Curtin University

Distributed Computing: Beyond RPC

Distributed Computing(COMP3008)

Discipline of Computing School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

Let’s introduce some other concepts you’re going to need

Components

- RPC is a great IPC mechanism….
- But it’s only a fix for a problem, namely, how to communicate.
- The real question is, what do you distribute?
- Where are you going to put what parts, and why?
- Components are the defacto solution to this problem

What are components?

A component is a set of functions that work together.

It is also the smallest chunk of an application that can be moved to a remote server.

Essentially provides “a service” to other components.

This is vaguely related to the concept of prefabrication.

Components and Objects

- Components are defined by the interface it exposes to external clients This is very similar to how objects can be defined by their interface to other objects.
- This makes OO style programming unreasonably effective in building components.
- This is one of the main reasons we're using C#.

OO Components

- Components are the interface for external clients, and link with the application dynamically via the operating system.
- Objects are the internal implementation of the component systems and are linked statically by the compiler.
- These objects can then have further interfaces to allow internal object systems to interact with the external clients via the OO interfaces.

Components, Objects, and RESTing

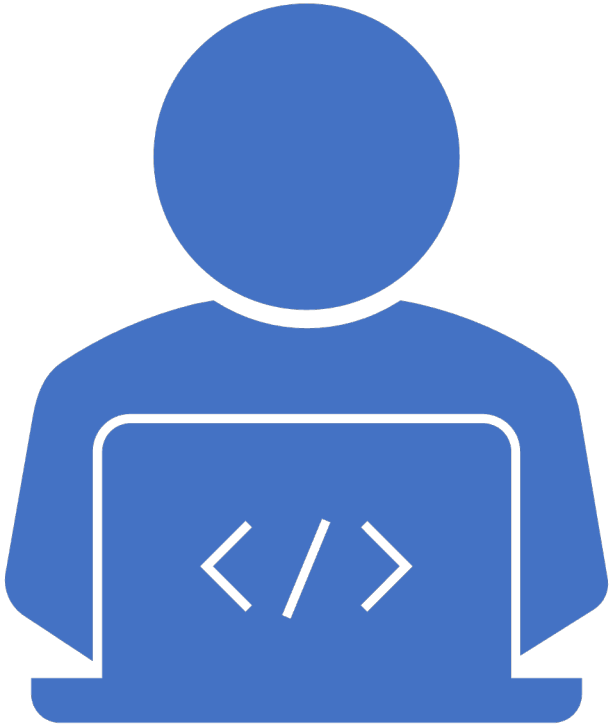
- These two concepts are very similar:
  - Component: Service-oriented, provides methods for clients
  - Object: Data-oriented, methods transform its own state.
- Components represent the services a server can perform; objects represent things in a simulated system.
- This is blurred significantly by representational state transfer services (services that produce and consume objects).

Components, Objects, and RESTing

- Example:
  - Broker.DoTransaction() is a service and so likely a function in a component.
  - Transaction.SetBroker() is a mutation and so likely a function in an object.
  - Both of these are valid actions in a modern REST based web service!
    - Depending on how you've built it of course.

Focusing on Objects and Components

- They both:
  - Define public interfaces for clients
  - Can inherit interfaces from other objects/components
  - Encapsulate data (and can provide information hiding)
  - Define a cohesive set of actions and variables that can fulfil a role in the application
  - Are reusable (theoretically).



Differences

- Objects
  - Technically specific
  - Are actually source code in the app.
  - Internal, only seen by other objects
  - Compile time linking
  - Implementation is specific and reusable.
- Components
  - Architecturally specific
  - Independent of actual code
  - Exposed to other applications
  - Use run time linking
  - Implementation details are obtuse

Components and RPC

- Put these together and you have the basis for modern distributed computing
  - Components are easy to think about, Objects naturally snap on to their interfaces.
- However this introduces many problems
  - First among these; The network is not a stable medium like RAM.
  - What are the consequences of spreading state over a network?

Networking: Making Lan Parties Hard Since 1999.

- When you're locally based, errors occur because you did something wrong.
- When you're on the network, things go wrong because of the network.
  - Network fails, no program for you
  - Server fails, no program for you
  - Local configuration fails, no program for you
- Big problem is figuring out which of these happened, and when.

*Nix vs Windows vs OS2

- Another major (extremely major) problem is that of varying architectures.
- A Linux program and a Windows program are not binary compatible.
- Tools are very often walled into an ecosystem
- Unearthly hackery often resulted.

The Service Layer

- The current way that operating systems work is via something called Service Oriented Architectures.
- These have become so pervasive that all operating systems (that matter) currently have
- Services inbuilt
  - Sorry Unix fanboys, Systems is actually a result of this.

What is an SOA?

- Services are components that clients connect to for execution of a task
  - For example, the printing service, or the login service.
- They cannot be passed around and are instead resident (clients have to go to them).
- This changes the mentality of designing a distributed system (and a system altogether).

Why?

- Well, have you ever tried to pass a pointer to something physically in memory on another machine?
- What if the objects that comprise your object are on another machine, and that machine cannot be accessed?
- Service structures prevent distribution by chunking off sets of objects by defining distinct siloed behaviours via groups of components.

The Evolution of Services

- Services are great
  - Lower coupling over networks
  - Less implicit trust between components
  - Improved security due to less exploitable pathways
- Services are about 20 years old now.
  - Are fundamental to operating systems
  - But have brought the Web into our computers
  - Object orientation has suffered
    - Objects are now second-class citizens
    - Good OO design is now actively eschewed

CORBA (Common Object Request Broker Architecture)

- CORBA is built around the idea of Object Request Brokers (or ORBs)
- ORBs are a middleware service that allow languages to communicate over the network
- ORBs are designed to be cross compatible, regardless of architecture or underlying language.
- ORBs are represented as objects, which allows the system to hide nasty code inside classes.

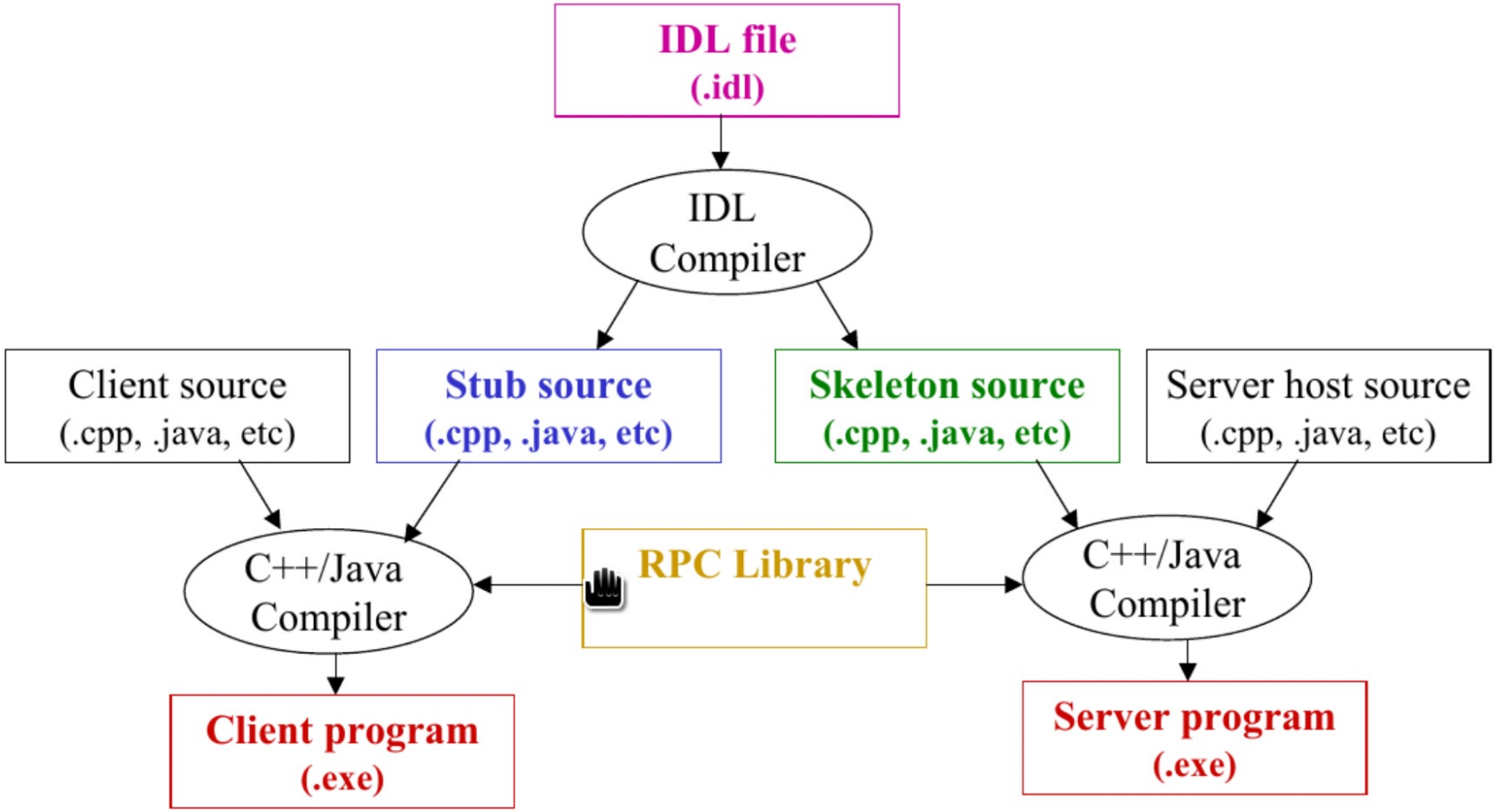
CORBA

- Why was this all so cool?
  - There was still no standardised format for passing data around the internet.
  - CORBA provided one that was language and system independent.
  - It also provided a language independent means of writing the interfaces, meaning clients and servers were implementation independent!
  - CORBA is still around today (although not popular)

CORBA IDL

```
module Utilities
{
   // Basic example interface
   interface Calculator
   {
      int Add(in int operand1, in int operand2);
      double Add(in double operand1, in double operand2);
   }

   // Interface inheritance
   interface ScientificCalculator : Calculator
   {
      // Returning values by the parameter list
      void SolveQuadratic(in float a, in float b, in float c, out float x1,
                                                                  out float x2);

      // Example with a string
      double EvaluateExpression(in string expr);
   }
}
```

CORBA IDL Process



CORBA Today

- Good idea, but there were problems
- Spec was hugely complicated because the ORBs were written by different vendors.
  - Who charged a lot
  - ORBs turned out to not be as interoperable as promised
- Competing less expensive frameworks killed off the project
  - Java RMI was free and did the same thing by 1999
  - Also, Microsoft

Java RMI

- Like CORBA, inheritance is used to hide the nasty stuff.
  - Server object inherits from UnicastRemoteObject
  - Again, no IDL class required.
- Java also has a name service for finding components
  - Called rmiregistry.
  - It’s a command line program.
- Problem: RMI has no inbuilt security integration.

