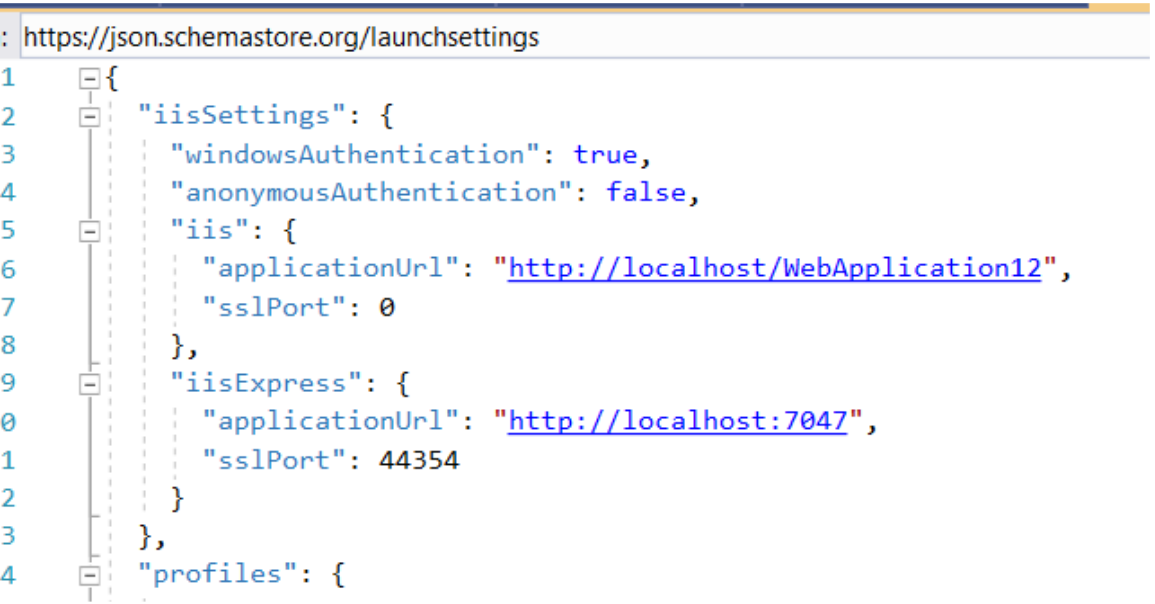
Windows Authentication

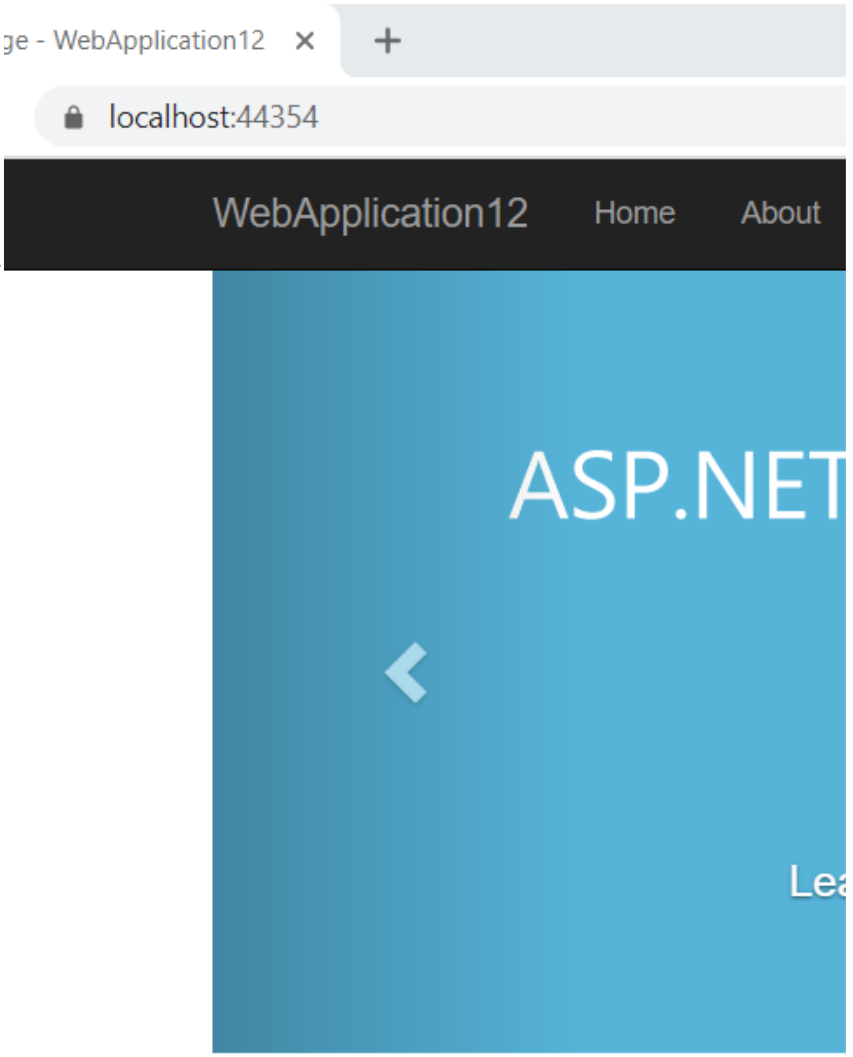
- Enable it in IIS




File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Any CPU
IIS Express
Index.cshtml
HomeController.cs
Program.cs
WebApplication12*
launchSettings.json
web.config
appsettings.jso
Server Explorer
Toolbox
Application
Build
Build Events
Package
Debug*
Signing
Code Analysis
TypeScript Build
Resources
Configuration: N/A
Platform: N/A
Profile: IIS Express
New...
Delete
Launch: IIS Express
Application arguments: Arguments to be passed to the application
Working directory: Absolute path to working directory
Browse...
Launch browser: Absolute or relative URL
Environment variables:
Name
