

## Ćwiczenie 2

### WCF – podstawy – definiowanie usług i klienta, operacje synchroniczne i asynchroniczne

*Autor: Mariusz Fraś*

#### 1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Poznanie podstawowej architektury aplikacji WCF
  2. Zapoznanie się z podstawami tworzenia usługi z opisem WSDL i dostępnej protokołem SOAP (tu: usługi WCF), i klienta takiej usługi (tu: klienta WCF).
  3. Poznanie opcji konfigurowania usług – punktów końcowych, transportu, sposobu działania usługi.
- **Pierwsza część zadania** jest do wykonania według podanej instrukcji i ewentualnych poleceń prowadzącego zajęcia laboratoryjne.
  - **Druga część zadania** jest do przygotowania i oddania lub do wykonania wg. poleceń prowadzącego na kolejnych zajęciach.

## 2 Zadanie – część I

### Konstrukcja podstawowej aplikacji WCF

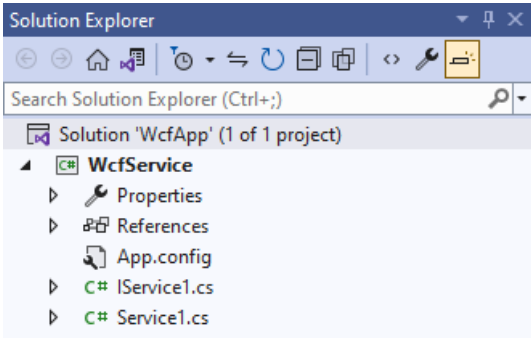
W zadaniu zostanie zrealizowane rozwiązanie obejmujące: **a)** usługę uruchamianą jako odrębna aplikacja i **b)** klienta korzystającego z tej usługi. Usługa i klient zostaną zrealizowane w narzędziu Visual Studio (dalej w skrócie VS) jako usługa WCF. Zadanie realizacji podstawowej aplikacji WCF klient-serwer składa się z kilku etapów:

1. Zdefiniowanie kontraktu usługi.
2. Implementacja kontraktu usługi (implementacja usługi).
3. Utworzenie aplikacji hostującej usługę  
– tu: jest to aplikacja konsolowa – tzw. self-hosting service.
4. Implementacja aplikacji klienckiej (w tym proxy klienta (ang. *proxy client*)).
5. Rozbudowa serwisu i klienta dla operacji asynchronicznych.

**UWAGA:** *zmiany* (np. nazw klas itp.) **w automatycznie wygenerowanym przez platformę kodzie najlepiej realizować poprzez opcję refaktoryzacji platformy**

- Zazwyczaj **opcja Refactor w menu kontekstowym lub odpowiednia opcja w menu kontekstowym** (np. *Rename* – gdy brak jest wyróżnionej opcji *Refactor*).

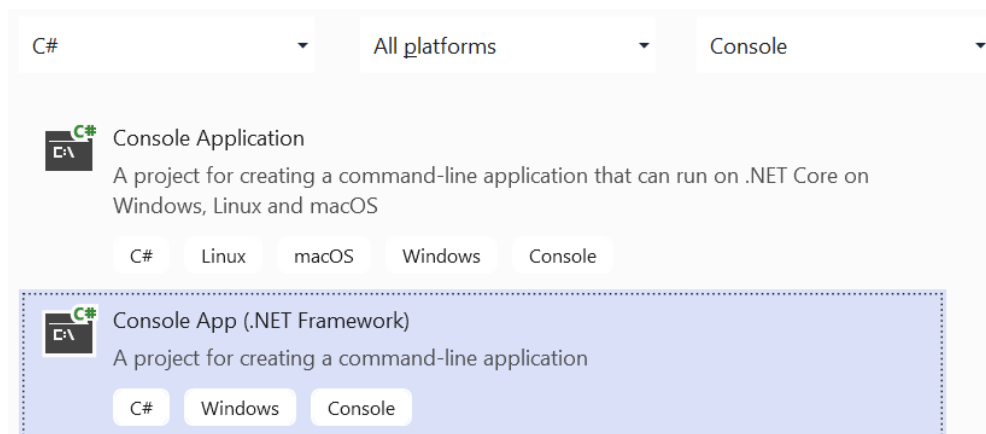
<p>1. Definiowanie kontraktu usługi WCF</p>	<ul style="list-style-type: none"> <li>• Utwórz nową solucję i projekt aplikacji z szablonu <b>Visual C# WCF Service Library</b> nadając własne nazwy solucji i projektowi (tu: WcfApp i WcfService).</li> </ul> <div data-bbox="438 1093 1420 1579"> </div> <ul style="list-style-type: none"> <li>• Przeglądnij zawartość projektu. Zwróć uwagę na następujące elementy: <ul style="list-style-type: none"> <li>○ <b>IService1.cs</b> – plik deklaracji (interfejs) kontraktu,</li> <li>○ <b>Service1.cs</b> – implementacja kontraktu,</li> <li>○ <b>App.config</b> – konfiguracja własności i dostępności implementacji kontraktu.</li> </ul> </li> </ul>
---	---

	 <p><b>Uwaga:</b> nazwy plików mogą się zmienić po zmianie nazwy interfejsu lub klasy z domyślnej (jak wyżej na rys.) na własną nazwę. Takie zachowanie jest konfigurowalne (można to wyłączyć).</p> <ul style="list-style-type: none"> <li>Otwórz plik <b>IService1.cs</b> i zdefiniuj kontrakt usługi – interfejs <b>ICalculator</b> zawierający metody Add, i Multiply: <ul style="list-style-type: none"> <li>Usuń nieużywany kod.</li> <li>Dopisz lub przerób kod do postaci: <pre>[ServiceContract(ProtectionLevel = ProtectionLevel.None)] public interface ICalculator {     [OperationContract]     double Add(double val1, double val2);     [OperationContract]     double Multiply(double val1, double val2); }</pre> </li> </ul> </li> <li><b>Uwaga:</b> tu po zmianie nazwy z <i>Iservice1</i> na <i>ICalculator</i> (opcją menu – nie manualnie!) nazwa pliku może zmienić się w projekcie.</li> <li>Własność (behavior) <i>ProtectionLevel</i> (ustawioną na <i>None</i>) dodano w celu uproszczenia usługi.</li> </ul>
2. Implementacja kontraktu usługi WCF	<ul style="list-style-type: none"> <li>Otwórz plik <b>Service1.cs</b>. Wpisz kod klasy <b>MyCalculator</b> implementującej interfejs <b>ICalculator</b>: Zaimplementuj każdą z wymaganych metod: <pre>public class MyCalculator : ICalculator {     public double Add(double val1, double val2) {         ...     }     public double Multiply(double val1, double val2) {         ...     } }</pre> </li> <li>W miejsce kropek ... dopisz odpowiedni kod dla każdej metody: <ul style="list-style-type: none"> <li>wykonanie odpowiedniego działania,</li> <li>wyświetlenie informacji w konsoli co jest wywołane, co otrzymano w wywołaniu i co jest zwracane,</li> <li>zwrócenie odpowiedniej wartości.</li> </ul> </li> </ul>