Java RMI Interface

```
import java.rmi.*;

public interface Calculator extends Remote {
    int Add(int operand1, int operand2) throws RemoteException;
}
```

Java RMI Server

```
import java.rmi.*;

public class CalculatorImpl extends UnicastRemoteObject ← RMI base object
                            implements Calculator    ← Interface class
{
   public int Add(int operand1, int operand2)                ← Actual interface function
                               throws RemoteException {
      return operand1 + operand2;
   }
}
-------------------------------------------------------------------------------
import java.rmi.*;

public class CalcServer {
   public static void Main(String[] args) {
      if (System.getSecurityManager() == null)
         System.setSecurityManager(new RMISecurityManager());  ← Need .policy file for this

      CalculatorImpl calc = new CalculatorImpl();             ← Create calculator server
      Naming.rebind("Calculator", calc);            ← Define name for name service lookups

      System.out.println("Press Enter to exit");
      System.in.readln();                           ← Wait for client requests
   }
}
```

Java RMI Client

```
import java.rmi.*;
import Calculator.*;                                    ← For server definition

public class CalcClient
{
   public static void Main(String[] args) {
      if (System.getSecurityManager() == null)
         System.setSecurityManager(new RMISecurityManager());   ← Need .policy file for this

      sURL = "rmi://localhost/Calculator";
      Calculator calc = (Calculator)Naming.lookup(sURL);   ← Attaches to Calculator object on
                                                              remote server (the Calc object is
                                                              already created)

      System.out.println("1 + 2 = " + calc.Add(1, 2));
   }
}
```

Fun fact about RMI

- Java RMI's biggest problem is that it is super tightly integrated with Java
- For example:
  - The RMI client doesn't have the stub code for the server.
  - Instead, it downloads it from the server on first connect.
    - Both versions of Java must be the same.
    - This has implications for security too, as it must trust the code it downloads.

Java RMI Today

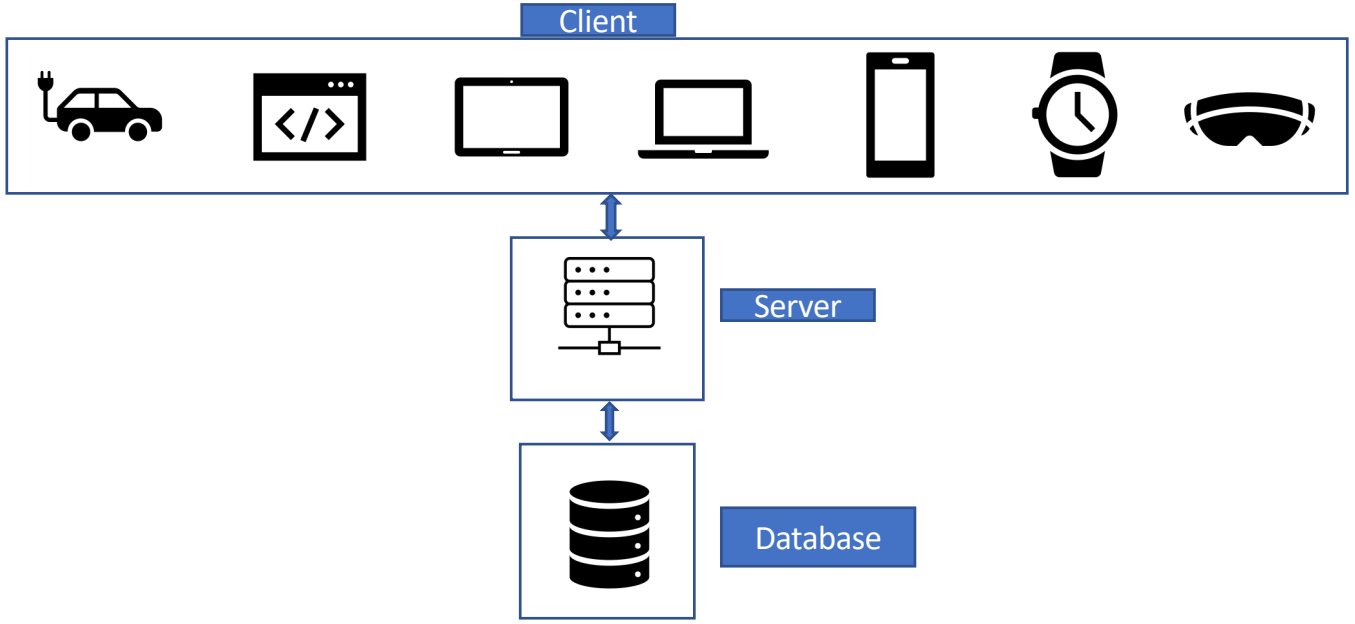
- Java RMI is still used today
- It works pretty well, and provides an all-in-one, no frills approach to component distribution.
- The only problem is, it's Java.
  - And therefore, kind of stands alone.

Windows Communication Foundation

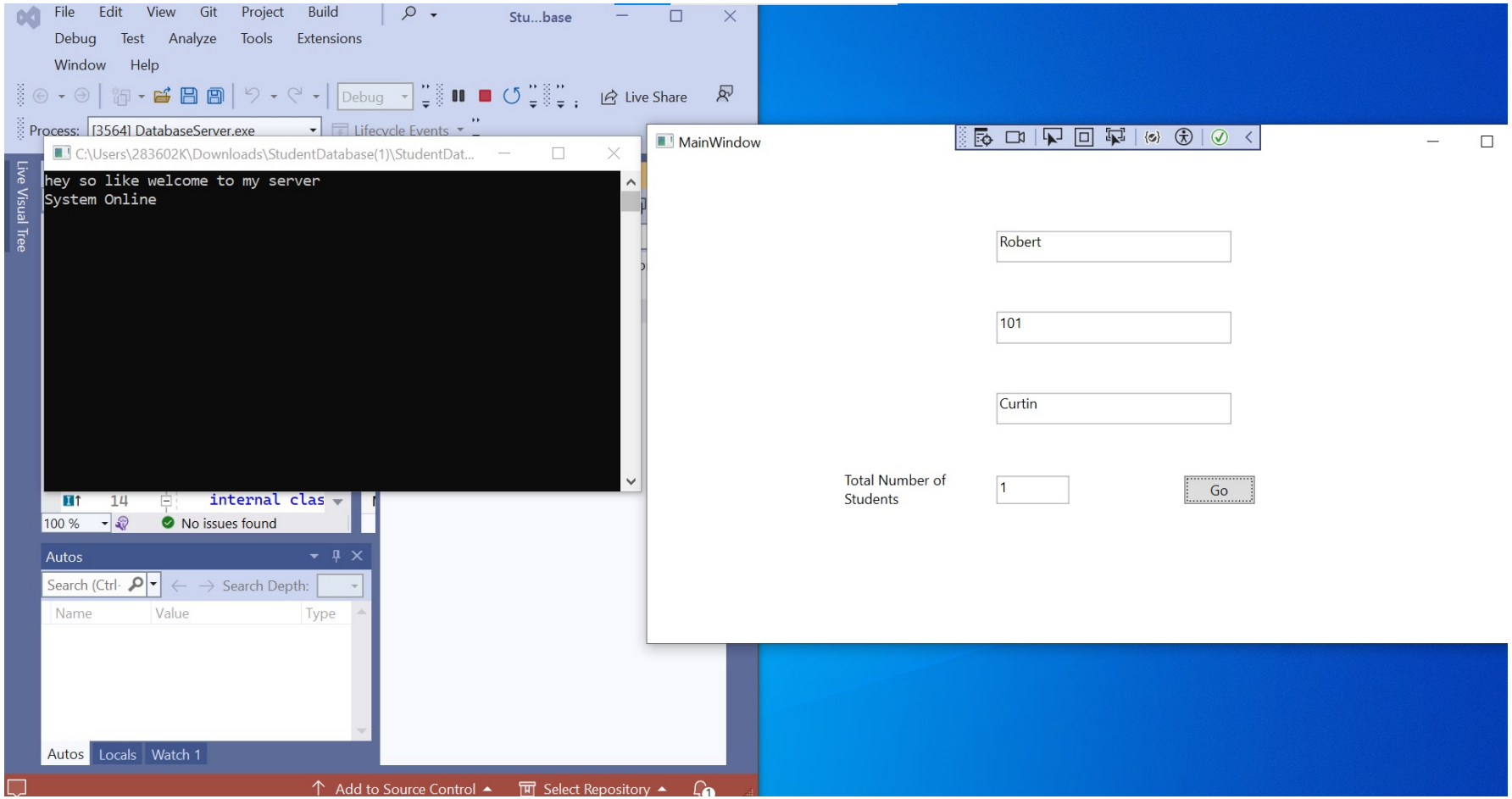
Windows Communication Foundation (WCF) is a powerful and flexible framework provided by Microsoft for building service-oriented applications. It is a part of the .NET Framework and is designed to facilitate the creation of secure, reliable, and interoperable distributed systems.

Creating a distributed application in .NET WCF

- Common elements – Database library, server and client



Our Example: Student Information System



What is DLL?

DLL stands for "Dynamic Link Library." It is a file format used in Microsoft Windows operating systems to store collections of code and data that can be used by multiple programs simultaneously.

Code Reusability: DLLs promote code reusability and modularity. Multiple applications can use the same DLL to access specific functions, reducing redundant code and saving memory space.

Dynamic Linking: The term "dynamic" in DLL refers to the fact that the linking (connecting the code) happens at runtime, not at compile time. When an application needs to use a function from a DLL, it links to the DLL dynamically during program execution.

Memory Efficiency: Since DLLs are shared among different applications, they can be loaded into memory once and used by multiple processes. This can lead to more efficient memory utilization.

DLL (What Windows say..)