Value
ASPNETCORE_ENVIRONMENT
Development
Add
Remove
Enable native code debugging
Enable SQL Server debugging
Web Server Settings
App URL: http://localhost:7047
IIS Express Bitness: Default
Enable SSL
https://localhost:44354/
Copy
Enable Anonymous Authentication
Enable Windows Authentication

Windows Authentication IIS Express





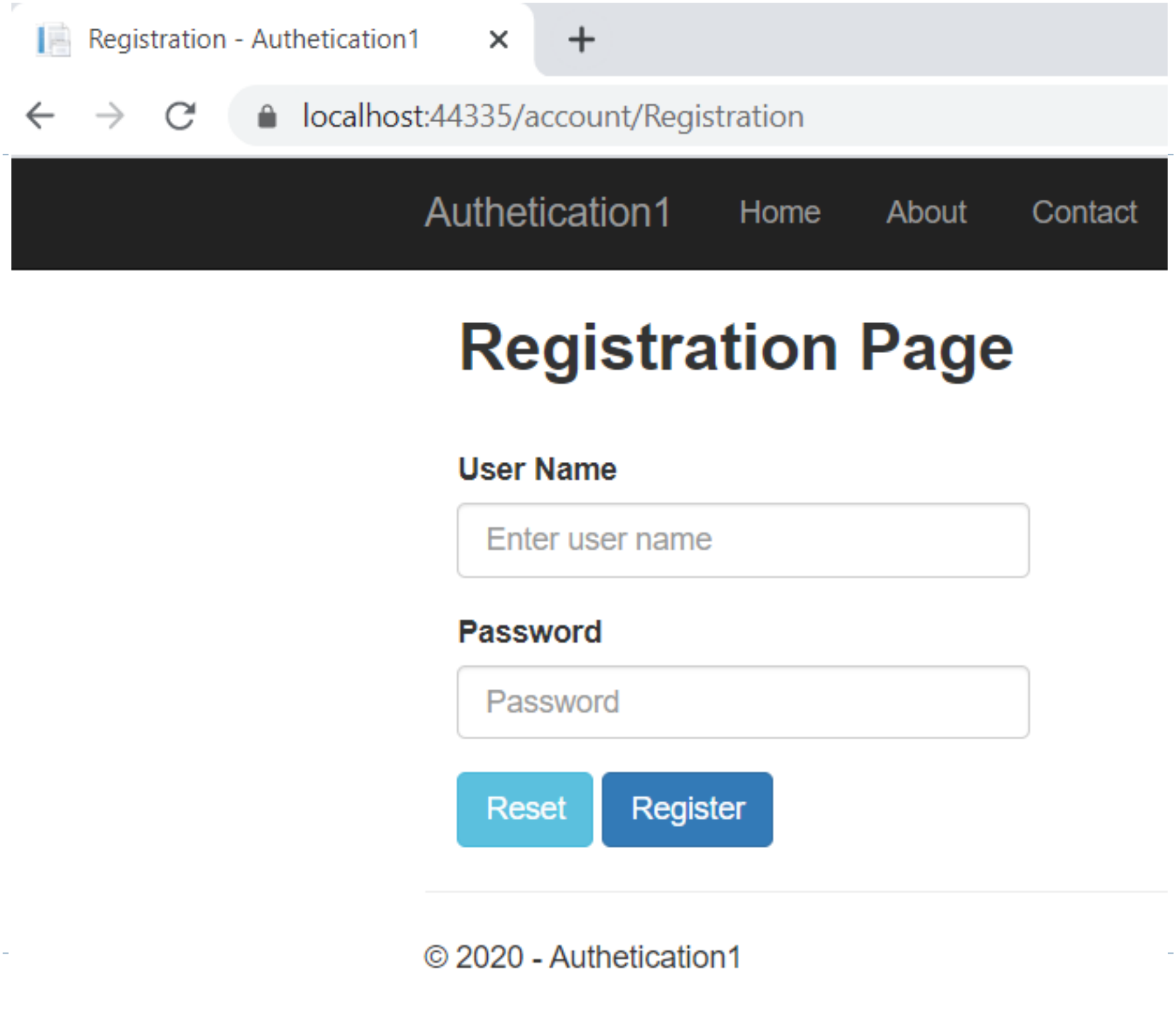
Forms Authentication

- Is used for Internet Web Applications.
- User do not have to be member of domain-based network (Windows Authentication)
- Authenticate users by using their own code and then maintain an authentication token in a cookie or in the page URL.
- Gmail.com, Amazon.com, Facebook.com are examples

How to use forms authentication?