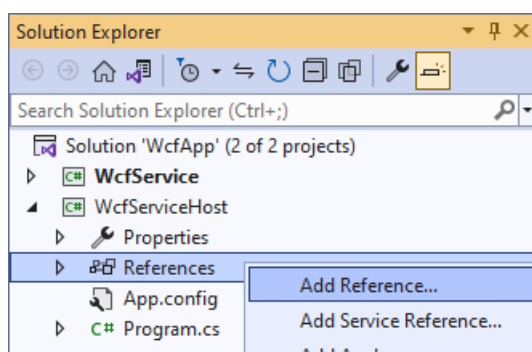
### 3. Hostowanie usługi WCF

Utwórz aplikację konsolową hostującą usługę WCF (Host usługi).

- Dodaj do dotychczasowej solucji drugi projekt aplikacji konsolowej nadając mu własną nazwę (tu: WcfServiceHost)  
– opcja: **Add... → New Project.**



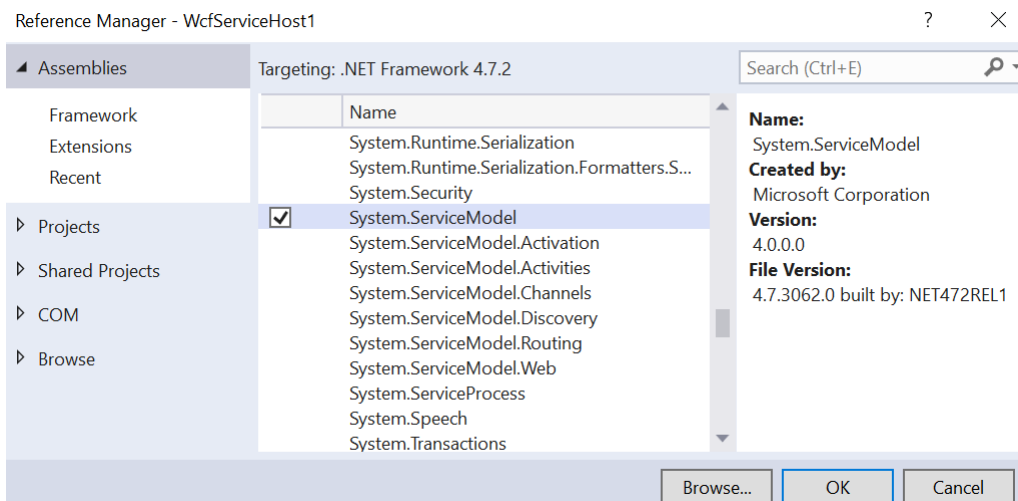
- Sprawdź (i ewentualnie ustaw) wersję Framework'a aplikacji.
  - W **Solution Explorer** menu kontekstowe Właściwości (**Properties**), opcja **Application → Target Framework.**
- Dodaj w projekcie referencję do projektu kontraktu usługi WCF:
  - Zaznacz w Solution Explorer folder **References** i wybierz opcję **Add Reference.**



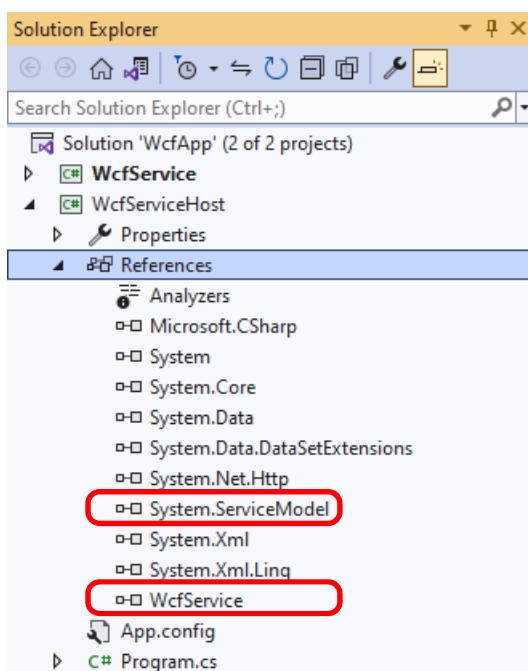
- W oknie menadżera referencji wybierz **Solution/Project**, zaznacz projekt kontraktu usługi WCF i zatwierdź:



- Dodaj w projekcie referencję do **System.ServiceModel**:
  - Kliknij w Solution Explorer prawym klawiszem folder **References** i wybierz opcję **Add Reference.**
  - W oknie menadżera referencji wybierz **Assemblies/Framework**, zaznacz **System.ServiceModel** i zatwierdź.



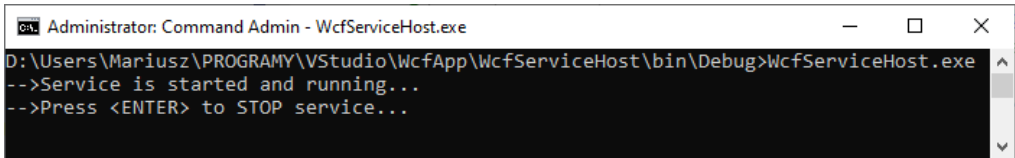
- Zweryfikuj pojawienie się dodatkowych referencji w projekcie jak na rysunku poniżej:



- Otwórz plik **Program.cs** i wpisz kod realizujący następujące funkcje:
  - Utworzenie URI z bazowym adresem serwisu.
  - Utworzenie instancji serwisu.
  - Dodanie punktu końcowego serwisu.
  - ustawienie metadanych (udostępnienia informacji o serwisie).
  - Uruchomienie serwisu (i na koniec zamknięcie serwisu).