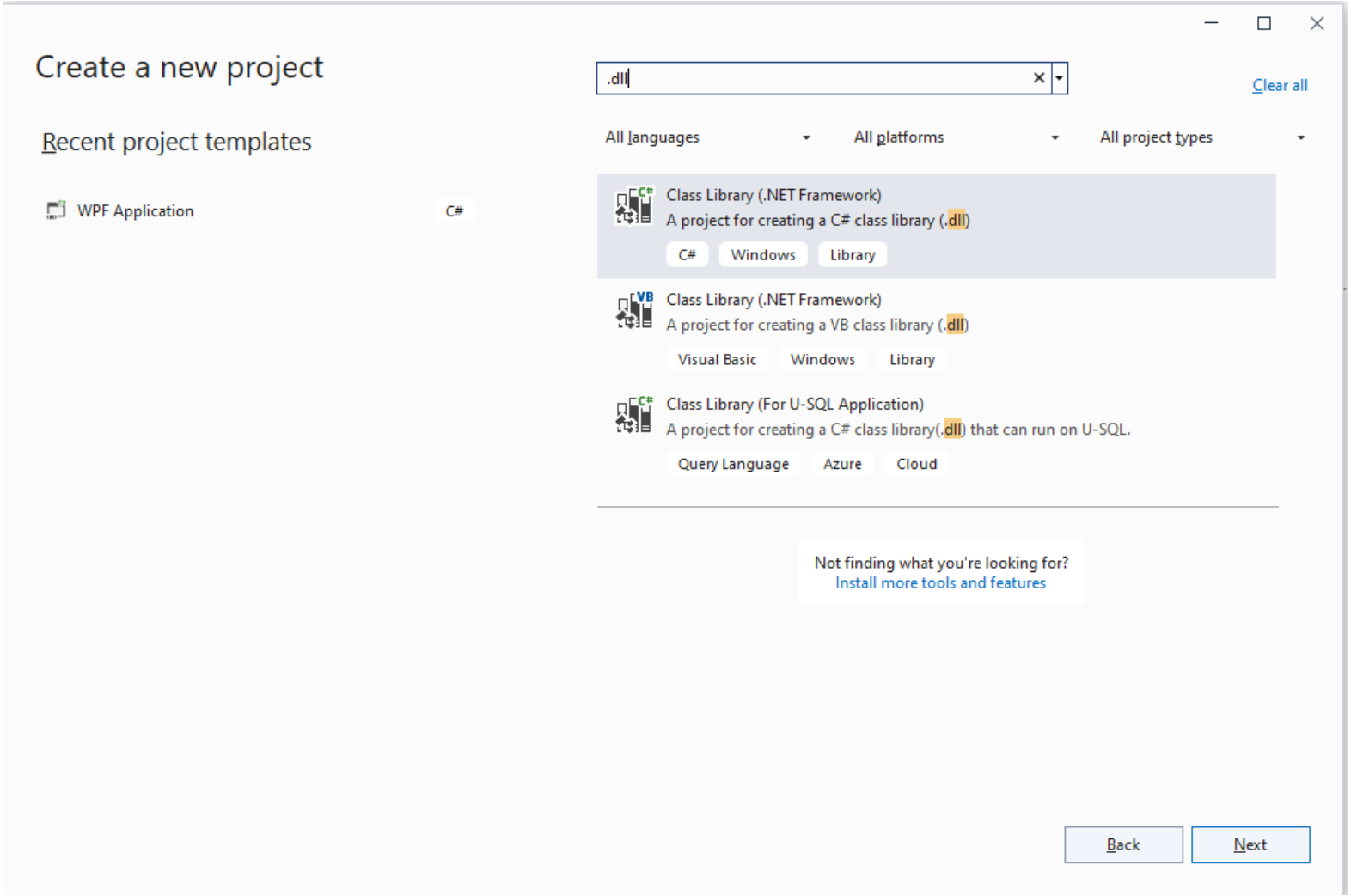
- https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

A DLL is a library that contains code and data that can be used by more than one program at the same time. For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions. Each program can use the functionality that is contained in this DLL to implement an **Open** dialog box. It helps promote code reuse and efficient memory usage.

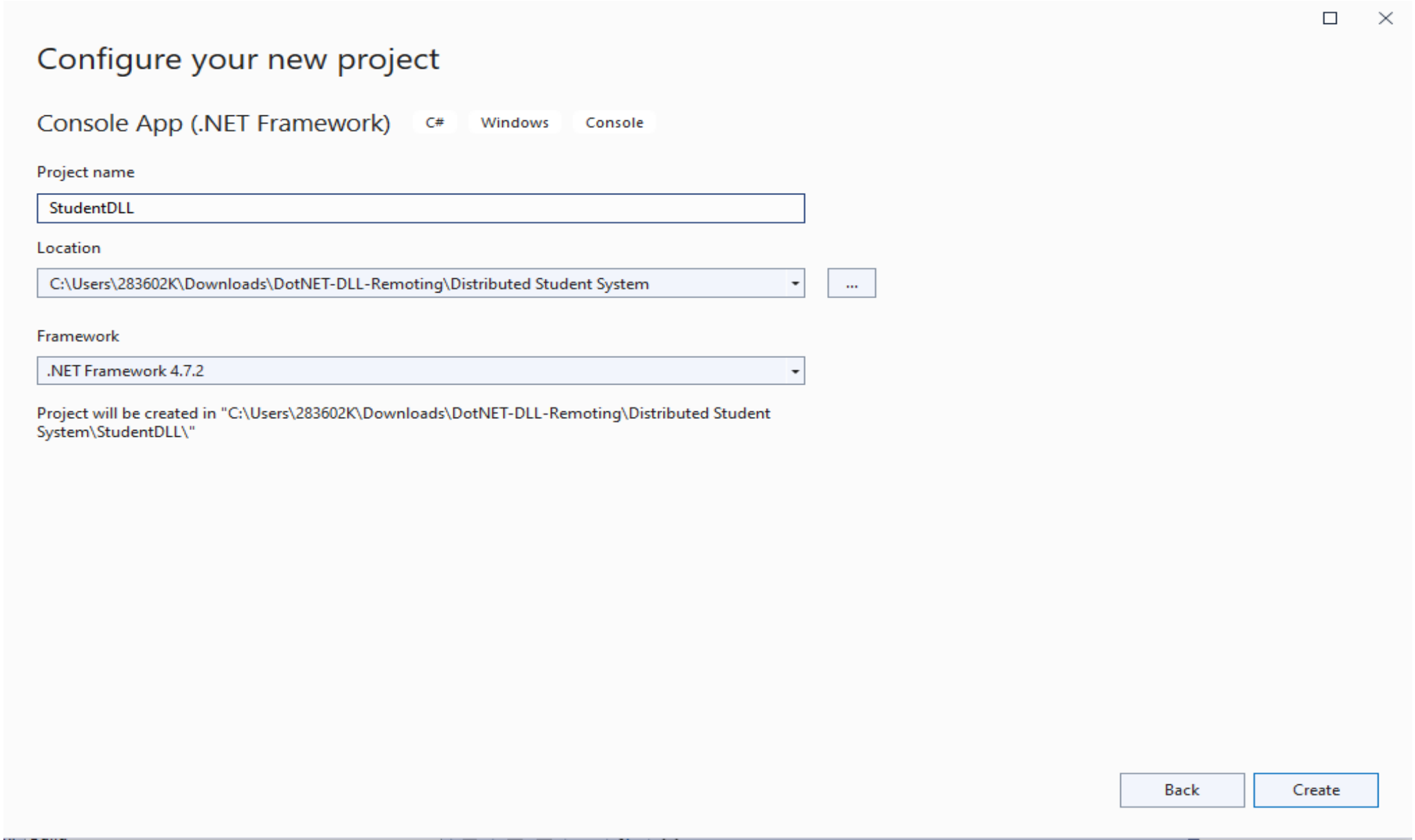
By using a DLL, a program can be modularized into separate components. For example, an accounting program may be sold by module. Each module can be loaded into the main program at run time if that module is installed. Because the modules are separate, the load time of the program is faster. And a module is only loaded when that functionality is requested.

Additionally, updates are easier to apply to each module without affecting other parts of the program. For example, you may have a payroll program, and the tax rates change each year. When these changes are isolated to a DLL, you can apply an update without needing to build or install the whole program again.

Building a simple DLL

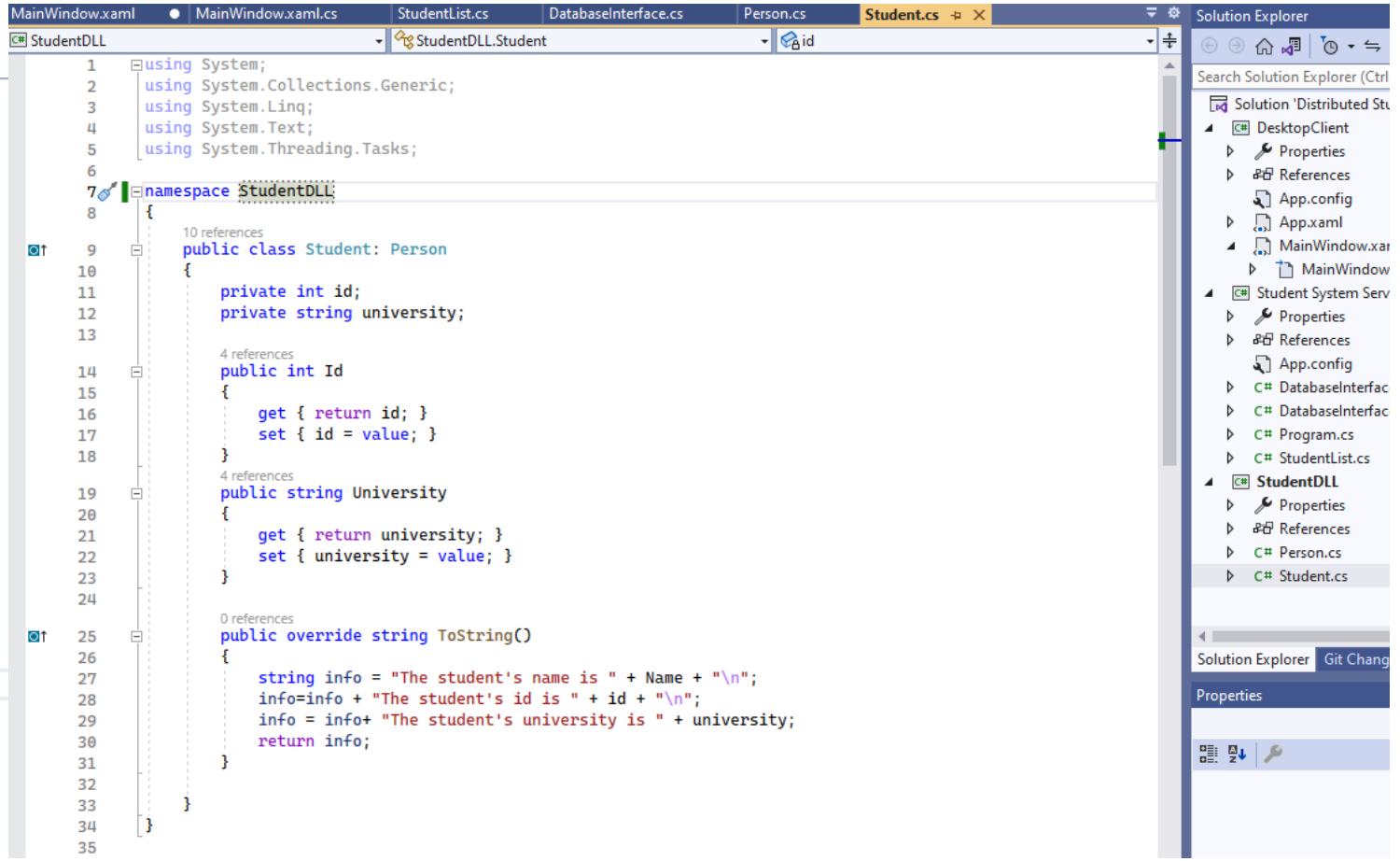


Building a simple DLL (cont..)