- Create a registration page

- Create a login page
- Collects the credentials from the users use predefine code to authenticate the credentials.


Registration - Authetication1
localhost:44335/account/Registration
Authetication1
Home
About
Contact
Registration Page
User Name
Enter user name
Password
Password
Reset
Register
© 2020 - Authetication1

RegisterModel.cs [Step1]

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5
6   namespace Authetication1.Models
7   {
        8 references
8       public class RegisterModel
9       {
10
            2 references | 0 exceptions
11          public int Id { get; set; }
            4 references | 0 exceptions
12          public string Name { get; set; }
            5 references | 0 exceptions
13          public string Password { get; set; }
14
            0 references | 0 exceptions
15          public string ConPassword { get; set; }
16      }
17  }
18
```

RegisterModelContext.cs [Context class] Step-2

```
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Threading.Tasks;
6
7   namespace Authetication1.Models
8   {
        8 references
9       public class RegisterModelContext:DbContext
10      {
            0 references | 0 exceptions
11          public RegisterModelContext(DbContextOptions<RegisterModelContext> options) : base(options)
12          {
13
14          }
            2 references | 0 exceptions
15          public DbSet<RegisterModel> RegisterModels { get; set; }
16
            0 references | 0 exceptions
17          protected override void OnModelCreating(ModelBuilder modelBuilder)
18          {
19              modelBuilder.Entity<RegisterModel>().HasData(
20                  new RegisterModel { Id=1, Name = "Admin", Password = "password" },
21                  new RegisterModel { Id=2, Name = "Shamim",Password= "shamim" }
22                  );
23          }
24      }
25  }
```

0 % 0 2 Ln: 18 Ch: 6 SPC CR

Appsettings.json [Step-3]

appsettings.json | dbo.RegisterModels [Data] | Registration.cshtml | RegisterModel.cs | RegisterModelContext.cs*

Schema: https://json.schemastore.org/appsettings

```
1   {
2     "Logging": {
3       "LogLevel": {
4         "Default": "Warning"
5       }
6     },
7     "AllowedHosts": "*",
8     "MyKey": "Value of myKey from appsettings.json",
9     "ConnectionStrings": {
10      "DBConnection": "server=(localdb)\\MSSQLLocalDB; database=RegisterModels;Trusted_Connection=true"
11    }
12  }
13
```

Configuration in the Startup class [Step-4]

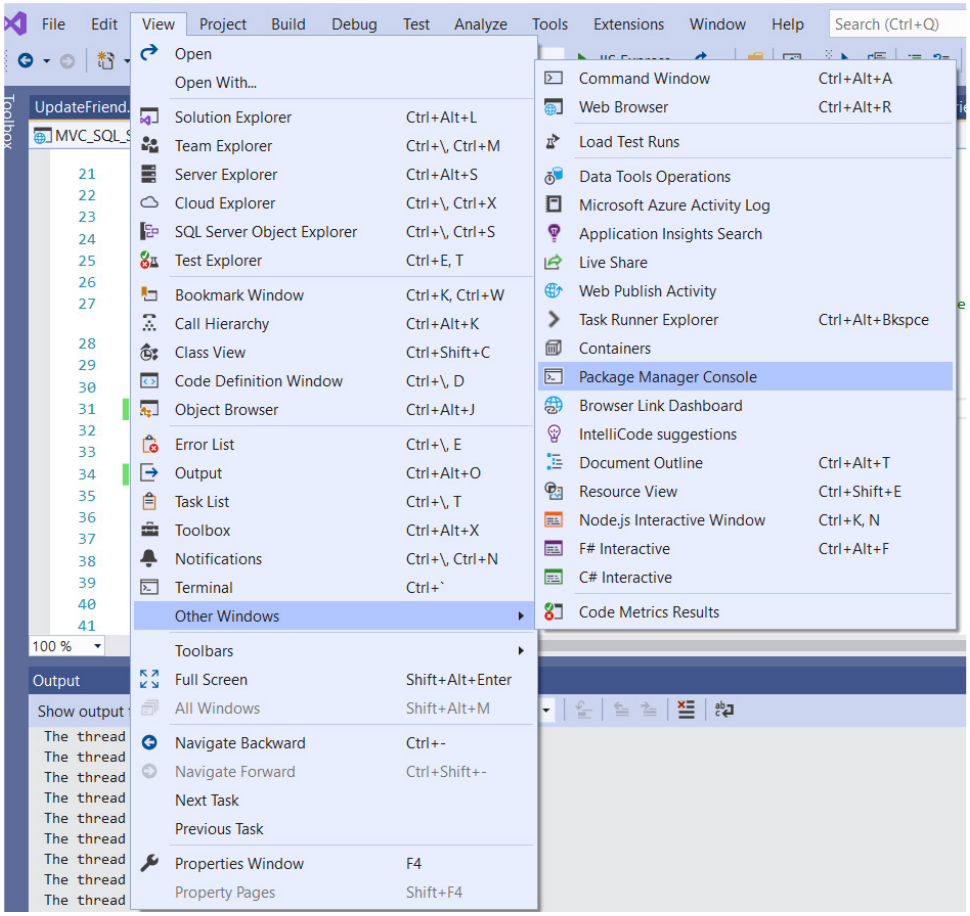
```
// This method gets called by the runtime. Use this method to add services to the container.
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{

    services.AddDbContextPool<RegisterModelContext>(
        options => options.UseSqlServer(
            Configuration.GetConnectionString("DBConnection")));