Zamiast **xxx** podaj nr portu sieciowego (np. 10000 + nr stanowiska w laboratorium).

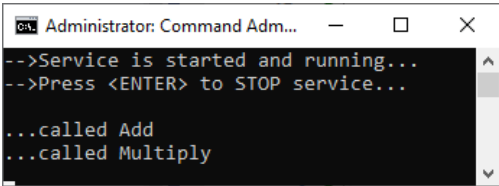
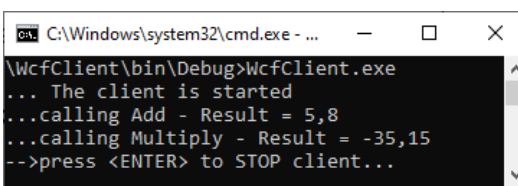
Zamiast **ServiceBaseName** (nazwa serwisu) wpisz własną nazwę swojej usługi.

	<pre> static void Main(string[] args) {     // Krok 1 URI dla bazowego adresu serwisu     Uri baseAddress = new Uri("http://localhost:xxx/ServiceBaseName");     // Krok 2 Instancja serwisu     ServiceHost myHost = new         ServiceHost(typeof(MyCalculator), baseAddress);     // Krok 3 Endpoint serwisu     BasicHttpBinding myBinding = new BasicHttpBinding();     ServiceEndpoint endpoint1 = myHost.AddServiceEndpoint (         typeof(ICalculator),         myBinding,         "endpoint1");      // Krok 4 Ustawienie metadanych     ServiceMetadataBehavior smb = new ServiceMetadataBehavior();     smb.HttpGetEnabled = true;     myHost.Description.Behaviors.Add(smb);      try {         // Krok 5 Uruchomienie serwisu.         myHost.Open();         Console.WriteLine("Service is started and running.");         Console.WriteLine("Press &lt;ENTER&gt; to STOP service...");         Console.WriteLine();         Console.ReadLine();    // aby nie kończyć natychmiast:         myHost.Close();     }     catch (CommunicationException ce) {         Console.WriteLine("Exception occurred: {0}", ce.Message);         myHost.Abort();     } } </pre> <ul style="list-style-type: none"> <li>• Usuń błędy poprzez dodanie importu odpowiednich bibliotek - po zaznaczeniu kursorem, opcja <b>Quick Actions and Refactorings...</b> w menu kontekstowym lub <b>Show potential fixes</b>.             <ul style="list-style-type: none"> <li>○ Najczęściej będzie to dodanie dyrektywy importu <b>using</b>.</li> </ul> </li> </ul>
4. Testowanie działania aplikacji	<p><b><i>UWAGA: aby uruchomić serwis poza platformą VS (np. z konsoli) trzeba mieć uprawnienia administratora w systemie. W innym przypadku trzeba system dodatkowo odpowiednio skonfigurować.</i></b></p> <ul style="list-style-type: none"> <li>• Przetestuj poprawność działania aplikacji             <ul style="list-style-type: none"> <li>○ Zbuduj kod wykonywalny aplikacji.</li> <li>○ Uruchom z linii komend aplikację hostującą serwis WCF</li> </ul> </li> </ul>  <ul style="list-style-type: none"> <li>• Sprawdź metadane serwisu i opis usługi             <ul style="list-style-type: none"> <li>○ Uruchom przeglądarkę i połącz się z adresem:  <b>http://localhost:xxx/ServiceBaseName</b></li> <li>○ Zapoznaj się z opisem serwisu.</li> </ul> </li> </ul>

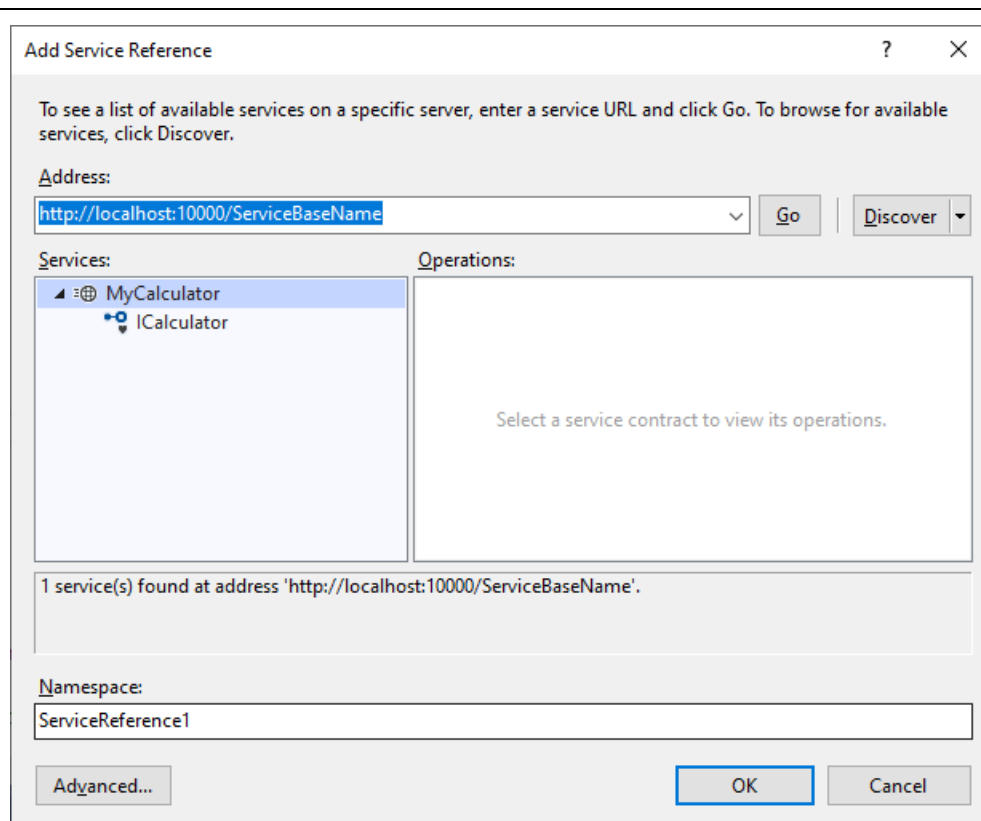
	<p><i>Połączenie z hostem i wyświetlenie strony z odpowiednim opisem oznacza poprawność działania aplikacji.</i></p> <ul style="list-style-type: none"> <li>o Przejdź do strony z opisem WSDL usługi. Zidentyfikuj ważne części opisu usługi: typy, wiadomości, operacje, punkt dostępu, itp.</li> <li>o Przejrzyj zawartość strony: <code>http://localhost:xxx/ ServiceBaseName?xsd=xsd0</code></li> </ul> <p>Uruchomienie usługi (nie hosta) z poziomu VS (Visual Studio) automatycznie uruchamia wbudowanego klienta umożliwiającego przetestowanie działania usługi.</p> <ul style="list-style-type: none"> <li>• Uruchom usługę z poziomu VS Zauważ, że w takim przypadku serwis jest dostępny (działa) na specjalnym porcie rezerwowanym przez VS (innym niż zdefiniowano w hoście). <ul style="list-style-type: none"> <li>o Kliknij w tym kliencie którąś operację (np. Add)</li> <li>o Wprowadź jakieś dane dla parametrów i wywołaj operację</li> <li>o Sprawdź postać wysyłanego i odbieranego komunikatu XML (wiadomości SOAP).</li> <li>o Zamknij działanie usługi uruchomionej z VS. Pozostaw działający serwis uruchomiony z konsoli.</li> </ul> </li> <li>• Uruchom program Postman (alternatywnie można SOAPUI) dla przetestowania działania serwisu. <ul style="list-style-type: none"> <li>o Utwórz żądanie HTTP konfigurując: <ul style="list-style-type: none"> <li>– metodę POST</li> <li>– adres taki jak dla endpointu usługi – sprawdź go w WSDL w sekcji <b>service-&gt;port-&gt;address</b>.</li> </ul> </li> </ul> </li> </ul> <p>Nagłówki HTTP:</p> <ul style="list-style-type: none"> <li>– <b>Content-type</b> = test/xml (taki jest w tym przypadku) <i>Uwaga:</i> dla bindingu WSHttpBinding typ jest inny: <i>application/soap+xml</i></li> <li>– <b>SOAPAction</b> – ustaw tu wartość atrybutu <b>soapAction</b> wywoływanej operacji zdefiniowanej w WSDL – tu zwykle o takiej postaci: <code>http://tempuri.org/service_interface_name/operation_name</code> (e.g. <code>http://tempuri.org/ICalculator/Add</code>) <i>Uwaga:</i> w WSDL nazwa operacji może zaczynać się małą literą zamiast dużą.</li> <li>o W ciele (body) żądania wpisz najprostszą postać wiadomości/żądania SOAP <ul style="list-style-type: none"> <li>– można skopiować ją z klienta testowego VS</li> <li>– pozostaw sekcję &lt;Header&gt; pustą (Postman miewa problemy z jej obsługą)</li> </ul> </li> </ul>
--	--