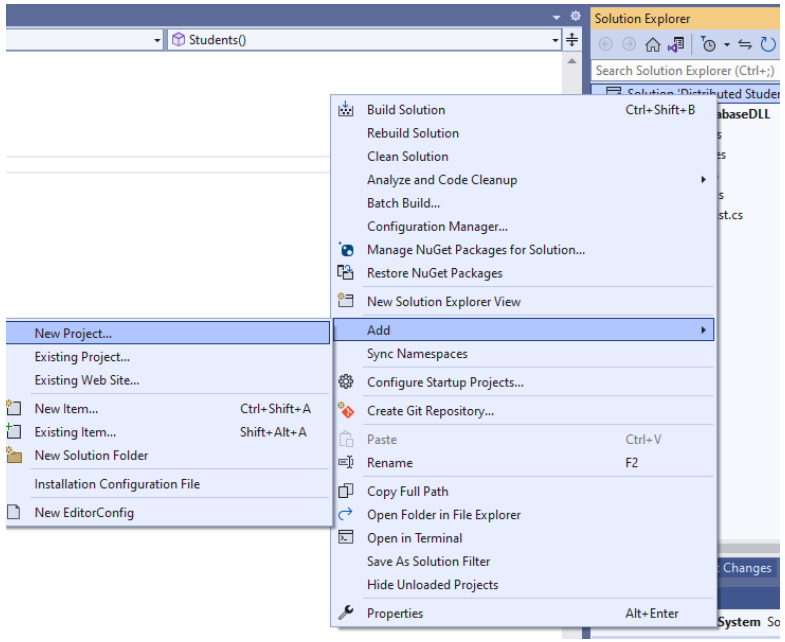
Adding the classes

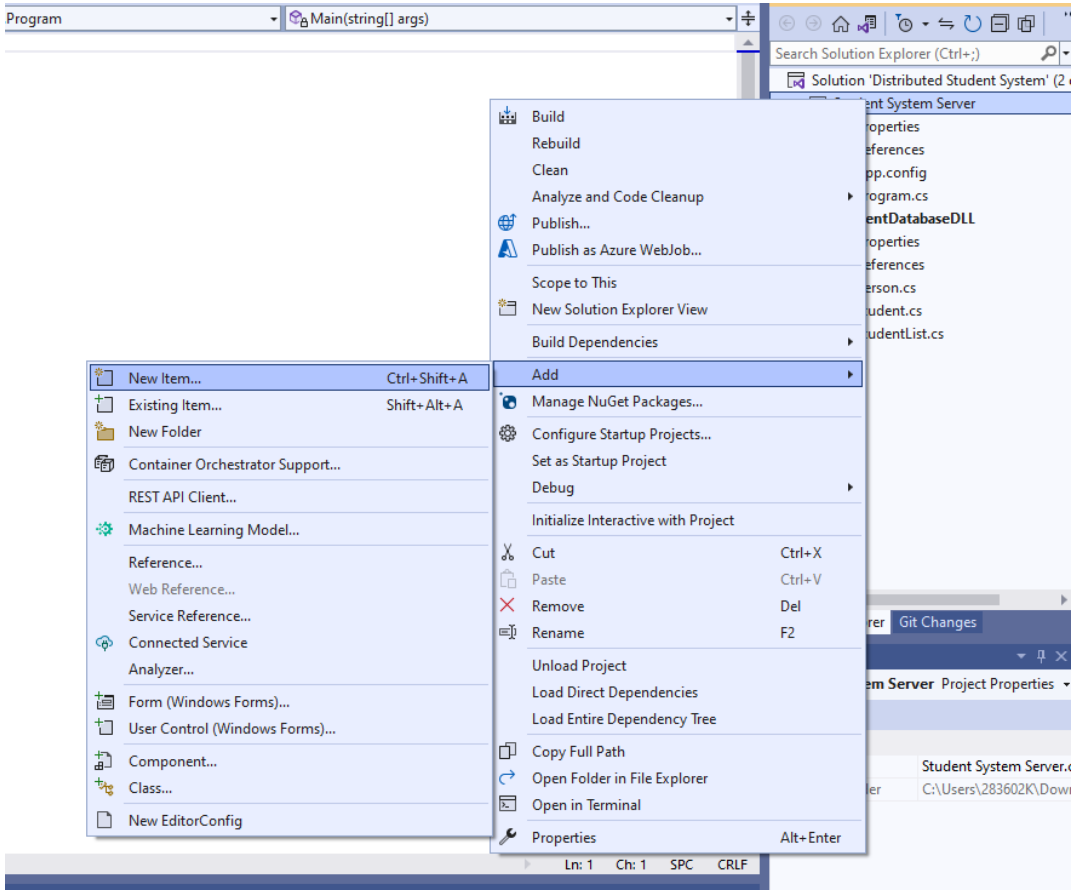
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StudentDLL
{
    1 reference
    public class Person
    {
        private string name;
        5 references
        public string Name {
            get { return name; }
            set { name = value; }
        }
    }
}
```

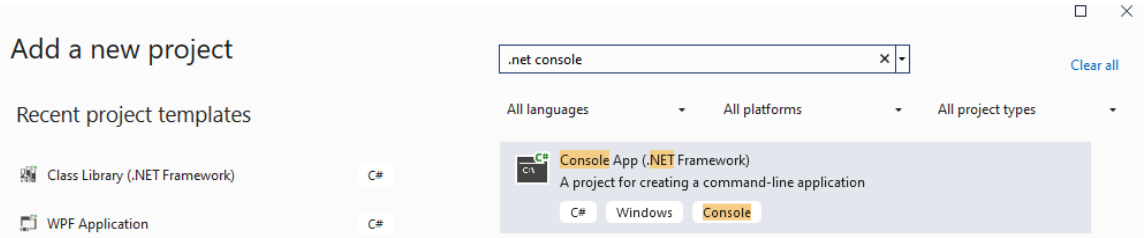


Building the Server

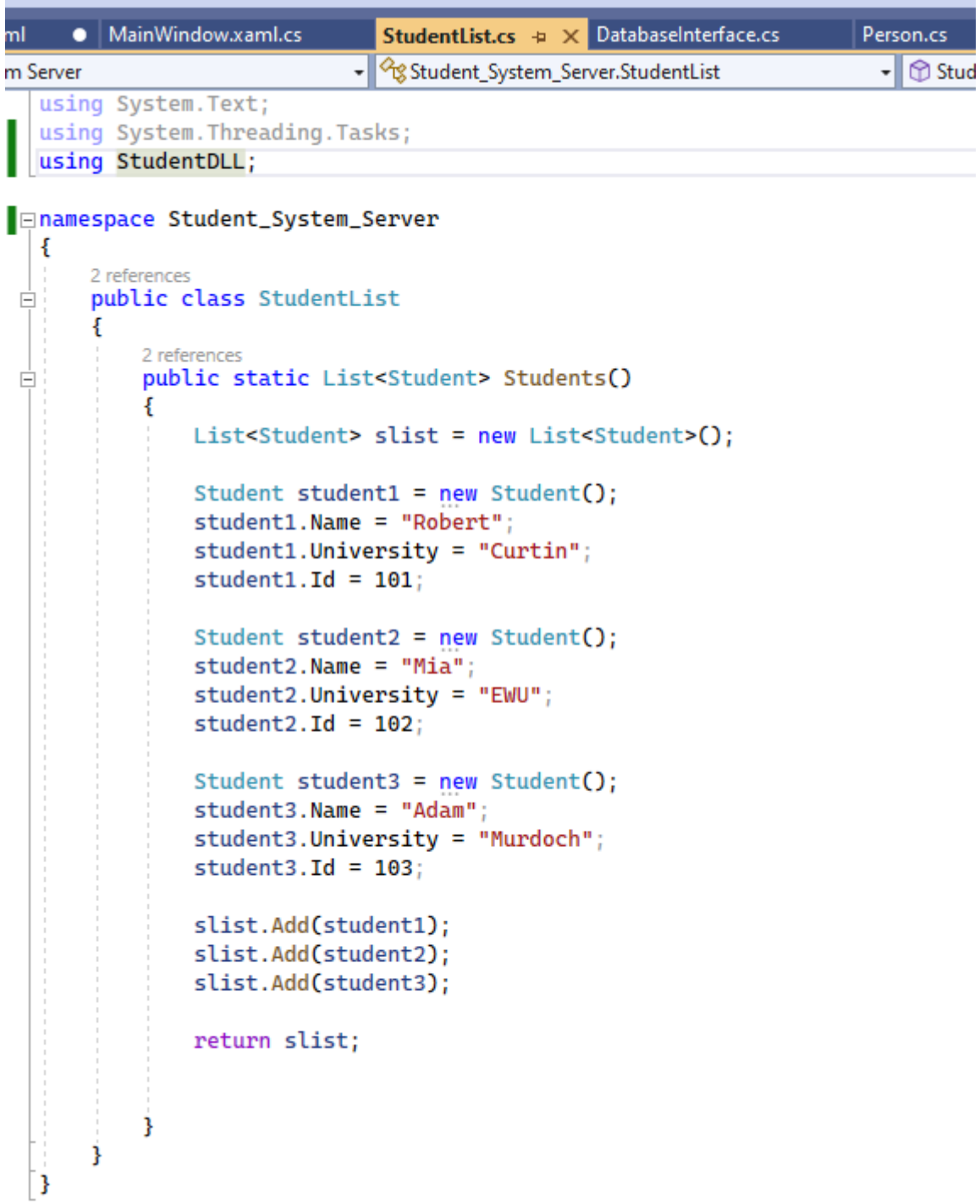
- It will be a console application



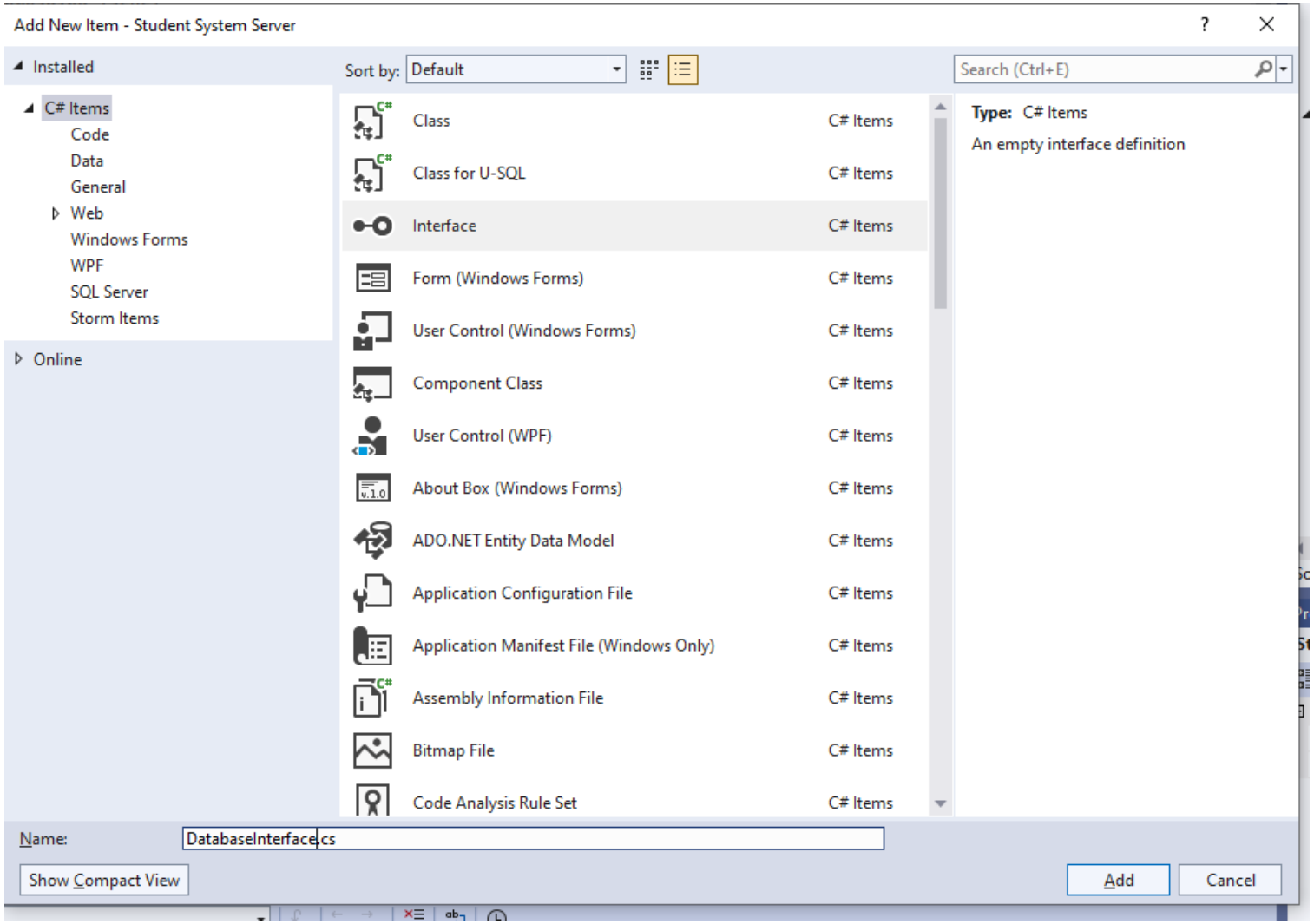


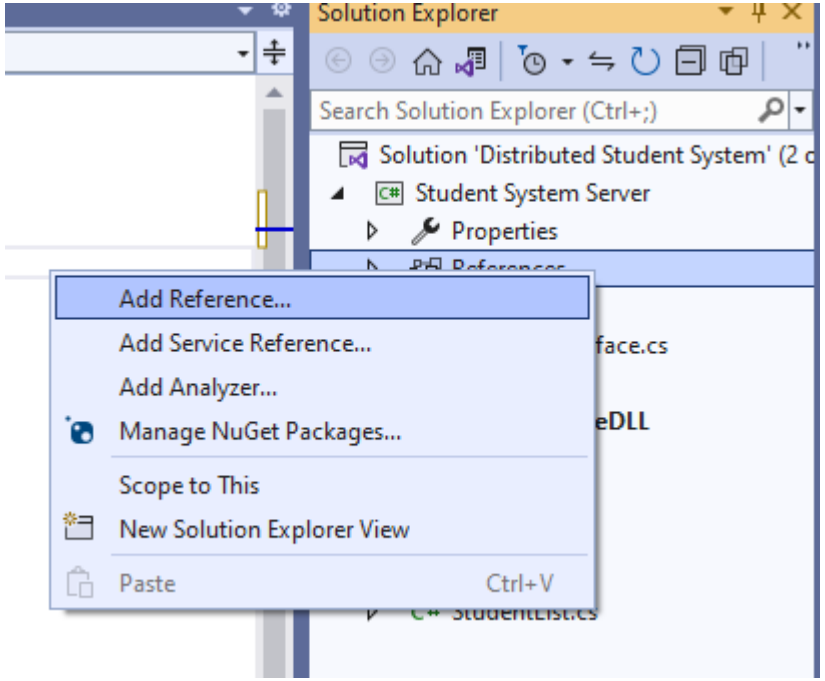


- Add reference to “StudentDLL”
- Create StudentList
  as the database

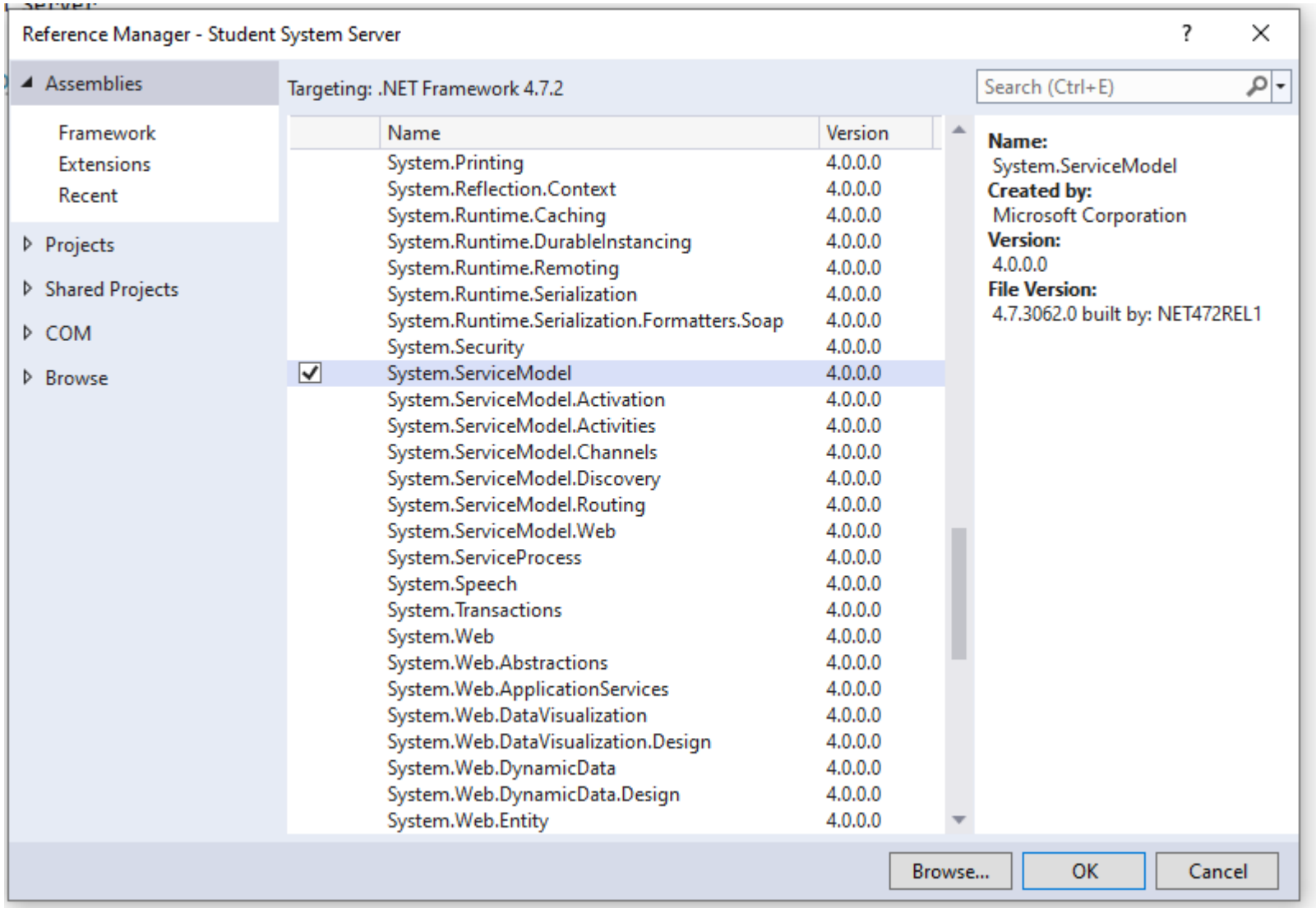


We want to reference System.ServiceModel !!





It is in Assemblies

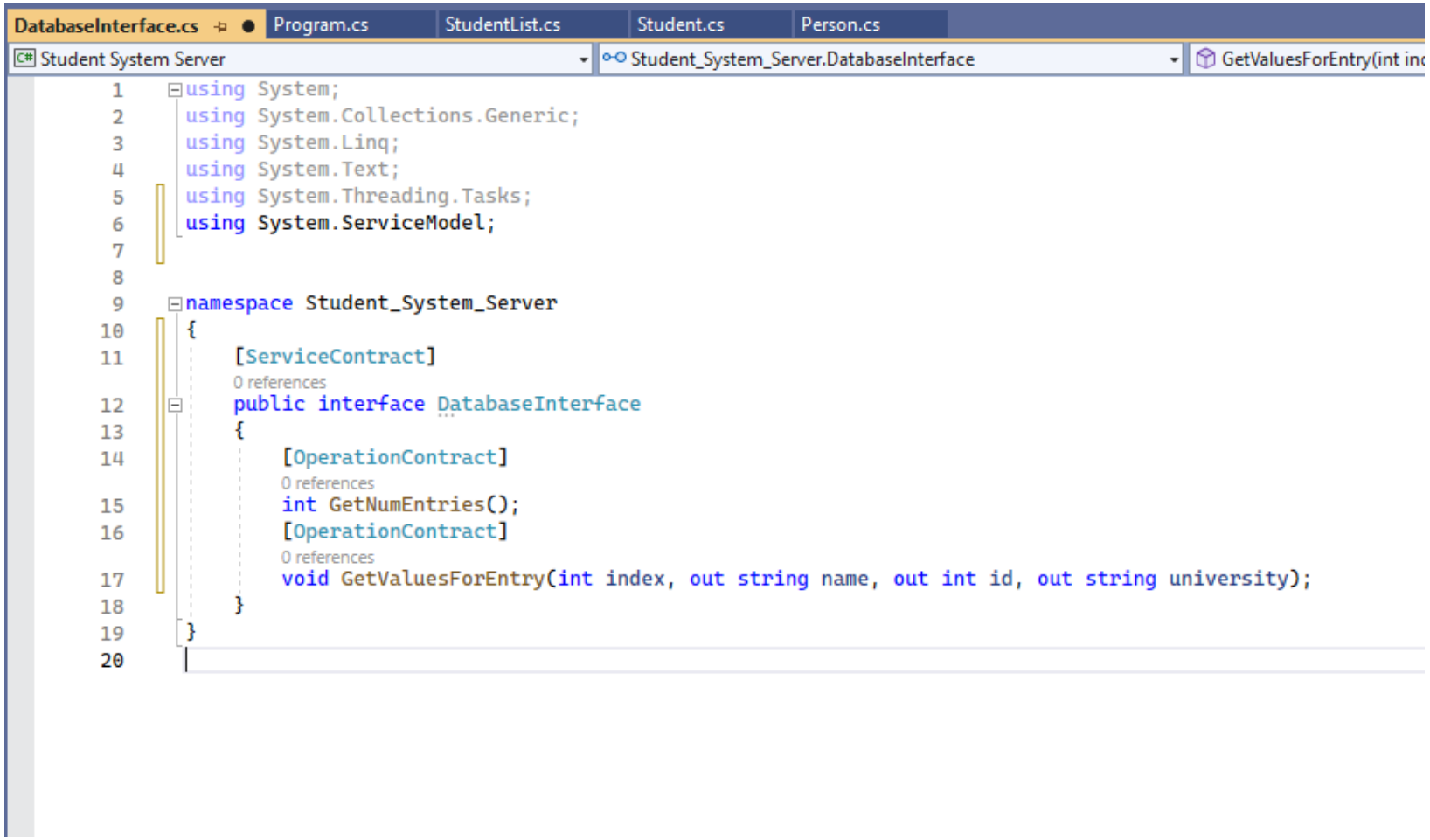


Service Contract and Operation contract in Windows Communication Foundation (WCF)