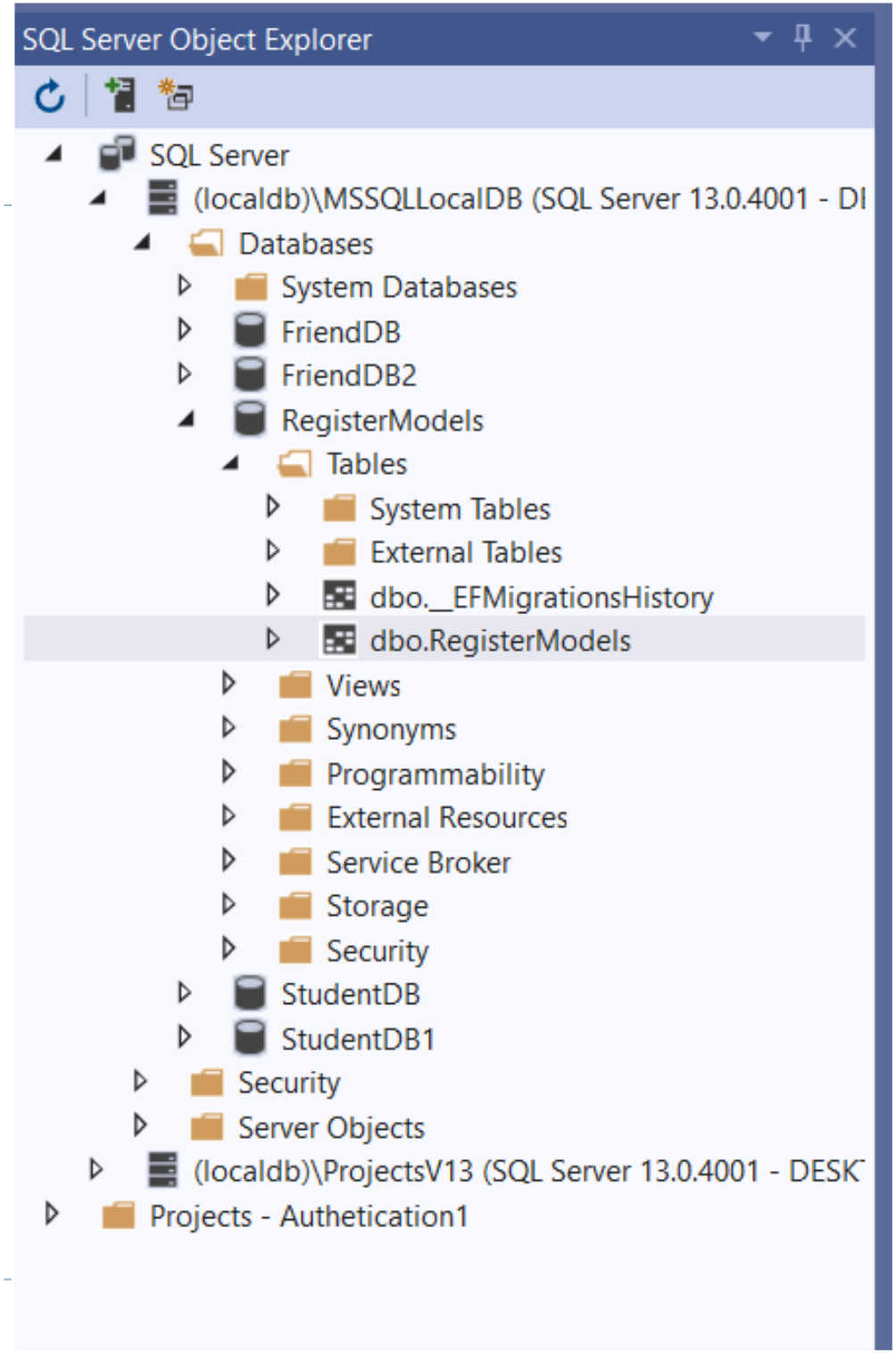
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