	<p>– żądanie powinno wyglądać podobnie do poniższego:</p> <pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"&gt;   &lt;s:Header/&gt;&lt;/s:Header&gt;   &lt;s:Body&gt;     &lt;Add xmlns="http://tempuri.org/"&gt;       &lt;val1&gt;-98.76&lt;/val1&gt;       &lt;val2&gt;12.34&lt;/val2&gt;     &lt;/Add&gt;   &lt;/s:Body&gt; &lt;/s:Envelope&gt;</pre> <ul style="list-style-type: none"> <li>o Uruchom/wyślij żądanie i sprawdź odpowiedź.</li> </ul> <p><b>Uwaga:</b> Ewentualne błędy, np. <i>"server was unable to process the request due to an internal error"</i> są zwykle powodowane błędami / literówkami w nazwach metod, parametrów, SOAPAction, używania małych zamiast wielkich liter (lub odwrotnie) itp.</p>
5. Implementacja klienta usługi – wersja I	<p>Utworzenie aplikacji klienta usługi i proxy klienta (<b>client proxy</b>) odrębnym kodem.</p> <ul style="list-style-type: none"> <li>• Utwórz odrębną solucję z projektem trzeci projekt aplikacji z szablonu C# <b>Console App (.NetFramework)</b> nadając mu własną nazwę (tu WcfClient).</li> <li>• Sprawdź (i ewentualnie ustaw) wersję Framework'a aplikacji (tak samo jak w drugim projekcie).</li> <li>• Dodaj w projekcie referencję do <b>System.ServiceModel</b> (tak samo jak w drugim projekcie (dla hosta)) – można też to zrobić na skutek podpowiedzi platformy przy wprowadzaniu kodu.</li> <li>• Dodaj do projektu interfejs (opcja <i>Add/New Item.../Interface</i>) Zdefiniuj dokładnie tak samo interfejs kontraktu usługi (<b>ICalculator</b>).</li> <li>• W pliku klienta Program.cs wprowadź kod, który:       <ul style="list-style-type: none"> <li>o Tworzy instancję klienta (client proxy)           <ul style="list-style-type: none"> <li>– utworzenie obiektu Uri adresu bazowego usługi.</li> <li>– utworzenie <i>binding'u</i></li> <li>– utworzenie punktu końcowego (<i>endpointa</i>)</li> <li>– utworzenie klienta proxy z użyciem fabryki kanału (<i>Channel factory</i>)</li> </ul> </li> <li>o Wywołuje operację serwisu z użyciem klienta proxy</li> <li>o Zamyka klienta</li> </ul> <pre>static void Main(string[] args) {     Console.WriteLine("... The client is started");     // Step 1: Create client proxy based on communication channel.     // base address:     Uri baseAddress;      // binding, address, endpoint address:     BasicHttpBinding myBinding = new BasicHttpBinding();     baseAddress = new         Uri("http://localhost:10000/ServiceBaseName/endpoint1");</pre> </li> </ul>