The service contract is an interface that defines the operations (methods) exposed by the WCF service. It acts as a contract between the service and the clients, specifying what operations the service provides. The clients use this contract to communicate with the service and invoke its methods.

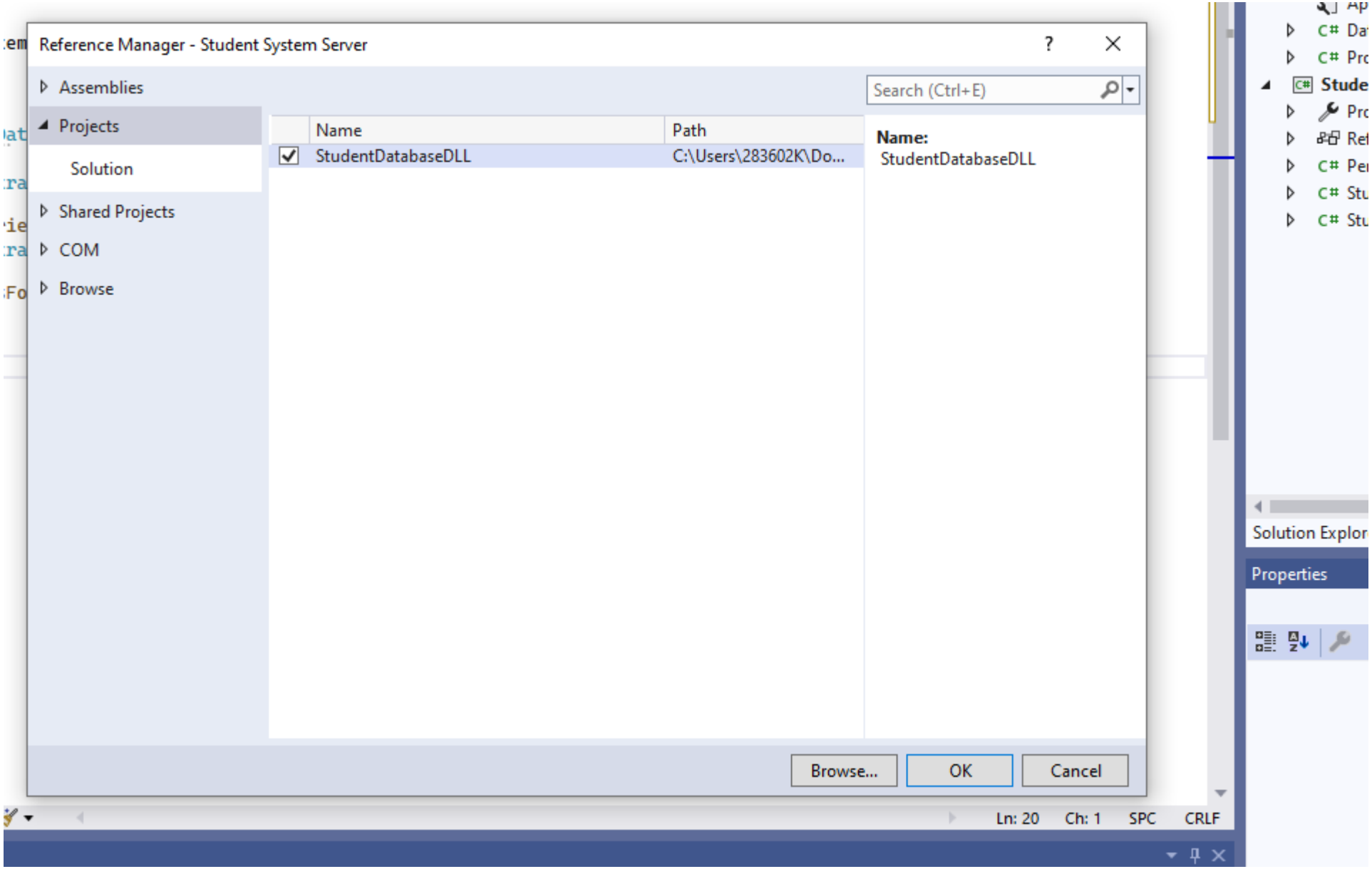
The operation contract is a method in the service contract that specifies a particular operation that the service can perform. It represents the individual service methods exposed to clients. Each operation contract is decorated with the attribute.

Service Contract in .NET WCF



- *Did you notice the "out" keyword??*

Linking the DLL as a reference



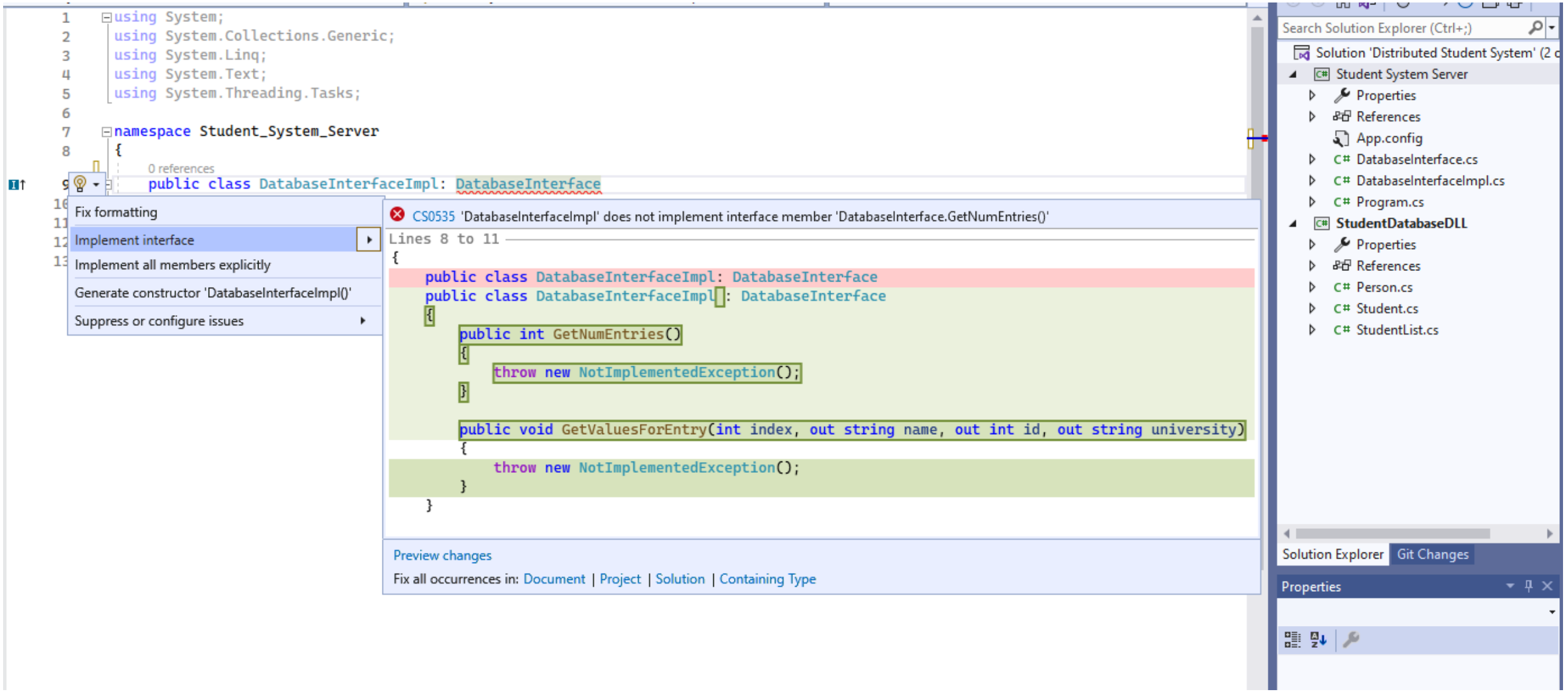
Service behavior in WCF

- In Windows Communication Foundation (WCF), the "ServiceBehavior" attribute is used to customize and configure the behaviour of a WCF service.
- It allows you to specify various service-level settings, such as concurrency mode, instance mode, transaction settings, error handling, etc.
- The ServiceBehavior attribute is applied to the service class and provides a way to control the service’s runtime behaviour.

Service behavior options in WCF

- **ConcurrencyMode**: This property specifies how the service instance handles multiple client requests concurrently. The available options are:
  - ❖Single: Only one request is processed at a time on the service instance (no concurrency).
  - ❖Multiple: The service instance can process multiple requests concurrently using multiple threads.
- **IncludeExceptionDetailInFaults**: When set to true, this property causes WCF to include the exception details in the fault message returned to the client in case of an unhandled exception.
- **UseSynchronizationContext**: When set to true, this property enables synchronization with the current SynchronizationContext, which is typically used in UI applications to marshal the callbacks to the UI thread.

Implementing the interface



Defining service behavior

DatabaseInterfaceImpl.cs | DatabaseInterface.cs | Program.cs | StudentList.cs | Student.cs | Person.cs

Student System Server | Student_System_Server.DatabaseInterfaceImpl | GetNumEntries()