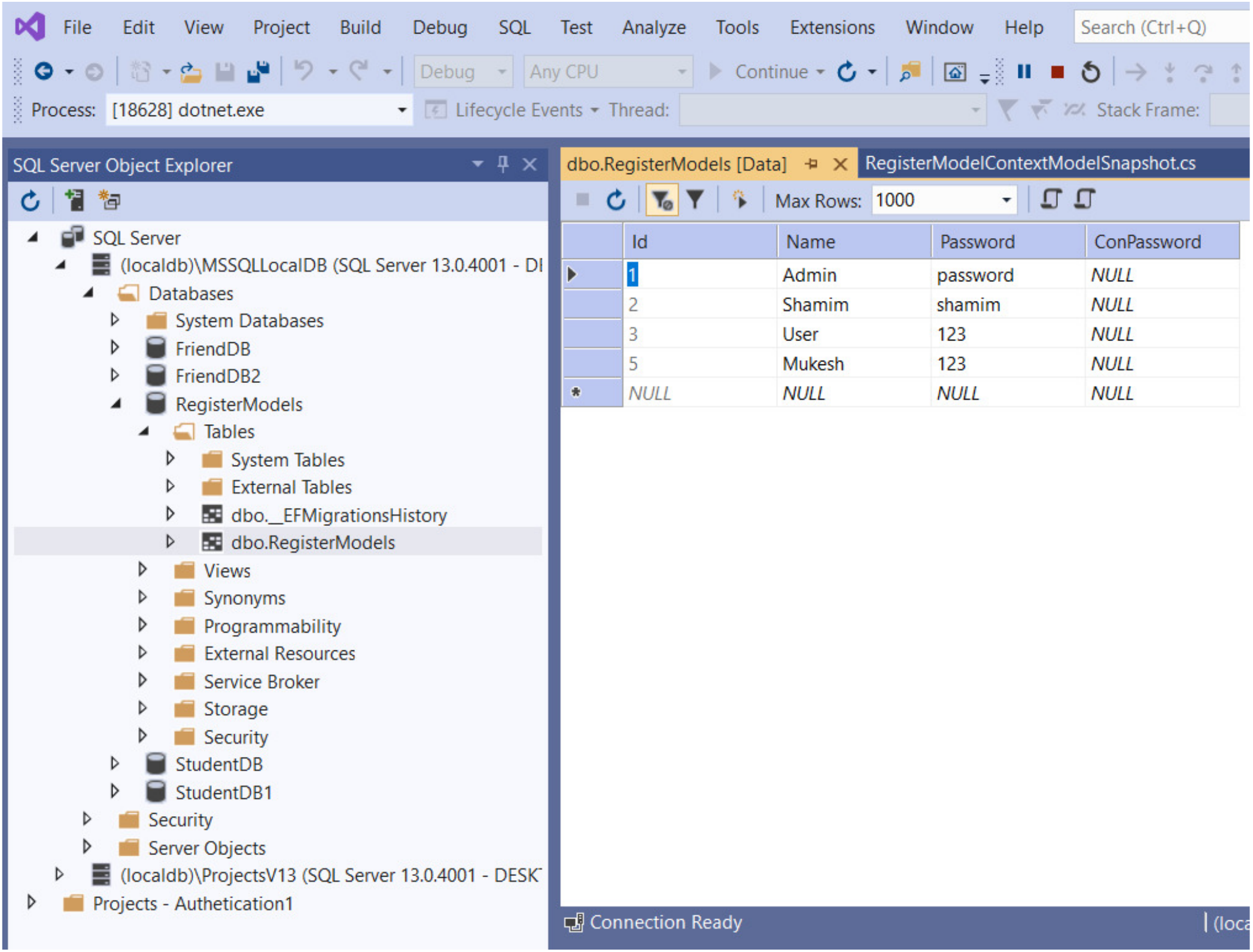
    services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie();
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Do Migration [Step-6]



- Add-Migration Initialize
  - A folder will appear
  - Current and future migration will appear there.
- Update-Database


SQL Server Object Explorer
SQL Server
(localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - DI
Databases
System Databases
FriendDB
FriendDB2
RegisterModels
Tables
System Tables
External Tables
dbo.__EFMigrationsHistory
dbo.RegisterModels
Views
Synonyms
Programmability
External Resources
Service Broker
Storage
Security
StudentDB
StudentDB1
Security
Server Objects
(localdb)\ProjectsV13 (SQL Server 13.0.4001 - DESK
Projects - Authetication1


File Edit View Project Build Debug SQL Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Debug
Any CPU
Continue
Process: [18628] dotnet.exe
Lifecycle Events
Thread:
Stack Frame:
SQL Server Object Explorer
SQL Server
(localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - DI
Databases
System Databases
FriendDB
FriendDB2
RegisterModels
Tables
System Tables
External Tables
dbo.__EFMigrationsHistory
dbo.RegisterModels
Views
Synonyms
Programmability
External Resources
Service Broker
Storage
Security
StudentDB
StudentDB1
Security
Server Objects
(localdb)\ProjectsV13 (SQL Server 13.0.4001 - DESK
Projects - Authetication1
dbo.RegisterModels [Data]
RegisterModelContextModelSnapshot.cs
Max Rows: 1000
Id Name Password ConPassword
1 Admin password NULL
2 Shamim shamim NULL
3 User 123 NULL
5 Mukesh 123 NULL
NULL NULL NULL NULL
Connection Ready