	<pre> EndpointAddress eAddress = new EndpointAddress(baseAddress); // channel factory: ChannelFactory&lt;ICalculator&gt; myCF = new     ChannelFactory&lt;ICalculator&gt;(myBinding, eAddress);  // client proxy (here myClient) based on channel ICalculator myClient = myCF.CreateChannel();  // Step 2: service operations call. Console.WriteLine("...calling Add (for endpoint1) "); double result = myClient.add(-3.7, 9.5); //just example values Console.WriteLine("Result = "+result);  [...] // here possible other operations Console.WriteLine("...press &lt;ENTER&gt; to STOP client..."); Console.WriteLine(); Console.ReadLine(); // to not finish app immediately:  // Step 3: Closing the client - closes connection and clears resources. ((IClientChannel)myClient).Close(); Console.WriteLine("...Client closed - FINISHED"); } </pre>
6. Testowanie działania aplikacji	<ul style="list-style-type: none"> <li>• Uruchom serwis (aplikację hostującą usługę WCF) w jednym oknie konsoli.</li> <li>• Uruchom klienta w drugim oknie konsoli.</li> <li>• Skontroluj wyniki działania.</li> <li>• Efekt działania klienta i serwisu powinien być podobny do zamieszczonych ilustracji.</li> </ul> <div style="display: flex; justify-content: space-around;">   </div> <ul style="list-style-type: none"> <li>• Zakończ działanie wszystkich aplikacji.</li> </ul>
7. Modyfikacja hosta serwisu	<ul style="list-style-type: none"> <li>• Otwórz plik <b>Program.cs</b> hosta i dopisz kod realizujący następujące funkcje:             <ul style="list-style-type: none"> <li>◦ Dodanie kolejnego endpointa (dla transportu WSHttp).</li> <li>◦ Wyświetlenie informacji o kontrakcie.</li> </ul> </li> <li>• <u>Przed uruchomieniem serwisu</u> (przed funkcją <code>Open()</code>) utwórz obiekt wiązania <code>WSHttpBinding</code> (dla transportu WS Http) i dodaj dodatkowy punkt końcowy/endpoint:             <pre> WSHttpBinding binding2 = new WSHttpBinding(); binding2.Security.Mode = SecurityMode.None; ServiceEndpoint endpoint2 = myHost.AddServiceEndpoint(     typeof(ICalculator),     binding2, "endpoint2"); </pre> </li> <li>• Następnie dopisz kod wyświetlający informacje o endpointach (jak poniżej dla endpointa 1), powielając go dla endpointa2:</li> </ul>

	<pre>Console.WriteLine("\n--&gt; Endpoints:"); Console.WriteLine("\nService endpoint {0}:", endpoint1.Name); Console.WriteLine("Binding: {0}", endpoint1.Binding.ToString()); Console.WriteLine("ListenUri: {0}", endpoint1.ListenUri.ToString());</pre> <p><b>Dodatkowa uwaga:</b>  <i>wiele elementów serwisu, w tym dodatkowe endpointy, można również definiować w pliku konfiguracyjnym projektu hosta <b>App.config</b>.</i></p>
8. Testowanie działania serwisu	<ul style="list-style-type: none"> <li>Przebuduj (opcja Rebuild) kontrakt usługi i host usługi.</li> <li>Uruchom z konsoli serwis i sprawdź działanie.             <ul style="list-style-type: none"> <li>Zapoznaj się z wyświetlanymi danymi w konsoli hosta.</li> </ul> </li> <li>Przetestuj działanie za pomocą aplikacji Postman             <ul style="list-style-type: none"> <li>zmień adres żądania na endpoint2</li> <li>zmień nagłówek Content-Type na <b>application/soap+xml</b></li> <li>zamieść w kopercie odwołanie do standardów SOAP oraz w nagłówku atrybuty:                 <ul style="list-style-type: none"> <li>- <b>Action</b> – takie jak w WSDL atrybut <b>soapAction</b></li> <li>- <b>To</b> – taki jak adres endpointa:</li> </ul> </li> </ul> </li> </ul> <pre>&lt;s:Envelope xmlns:a="http://www.w3.org/2005/08/addressing"   xmlns:s="http://www.w3.org/2003/05/soap-envelope"&gt;   &lt;s:Header&gt;     &lt;a:Action s:mustUnderstand="1"&gt;       http://tempuri.org/ICalculator/Add     &lt;/a:Action&gt;     &lt;a:To&gt;http://localhost:10000/MyService/endpoint2&lt;/a:To&gt;   &lt;/s:Header&gt;   &lt;s:Body&gt; [...] &lt;/s:Body&gt; &lt;/s:Envelope&gt;</pre> <ul style="list-style-type: none"> <li>Wyślij żądanie i sprawdź odpowiedź.</li> </ul>
9. Implementacja klienta usługi – wersja II – konfigurowanie proxy klienta	<p>Utworzenie proxy klienta (<b>client proxy</b>) z użyciem funkcji Visual Studio: <b>Add Service Reference</b>.</p> <ul style="list-style-type: none"> <li>Dodaj w projekcie klienta referencję serwisową do zdefiniowanej usługi (Ilustracja dalej na rysunku):             <ul style="list-style-type: none"> <li><b>Uruchom najpierw aplikację hostującą usługę WCF!</b></li> <li>Zaznacz w Solution Explorer prawym klawiszem folder <b>References</b> i wybierz opcję <b>Add Service Reference</b>.</li> <li>W oknie Add Service Reference, w polu <b>Address</b> wpisz adres usługi (endpoint):  <b>http://localhost:xxx/ServiceBaseName</b>                Zamiast <b>xxx</b> podaj odpowiedni nr portu.</li> <li>Naciśnij przycisk <b>Go</b> i powinna pojawić się dostępna usługa na podanym punkcie dostępu (patrz rys. dalej). Wybranie kontraktu (interfejsu) dodatkowo pokaże dostępne operacje (metody).</li> <li>Zatwierdź wybór przyciskiem <b>OK</b>.</li> </ul> </li> </ul>

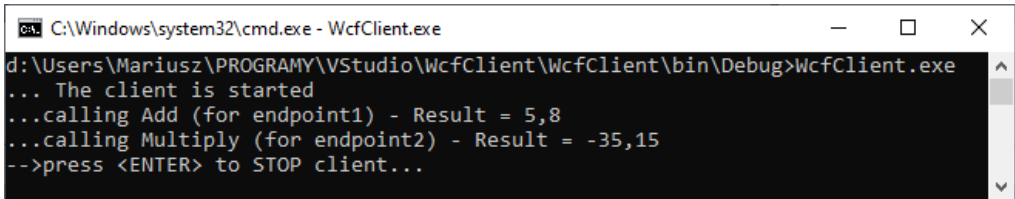


***W powyższy sposób generowany jest kod proxy klienta (client proxy) realizującego wywołania usług – kod dodatkowego modułu aplikacji.***

Konfiguracja klienta zawarta jest, w utworzonym poprzez dodanie referencji serwisowej, pliku konfiguracyjnym **App.config** projektu.