```
1   using StudentDatabaseDLL;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.ServiceModel;
6   using System.Text;
7   using System.Threading.Tasks;
8
9   namespace Student_System_Server
10  {
11      [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, UseSynchronizationContext = false)]
        0 references
12      public class DatabaseInterfaceImpl : DatabaseInterface
13      {
            1 reference
14          public int GetNumEntries()
15          {
16              return StudentList.Students().Count;
17          }
18
            1 reference
19          public void GetValuesForEntry(int index, out string name, out int id, out string university)
20          {
21              List<Student> slist = StudentList.Students();
22              name = slist[index - 1].Name;
23              id = slist[index - 1].Id;
24              university = slist[index - 1].University;
25          }
26      }
27  }
28
```

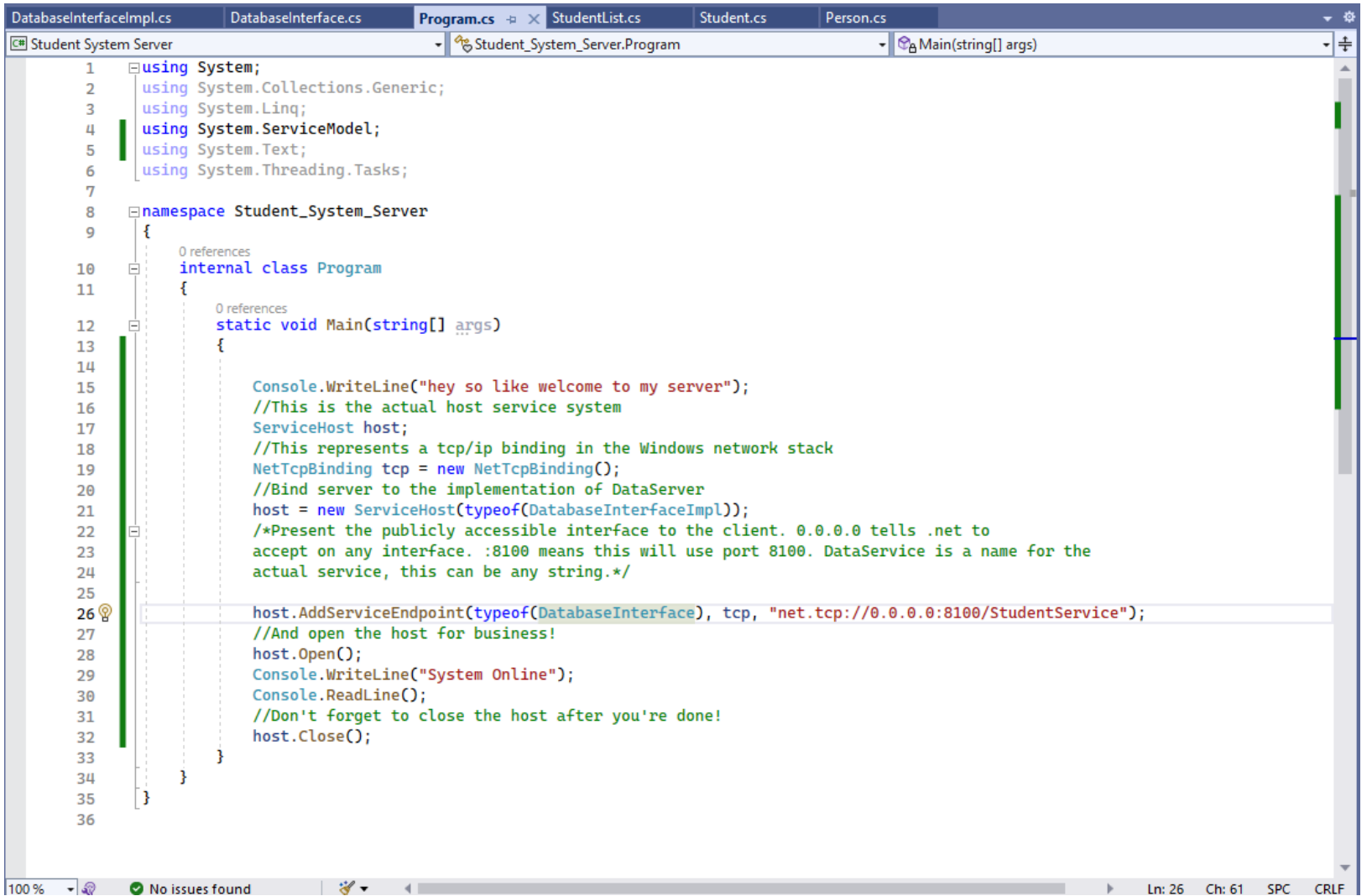
100 % | No issues found | Ln: 11 | Ch: 101 | SPC | CRLF

WCF Bindings

- WCF uses bindings to specify the transport protocol and message format used for communication.
  - ❖BasicHttpBinding: Suitable for interoperable communication with web services using HTTP. It supports SOAP 1.1 messages.
  - ❖WSHttpBinding: Provides more secure and feature-rich communication compared to BasicHttpBinding. It supports WS-Security, WS-ReliableMessaging, and supports both SOAP 1.1 and SOAP 1.2 messages.
  - ❖NetTcpBinding: Suitable for communication over TCP for high-performance scenarios within an intranet environment. It uses binary message encoding, making it more efficient than HTTP-based bindings.

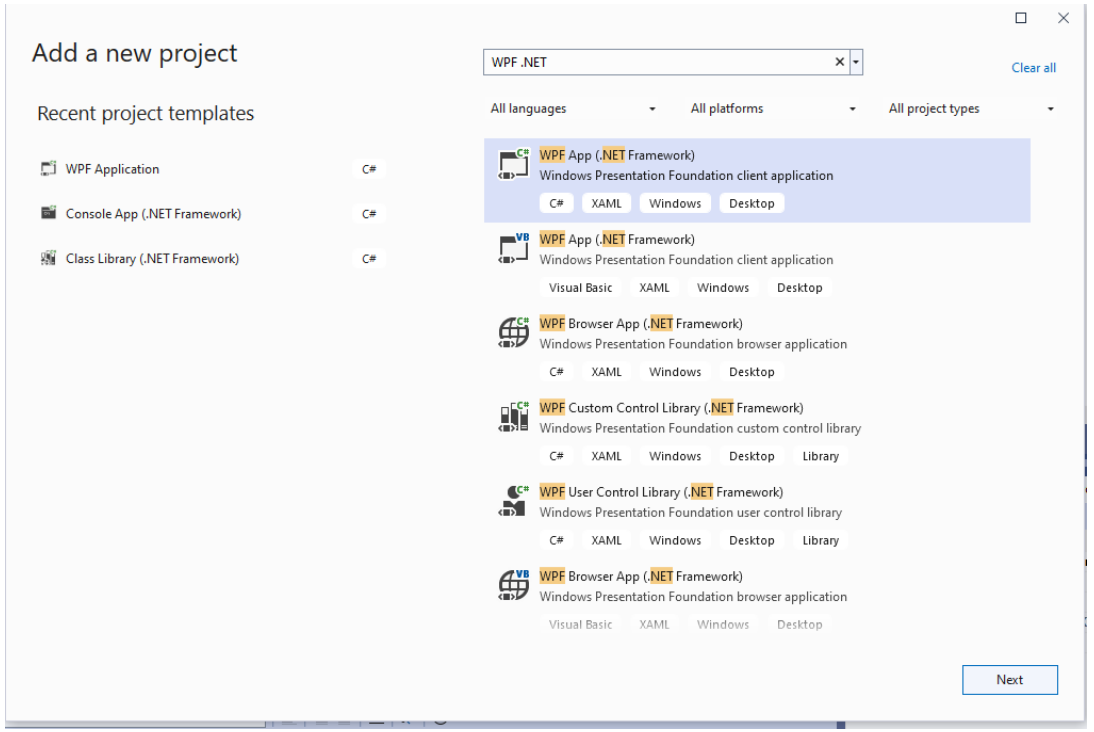
WCF Endpoints

- An endpoint combines a contract, binding, and address to define how clients can access the service.

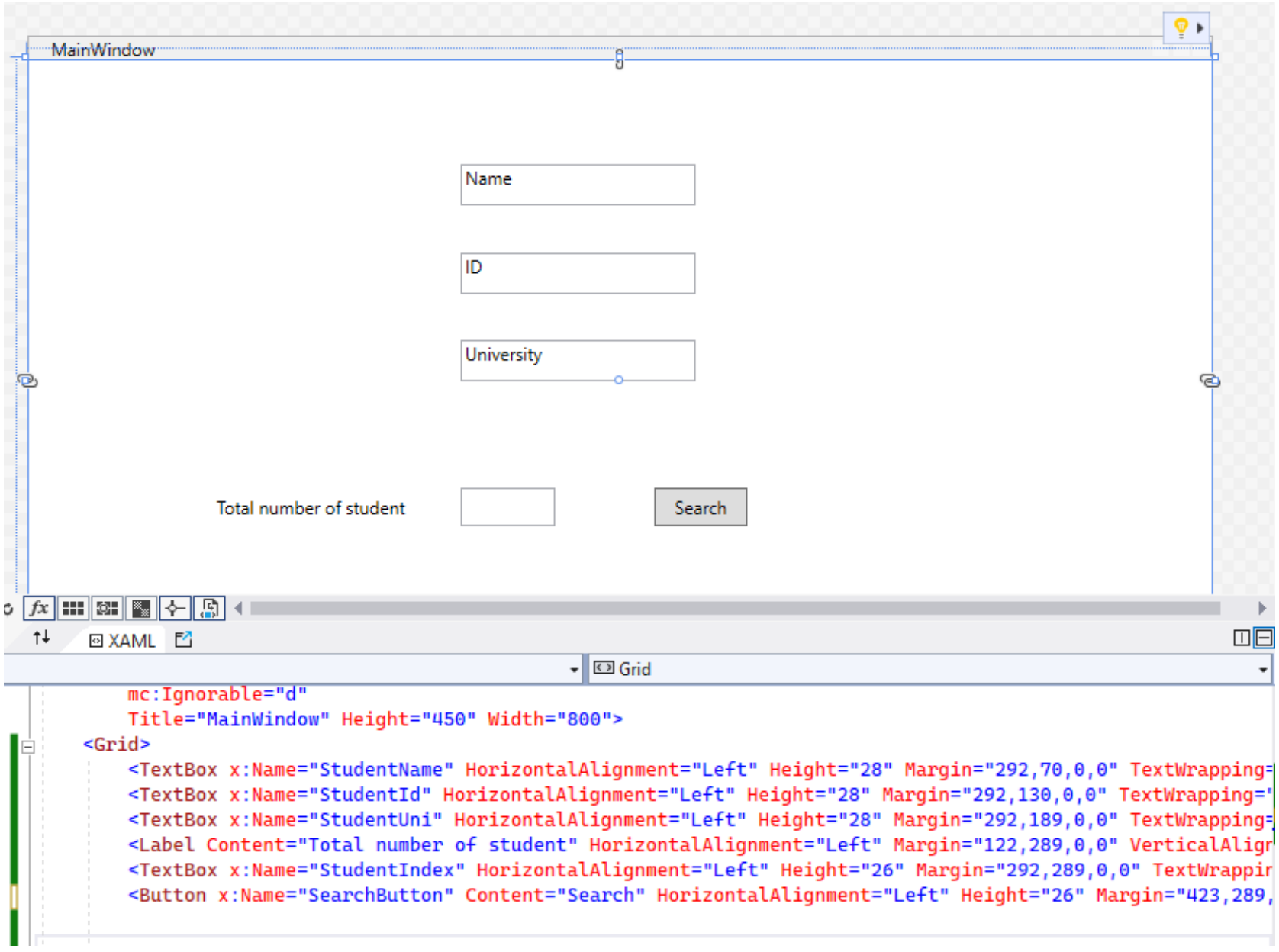


Now building the client

- Create a .NET WPF project (.NET Framework). Choosing the right framework is important!!
- Create a reference to “Student System Server” –Why??
- Create a reference to System.ServiceModel



The user interface

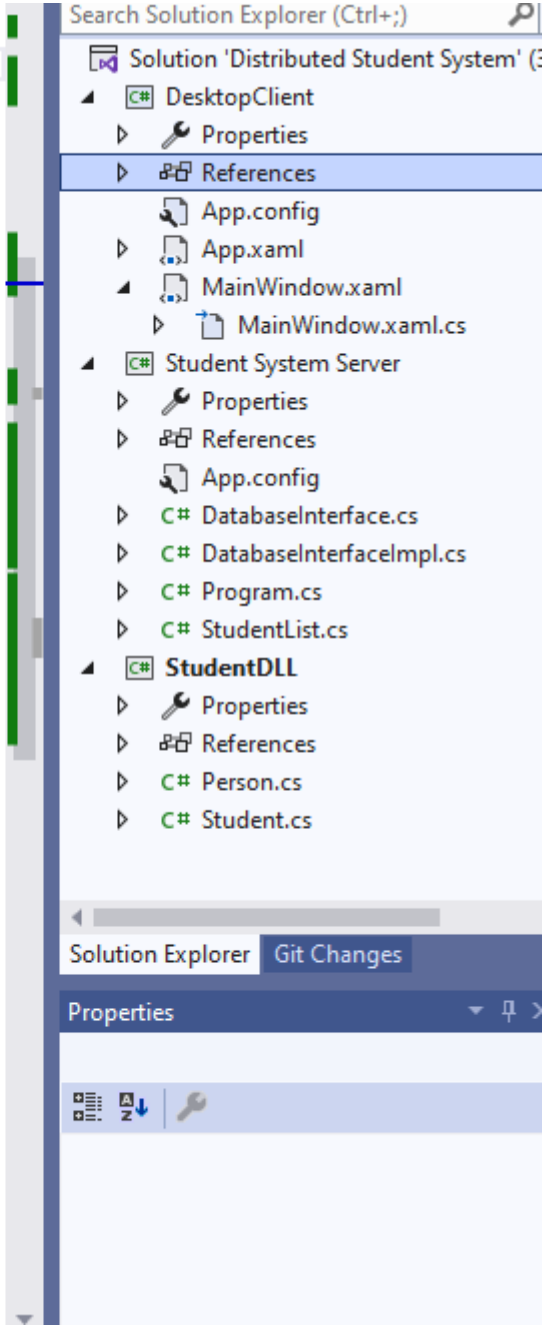


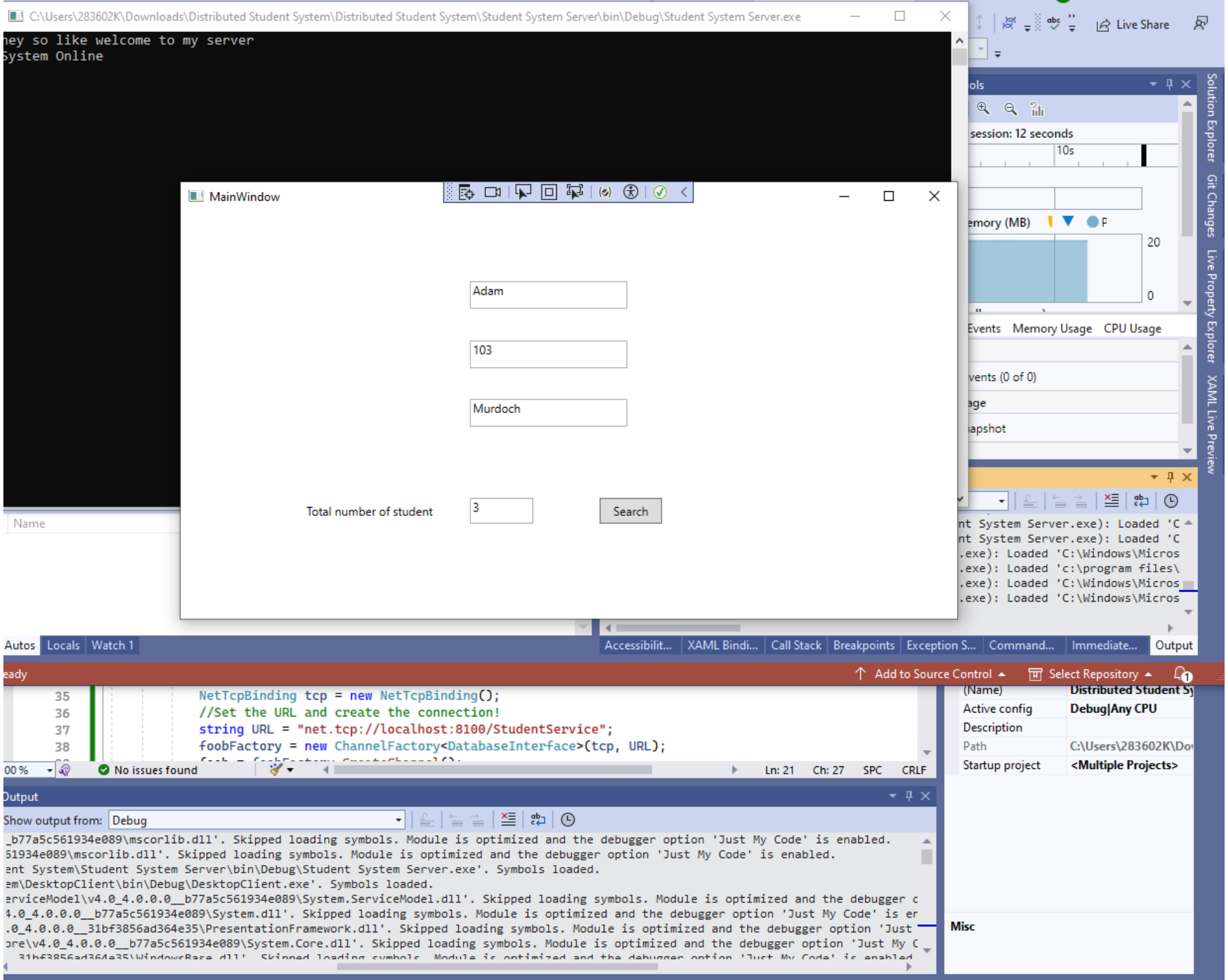
WCF ChannelFactory

The ChannelFactory is a client-side object that facilitates the creation of channels to communicate with a WCF service.
Steps:

1. You need to define the service contract (interface) that matches the service you want to communicate with. This contract should be the same as the one used by the actual WCF service. [That's why we are creating a reference to the server].
2. You need to specify the service contract type and the binding configuration.
3. Once you have the ChannelFactory, you can create a channel that implements the service contract. The channel represents the communication link between the client and the service.