AuthenticationController

```
1 reference
public class AccountController : Controller
{
    private RegisterModelContext context1;
    private RegisterModel user;
    private List<RegisterModel> _register;
    0 references | 0 exceptions
    public AccountController(RegisterModelContext context1) {
        this.context1 = context1;

    }
    0 references | 0 requests | 0 exceptions
    public IActionResult Registration()
    {
        return View();
    }
    [HttpPost]
    0 references | 0 requests | 0 exceptions
    public IActionResult Registration(string userName, string password)
    {

        if (string.IsNullOrEmpty(userName) || string.IsNullOrEmpty(password))
        {
            return View();
        }

        else
        {
            // user = context1.RegisterModels.Find(userName);
            user = new RegisterModel { Name = userName, Password = password };

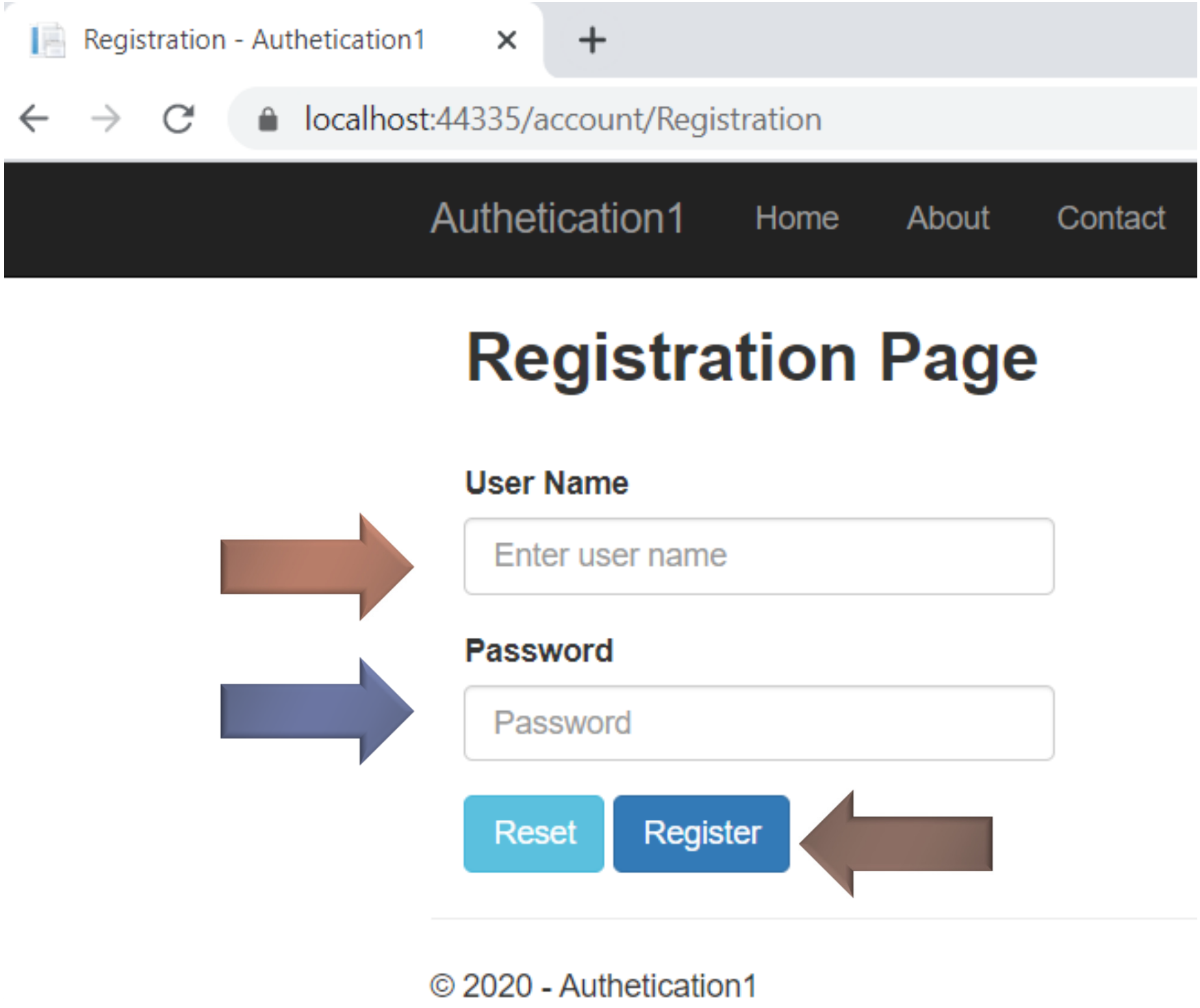
                context1.RegisterModels.Add(user);
                context1.SaveChanges();
                return RedirectToAction("Login"); //Give message already register

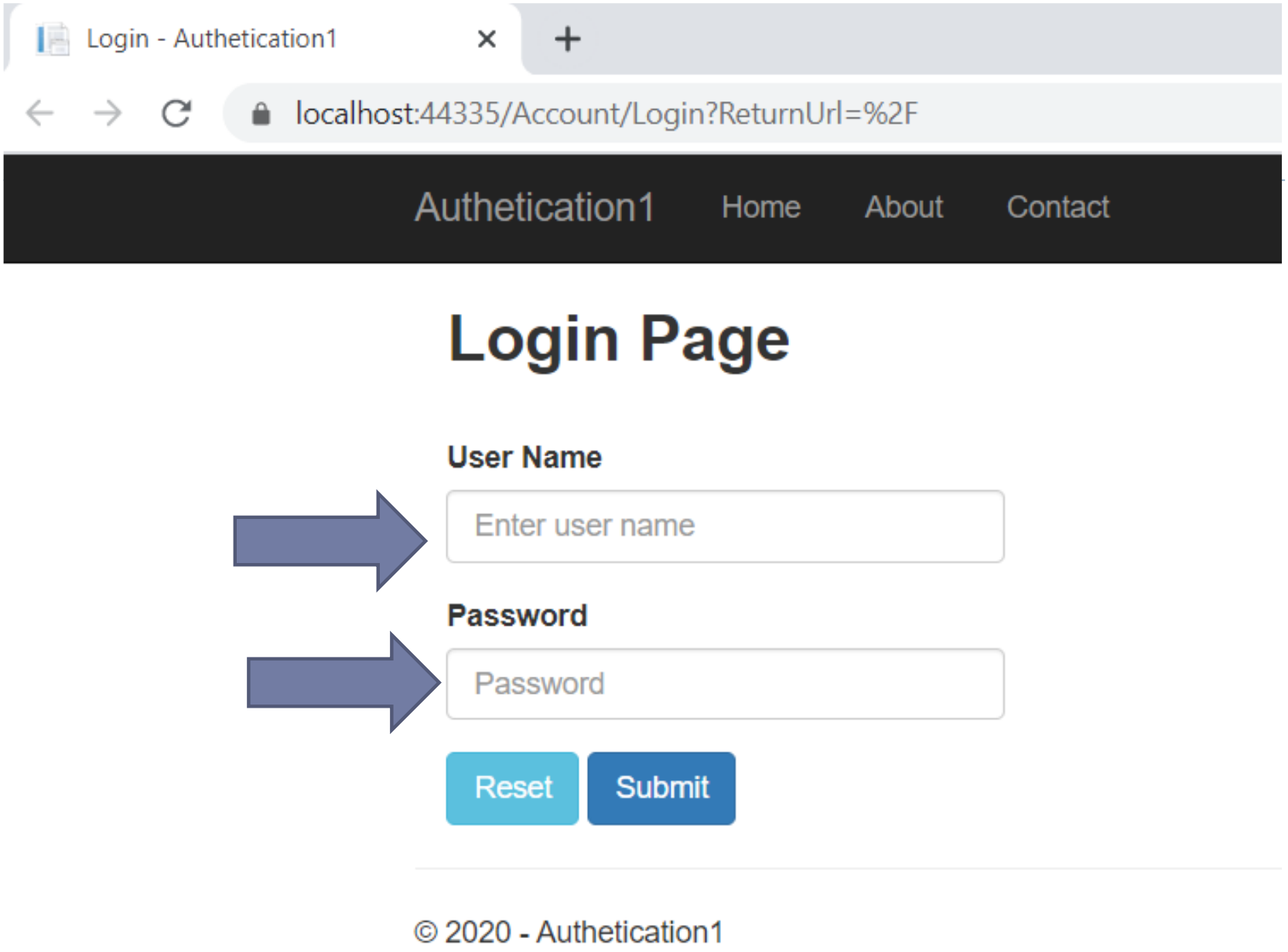
            }
            //return View();
    }
```