- Otwórz plik **App.config** klienta i przeanalizuj jego zawartość.
- Zwróć uwagę zwłaszcza na sekcję i nazwę i typ wiązania (binding), sekcję i nazwę punktu dostępu (endpoint) oraz specyfikację kontraktu (contract).
- Jego zawartość powinna być podobna do tej na ilustracji.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup> ...</startup>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_ICalculator" />
      </basicHttpBinding>
      <wsHttpBinding>
        <binding name="WSHttpBinding_ICalculator">
          <security mode="None" />
        </binding>
      </wsHttpBinding>
    </bindings>
```

	<pre> &lt;client&gt;   &lt;endpoint     address="http://localhost:10000/MyService/endpoint1"     binding="basicHttpBinding"     bindingConfiguration="BasicHttpBinding_ICalculator"     contract="ServiceReference1.ICalculator"     name="BasicHttpBinding_ICalculator" /&gt;    &lt;endpoint     address="http://localhost:10000/MyService/endpoint2"     binding="wsHttpBinding"     bindingConfiguration="WSHttpBinding_ICalculator"     contract="ServiceReference1.ICalculator"     name="WSHttpBinding_ICalculator" /&gt;  &lt;/client&gt; &lt;/system.serviceModel&gt; &lt;/configuration&gt; </pre>
10. Implementacja klienta usługi – wersja II – utworzenie proxy i wywołanie	<ul style="list-style-type: none"> <li>W pliku klienta <b>Program.cs</b> dopisz kod: <ul style="list-style-type: none"> <li>Utworzenie instancji proxy klienta (client proxy).</li> <li>Wywołanie operacji usługi z klienta.</li> </ul> <pre> CalculatorClient myClient2 = new     CalculatorClient("WSHttpBinding_ICalculator"); Console.WriteLine("...calling Multiply (for endpoint2) - "); result = myClient2.Multiply(-3.7, 9.5); //just example values Console.WriteLine("Result = " + result); </pre> <p>Obiekt klienta proxy (tu myClient2) tworzy się wg. wskazówki ze strony z opisem usługi:</p> <ul style="list-style-type: none"> <li>– nazwa klasy jest taka jak nazwa klasy usługi plus człon <b>"Client"</b>,</li> <li>– jednak jeśli usługa ma więcej niż jeden punkt dostępowy (endpoint) to konstruktor musi specyfikować jeden z nich,</li> <li>– do specyfikacji używana jest nazwa endpointa z pliku <b>App.config</b> (tu: <b>"WSHttpBinding_ICalculator"</b>).</li> </ul> <li>Usuń błędy poprzez dodanie importu odpowiednich klas w tym <b>ServiceReference1</b>.</li> </li></ul>
11. Testowanie działania aplikacji	<ul style="list-style-type: none"> <li>Uruchom serwis (aplikację hostującą usługę WCF) w jednym oknie konsoli, klienta w drugim oknie konsoli i skontroluj wyniki działania.</li> <li>Efekt działania powinien być podobny do poniższego.</li> </ul> 
12. Operacje asynchroniczne – wersja I	<p>Asynchroniczne wykonanie operacji można zrealizować na kilka sposobów. Nowa wersja WCF automatycznie generuje metody asynchroniczne zwracające <b>Task&lt;T&gt;</b> wg modelu <b>ATM</b> (<i>Asynchronous Task Model</i>). Mają one nazwy z dodanym członem <b>Async</b>. Dlatego zazwyczaj rekomendowane jest użycie tego podejścia.</p>

**Serwis:**

- Dopisz do kontraktu (interfejs i implementacja) kolejną operację **HMultiply** – taką jak Multiply ale z dodanym uśpieniem na 5 sek. (coś jakby symulacja długotrwałych obliczeń (*Heavy Multiply*)).
- Dla kontraktu usługi zdefiniuj zachowanie aby wykonywać działania instancji usługi wielowątkowo **ConcurrencyMode=Multiple**.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
ConcurrencyMode = ConcurrencyMode.Multiple)]
```

```
public class MyCalculator : ICalculator { [...] }
```

[...] – oznacza już istniejące fragmenty kodu.

- Zbuduj (*Rebuild*) serwis od nowa i uruchom go.

**Klient:**

W kliencie zdefiniujemy oddzielną metodę, w której będziemy oczekiwali na rezultat wywołanej wcześniej operacji asynchronicznej serwisu (która zwraca obietnicę wykonania Task<T>)

- Zaktualizuj referencje serwisową (aby klient widział zmiany w serwisie).
- Zdefiniuj w kliencie metodę, która wywoła z oczekiwaniem asynchroniczną wersję metody HMultiply – **HmultiplyAsync**:

```
static async Task<double> callHMultiplyAsy(double n1, double n2) {
    Console.WriteLine("2.....called callHMultiplyAsync");
    double reply = await myClient2.HMultiplyAsync(n1, n2);
    Console.WriteLine("2.....finished HMultiplyAsync");
    return reply;
}
```

- W metodzie Main dopisz:
  - za wywołaniem metody **Multiply**, wywołanie metody **callHMultiplyAsync**:
 

```
Console.WriteLine("2...calling HMultiply ASYNCHRONOUSLY !!!");
Task<double> asyResult = callHMultiplyAsync(1.1, -3.3);
```
  - za tym wywołaniem dopisz wstrzymanie działania (Thread.Sleep) na ok 100ms (aby lepiej wylapać kolejność działań), a następnie kolejne wywołanie dodawania (żądanie synchroniczne).
  - za tym wywołaniem, pod koniec, przed zamknięciem aplikacji (klientów proxy) dopisz pobranie rezultatu z metody asynchronicznej i wypisz wynik:
 

```
result = asyResult.Result;
Console.WriteLine("2...HMultiplyAsync Result = " + result);
```

**Uwaga:** tu jeśli wyniku jeszcze nie będzie, nastąpi wstrzymanie działania klienta.

- Uruchom aplikację i sprawdź działanie.  
Zwróć uwagę na kolejność wywołań operacji i wypisania otrzymanych wyników.