```
using Student_System_Server;
using System.ServiceModel;
namespace DesktopClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {
        private DatabaseInterface foob;
        0 references
        public MainWindow()
        {
            InitializeComponent();

            ChannelFactory<DatabaseInterface> foobFactory;
            NetTcpBinding tcp = new NetTcpBinding();
            //Set the URL and create the connection!
            string URL = "net.tcp://localhost:8100/StudentService";
            foobFactory = new ChannelFactory<DatabaseInterface>(tcp, URL);
            foob = foobFactory.CreateChannel();
            //Also, tell me how many entries are in the DB.
            StudentIndex.Text = foob.GetNumEntries().ToString();
        }

        0 references
        private void SearchButton_Click(object sender, RoutedEventArgs e)
        {
            string name = null;
            int id = 0;
            string universityName = null;
            foob.GetValuesForEntry(Int32.Parse(StudentIndex.Text), out name, out id, out universityName);
            StudentId.Text = id.ToString();
            StudentName.Text = name;
            StudentUni.Text = universityName;
        }
    }
}
```




C:\Users\283602K\Downloads\Distributed Student System\Distributed Student System\Student System Server\bin\Debug\Student System Server.exe
ney so like welcome to my server
System Online
MainWindow
Adam
103
Murdoch
Total number of student
3
Search
Live Share
session: 12 seconds
10s
emory (MB)
20
0
Events
Memory Usage
CPU Usage
vents (0 of 0)
age
apshot
Name
nt System Server.exe): Loaded 'C
nt System Server.exe): Loaded 'C
.exe): Loaded 'C:\Windows\Micros
.exe): Loaded 'c:\program files\
.exe): Loaded 'C:\Windows\Micros
.exe): Loaded 'C:\Windows\Micros
Autos
Locals
Watch 1
Accessibilit...
XAML Bindi...
Call Stack
Breakpoints
Exception S...
Command...
Immediate...
Output
eady
Add to Source Control
Select Repository
35 NetTcpBinding tcp = new NetTcpBinding();
36 //Set the URL and create the connection!
37 string URL = "net.tcp://localhost:8100/StudentService";
38 foobFactory = new ChannelFactory<DatabaseInterface>(tcp, URL);
00 %
No issues found
Ln: 21
Ch: 27
SPC
CRLF
Output
Show output from:
Debug
_b77a5c561934e089\mscorlib.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
51934e089\mscorlib.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
ent System\Student System Server\bin\Debug\Student System Server.exe'. Symbols loaded.
em\DesktopClient\bin\Debug\DesktopClient.exe'. Symbols loaded.
erviceModel\v4.0_4.0.0.0__b77a5c561934e089\System.ServiceModel.dll'. Skipped loading symbols. Module is optimized and the debugger c
4.0_4.0.0.0__b77a5c561934e089\System.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is er
.0_4.0.0.0__31bf3856ad364e35\PresentationFramework.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just
ore\v4.0_4.0.0.0__b77a5c561934e089\System.Core.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My C
(Name)
Distributed Student Sy
Active config
Debug|Any CPU
Description
Path
C:\Users\283602K\Do
Startup project
<Multiple Projects>
Misc
Solution Explorer
Git Changes
Live Property Explorer
XAML Live Preview