Registration View

appsettings.json | RegisterModel.cs | RegisterModelContext.cs* | Startup.cs | Registration.cshtml

```
1
2     @{ ViewData["Title"] = "Registration"; }
3
4   <div class="container">
5       <div class="row">
6           <div class="col-md-3">
7               <h2><strong>Registration Page </strong></h2><br />
8               <form asp-action="Registration" method="post">
9                   <div class="form-group">
10                      <label>User Name</label>
11                      <input type="text" class="form-control" id="userName" name="userName" placeholder=
12                  </div>
13                  <div class="form-group">
14                      <label>Password</label>
15                      <input type="password" class="form-control" name="password" id="password" placehol
16                  </div>
17                  <div class="form-check">
18                      <button class="btn btn-info" type="reset">Reset</button>
19                      <button type="submit" class="btn btn-primary">Register</button>
20                  </div>
21              </form>
22          </div>
23      </div>
24  </div>
25
26
```


Registration - Authetication1
localhost:44335/account/Registration
Authetication1
Home
About
Contact
Registration Page
User Name
Enter user name
Password
Password
Reset
Register
© 2020 - Authetication1


Login - Authetication1
localhost:44335/Account/Login?ReturnUrl=%2F
Authetication1
Home
About
Contact
Login Page
User Name
Enter user name
Password
Password
Reset
Submit
© 2020 - Authetication1

Login View

```
        [HttpGet]
        0 references | 0 requests | 0 exceptions
        public IActionResult Login()
        {
            return View();
        }
```

Authetication\Login

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

```
<div class="container">
    <div class="row">
        <div class="col-md-3">
            <h2><strong>Login Page </strong></h2><br />
            <form asp-action="login" method="post">
                <div class="form-group">
                    <label>User Name</label>
                    <input type="text" class="form-control" id="userName" name="userName" placeholder="Enter user name
                </div>
                <div class="form-group">
                    <label>Password</label>
                    <input type="password" class="form-control" name="password" id="password" placeholder="Password">
                </div>
                <div class="form-check">
                    <button class="btn btn-info" type="reset">Reset</button>
                    <button type="submit" class="btn btn-primary">Submit</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

[HttpPost]

```
public IActionResult Login(string userName, string password)
{
    if (string.IsNullOrEmpty(userName) || string.IsNullOrEmpty(password))
    {
        return RedirectToAction("Login");
    }
    //Check the user name and password
    //Here can be implemented checking logic from the database
    ClaimsIdentity identity = null;
    bool isAuthenticated = false;