<p>13. Operacje asynchroniczne – wersja I – wstępne przygotowanie</p>	<p>Pierwszym podejściem, zgodnym ze wzorcem SOA wykonania operacji asynchronicznie za pomocą żądań jednokierunkowych – bez odpowiedzi (One-way) – jest definiowanie kontraktu zwrotnego (Callback Contract) – jedno żądanie one-way wywołuje operację, drugie żądanie (z serwisu do klienta) zwraca wynik.</p> <p><b>Serwis:</b></p> <p>Usługa będzie realizowana w oddzielnym projekcie.</p> <ul style="list-style-type: none"> <li>• Dodaj do solucji projekt WCF Library trzeciej usługi (np. o nazwie CallbackService) – projekt drugiego kontraktu.</li> <li>• Dodaj w hoście referencję do tego projektu.</li> </ul>
<p>14. Definiowanie kontraktu usługi z operacjami typu callback</p>	<p>Zdefiniuj w projekcie nowy kontrakt usługi typu <b>Callback</b> mającej jedną operację (metodę). W tym celu definiuje się:</p> <ul style="list-style-type: none"> <li>- operacje typu <b>OneWay</b>,</li> <li>- zachowanie usługi typu <b>CallbackContract</b> specyfikujące typ interfejsu zwrotnego (tu: specyfikujemy go jako <b>ISuperCalcCallback</b>) – jest to interfejs klienta dla obsługi wywołań zwrotnych – ten interfejs musi być implementowany w kliencie,</li> <li>- zachowanie można zdefiniować jako atrybut kontraktu serwisu (w <b>[ServiceContract]</b>),</li> <li>- dodatkowo określimy wymaganie działania instancji serwisu w ramach sesji.</li> </ul> <ul style="list-style-type: none"> <li>• Zdefiniuj kod interfejsu kontraktu usługi <b>ISuperCalc</b> zawierający metodę/operację asynchroniczną (z callbackiem) <b>Factorial</b> (obliczanie silni), definiując dodatkowo atrybut serwisu <b>CallbackContract</b> i wymaganie trybu działania w sesji: <pre> [ServiceContract (SessionMode = SessionMode.Required,                   CallbackContract=typeof(ISuperCalcCallback))] public interface ISuperCalc {     [OperationContract(IsOneWay = true)]     void Factorial(double n);     [OperationContract(IsOneWay = true)]     void DoSomething(int sec); } </pre> </li> <li>• Zdefiniuj w tym pliku również interfejs <b>ISuperCalcCallback</b> zawierający opis metod wywoływanych u klienta w celu przekazania rezultatów wykonania operacji <b>Factorial</b> – tu zawierający metodę <b>FactorialResult</b> dla wyniku obliczeń silni. <p>Dopisz w tym samym pliku drugi interfejs:</p> <pre> public interface ISuperCalcCallback {     [OperationContract(IsOneWay = true)]     void FactorialResult(double result); } </pre> </li> </ul>

15. Implementacja kontraktu usługi	<p>Zaimplementuj kontrakt – klasę implementującą każdą z wymaganych metod interfejsu <b>ISuperCalc</b>.</p> <ul style="list-style-type: none"> <li>W pliku <b>Service1.cs</b> wpisz kod klasy <b>MySuperCalc</b> implementującej interfejs <b>ISuperCalc</b>.             <ul style="list-style-type: none"> <li>Dla usługi definiuje się także zachowanie <b>InstanceContextMode=PerSession</b> oznaczające tworzenie instancji obiektu (instancji usługi) dla każdej sesji.</li> <li>W konstruktorze pobierany jest uchwytu (handler) dla callback'u.                 <pre>[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession,                     ConcurrencyMode = ConcurrencyMode.Multiple)]  public class MySuperCalc : ISuperCalc {     double result;     ISuperCalcCallback callback = null;     public MySuperCalc() {         callback = OperationContext.Current.GetCallbackChannel             &lt;ISuperCalcCallback&gt;();     }     public void Factorial(double n) {         Console.WriteLine("...called Factorial({0})", n);         Thread.Sleep(1000);         result = 1;         for (int i = 1; i &lt;= n; i++ )             result *= i;         callback.FactorialResult(result);     } }</pre> </li> </ul> </li> <li>Na końcu wywołujemy metodę callback'ową w kliencie.</li> </ul>
16. Rozbudowa hosta dla trzeciej usługi	<p>Dopisz w kodzie aplikacji hostującej uruchomienie drugiego serwisu.</p> <ul style="list-style-type: none"> <li>W pliku Program.cs dopisz w odpowiednich miejscach kod realizujący następujące funkcje:             <ul style="list-style-type: none"> <li>Utworzenie URI z bazowym adresem drugiego serwisu.</li> <li>Utworzenie obiektu hosta drugiego serwisu.</li> <li>Dodanie punktu końcowego z wiązaniem <b>WSDualHttpBinding</b>.</li> <li>Zdefiniowanie metadanych serwisu.</li> <li>Uruchomienie drugiego serwisu.</li> </ul> </li> </ul> <p><i>[...] oznacza już istniejące fragmenty kodu.</i></p> <pre>static void Main(string[] args) {     [...]     Uri baseAddress3 = new Uri(...);     ServiceHost myHost3 = new         ServiceHost(typeof(MySuperCalc), baseAddress3);     WSDualHttpBinding myBinding3 = new WSDualHttpBinding();     ServiceEndpoint endpoint3 =         myHost3.AddServiceEndpoint(typeof(ISuperCalc),                                     myBinding3, "endpoint3");     myHost3.Description.Behaviors.Add(smb);     try {         [...]     } }</pre>



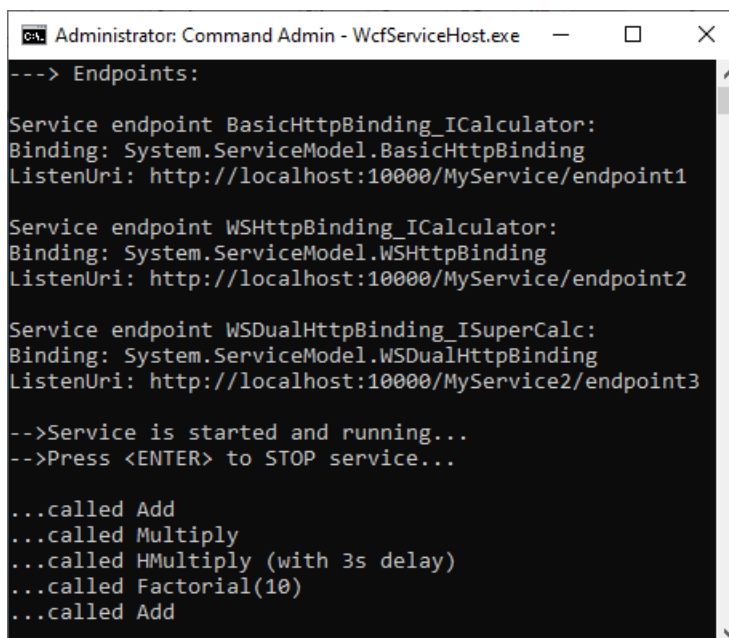
	<pre> myHost3.Open(); Console.WriteLine("--&gt; Service SuperCalc is running."); [...] myHost3.Close(); } catch (CommunicationException ce) { [...] myHost3.Abort(); } } </pre> <p>Uruchom z konsoli serwis (host) i sprawdź działanie.</p>
<p>17.Rozbudowa klienta dla wykorzystania drugiej usługi</p>	<ul style="list-style-type: none"> <li>• Dodaj w kliencie referencję serwisową do drugiej usługi: <b>Uwaga: pamiętaj o uruchomieniu najpierw aplikacji hostującej usługę!</b></li> <li>• Dodaj do projektu klienta nową klasę (tu o nazwie <b>SuperCalcCallback</b>), w której zdefiniowane będą operacje wywoływane zwrótnie przez serwis, aby odesłać wyniki działania jego operacji usług. <pre> class SuperCalcCallback : ISuperCalcCallback {     public void FactorialResult(double result) {         //here the result is consumed         Console.WriteLine(" Factorial = {0}", result);     } } </pre> </li> <li>• Otwórz plik <b>Program.cs</b> i dopisz kod w funkcji <b>Main</b> realizujący następujące działania: <ul style="list-style-type: none"> <li>○ Utworzenie obiektu uchwytu (handlera) z operacjami odbioru wyników od serwisu.</li> <li>○ Utworzenie instancji klienta proxy (<i>proxy client</i>).</li> <li>○ Wywołanie operacji usługi z klienta (proxy).</li> <li>○ Zamknięcia klienta</li> </ul> <p>Dopisz ten kod za wywołaniem metody <b>callHMultiplyAsync</b>. Odbiór i wypisywanie wyników będzie asynchroniczny – inicjowany przez serwis.</p> <pre> static void Main(string[] args) {     [...]     SuperCalcCallback myCbHandler = new SuperCalcCallback ();     InstanceContext instanceContext =         new     InstanceContext(myCbHandler);     SuperCalcClient myClient3 = new     SuperCalcClient(instanceContext);     double value1 = 10;     Console.WriteLine("...calling Factorial({0})...", value1);     myClient3.Factorial(value1);     [...]     client3.Close();     Console.WriteLine("CLIENT3 - STOP"); } </pre> </li> </ul>



### 18. Testowanie działania aplikacji

- Uruchom serwis (aplikację hostującą usługi) w jednym oknie konsoli.
- Uruchom klienta w drugim oknie konsoli.
- Skontroluj wyniki działania. Zwróć uwagę na momenty czasowe działania serwisu i klienta.
- Ostateczny wynik działania powinien być podobny do poniższego:

Okno hosta:



```
Administrator: Command Admin - WcfServiceHost.exe
---> Endpoints:

Service endpoint BasicHttpBinding_ICalculator:
Binding: System.ServiceModel.BasicHttpBinding
ListenUri: http://localhost:10000/MyService/endpoint1

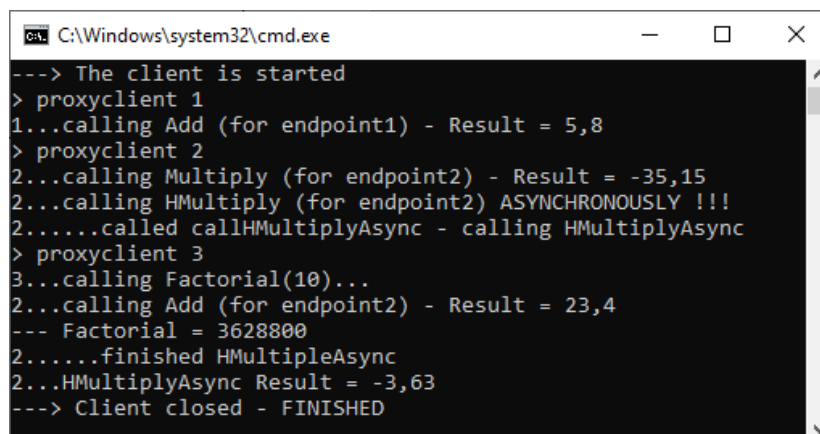
Service endpoint WSHttpBinding_ICalculator:
Binding: System.ServiceModel.WSHttpBinding
ListenUri: http://localhost:10000/MyService/endpoint2

Service endpoint WSDualHttpBinding_ISuperCalc:
Binding: System.ServiceModel.WSDualHttpBinding
ListenUri: http://localhost:10000/MyService2/endpoint3

-->Service is started and running...
-->Press <ENTER> to STOP service...

...called Add
...called Multiply
...called HMultiply (with 3s delay)
...called Factorial(10)
...called Add
```

Okno klienta:



```
C:\Windows\system32\cmd.exe
---> The client is started
> proxyclient 1
1...calling Add (for endpoint1) - Result = 5,8
> proxyclient 2
2...calling Multiply (for endpoint2) - Result = -35,15
2...calling HMultiply (for endpoint2) ASYNCHRONOUSLY !!!
2.....called callHMultiplyAsync - calling HMultiplyAsync
> proxyclient 3
3...calling Factorial(10)...
2...calling Add (for endpoint2) - Result = 23,4
--- Factorial = 3628800
2.....finished HMultipleAsync
2...HMultiplyAsync Result = -3,63
---> Client closed - FINISHED
```

## 3 Zadanie – część II

- Przećwiczyć wg instrukcji technikę tworzenia serwisów i klientów WCF.
  - Definiowanie, konfiguracja i implementacja kontraktów.
  - Tworzenie hosta usługi. Definiowanie punktów końcowych (endpoint).
  - Tworzenie klienta, wiązanie i wywoływanie operacji usług.
  - Operacje asynchroniczne wg modelu ATM i CallbackContract.
- Przygotować się do napisania na zajęciach aplikacji o podobnych funkcjonalnościach lub modyfikacji apl. według wskazówek prowadzącego.