    //user=context1.RegisterModels.Find(userName);

    IEnumerable<RegisterModel> _register =  context1.RegisterModels;
    //int userId= _register.Max(v => v.Id) + 1;
    foreach (var user in _register)
    {
        if (userName == "Admin" && password == user.Password)
        {
            var claims = new List<Claim> {
                new Claim(ClaimTypes.Name, userName),
                new Claim(ClaimTypes.Role, "Admin")
            };
            identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
            isAuthenticated = true;
            break;
        }
        else if (userName == user.Name && password == user.Password)
        {
            var claims = new List<Claim> {
                new Claim(ClaimTypes.Name, userName),
                new Claim(ClaimTypes.Role, "User")
            };
            identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
            isAuthenticated = true;
            break;

        }
    }
```

A ClaimsIdentity describe the entity that the corresponding identity represents, and can be used to make authorization and authentication decisions.

```
    if (isAuthenticated)
    {
        var principal = new ClaimsPrincipal(identity);

        var login = HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal);

        return RedirectToAction("Index", "Home");
    }
    else return RedirectToAction("Registration", "Account");
}
```

ClaimsPrincipal exposes a collection of identities, each of which is a ClaimsIdentity. In the common case, this collection, which is accessed through the Identities property, will only have a single element.

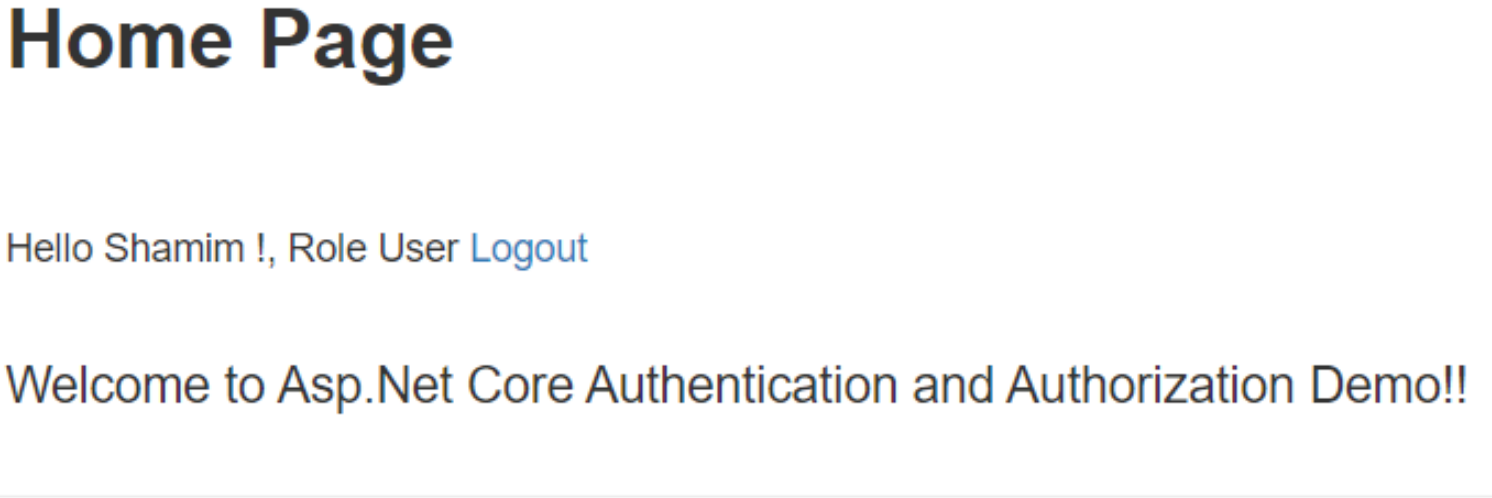
Home Controller

```
10  namespace Authetication1.Controllers
11  {
12      [Authorize(Roles = "Admin, User")]
            0 references
13      public class HomeController : Controller
14      {
                0 references | 0 requests | 0 exceptions
15          public IActionResult Index()
16          {
17              return View();
18          }
19          [Authorize(Roles = "Admin")]
                0 references | 0 requests | 0 exceptions
20          public IActionResult Setting()
21          {
22              return View();
23
24          }
25          [Authorize(Roles = "Admin")]
                0 references | 0 requests | 0 exceptions
26          public IActionResult About()
27          {
28              ViewData["Message"] = "Your application description page.";
29
30              return View();
31          }
32
```

Index View

```
@{ ViewData["Title"] = "Home Page"; }

<div class="container">
    <div class="row">
        <div class="col-md-12">
            <h2><strong>Home Page </strong></h2><br /><br />
            Hello @User.Identity.Name !, Role @User.FindFirst(claim => claim.Type == System.Security.Claims.ClaimTypes.Role)?.Value
            <a asp-action="logout" asp-controller="account">
                Logout
            </a>
            <br />
            <br />
            <h4>Welcome to Asp.Net Core Authentication and Authorization Demo!!</h4>
        </div>
    </div>
</div>
```



Logout

```
0 references | 0 requests | 0 exceptions
public IActionResult Logout()
{
    var login = HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToAction("Login");
}
```