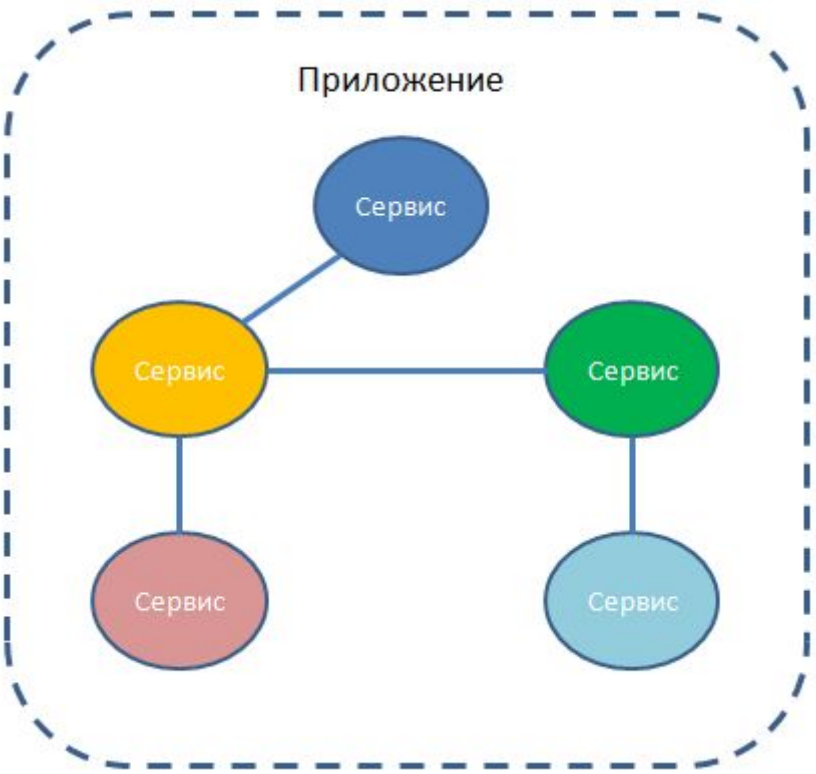
Урок 8

Обзор сервис-ориентированной архитектуры приложений (SOA)

Web-сервис asmx. WCF-сервис. Web-api сервис.

Обзор сервис-ориентированной архитектуры приложений (SOA)

Сервис – это функциональная единица, доступная внешнему миру с помощью стандартного механизма обмена сообщениями. Сервис-ориентированное приложение представляет собой агрегацию сервисов в логически целостное приложение, также как объектно-ориентированное приложение является агрегацией отдельных объектов.



Приложение также может выступать как еще один сервис, аналогично тому, как крупный объект может быть композицией более мелких объектов.

Внутри сервиса разработчики могут по-прежнему использовать специфические языки программирования, технологии и фреймворки. Однако между сервисами сохраняются стандартные протоколы и сообщения, контракты и обмен метаданными.

Сервисы, входящие в состав отдельного приложения, могут располагаться в одном и том же месте или быть распределены по разным узлам локальной сети или Интернета. Отдельные сервисы могут разрабатываться разными компаниями, размещаться на разных технологических платформах и находиться в разных часовых поясах. Все эти аспекты скрыты от клиентских приложений, которые могут взаимодействовать с сервисами путем отправки и получения сообщений. Механизм обмена сообщениями между клиентскими приложениями и сервисами сводит на нет разницу между ними, преобразуя входящие и исходящие сообщения к стандартным протоколам сетевого обмена.

Принципы сервис-ориентированной архитектуры

- **Сервисы должны иметь четкие границы**. Любой сервис ограничен возможностями технологии, с помощью которой он реализован, и своим расположением. Эти ограничения не должны проявляться в контрактах сервисов или типах данных, используемых для обмена.

- **Сервисы автономны**. Сервис для работы не должен нуждаться в своих клиентах или других сервисах. Он должен функционировать и обновляться независимо от клиентских приложений. Также сервис должен иметь свой механизм авторизации, не зависящий от уровня доступа пользователя клиентского приложения.

- **Сервис предоставляет только контракты взаимодействия с ним и схему данных**. Детали реализации и работы веб-сервиса должны быть недоступны извне.

- **Совместимость сервисов основана на правилах (policy)**. Сервисы должны публиковать правила, описывающие, что они могут делать и как клиентские приложения взаимодействуют с ними.

Наиболее распространенные технологии реализации веб-сервисов

- **SOAP** (Simple Object Access Protocol). По сути, это три стандарта: SOAP/WSDL/UDDI;

- **REST** (Representational State Transfer);

Стандарты, используемые для построения веб-сервиса:

- **HTTP** (Hypertext Transfer Protocol). HTTP – это протокол, реализующий принцип запрос/ответ. Обычно реализуется поверх TCP/IP-протокола. HTTP-клиент открывает TCP-подключение к серверу и отправляет HTTP-запрос. Запрос содержит метод **GET** или **POST**. **GET** используется для получения с сервера файла – например, HTML-страницы. Для передачи данных предназначен метод **POST**. Ответ на HTTP-запрос содержит возвращаемые данные;

- **XML** и **XML-схема**. **XML** – ключевая технология веб-сервисов. **SOAP** использует **XML** как формат представления данных. **WSDL** использует XML-схему для описания структуры веб-сервисов. **UDDI** использует XML-схему для описания структуры репозитория веб-сервисов и для взаимодействия с репозиторием;

- **SOAP**. Протокол для обмена информацией в децентрализованной, распределенной среде.

- **WSDL** (Web Service Description Language). Используется для описания интерфейса веб-сервиса.

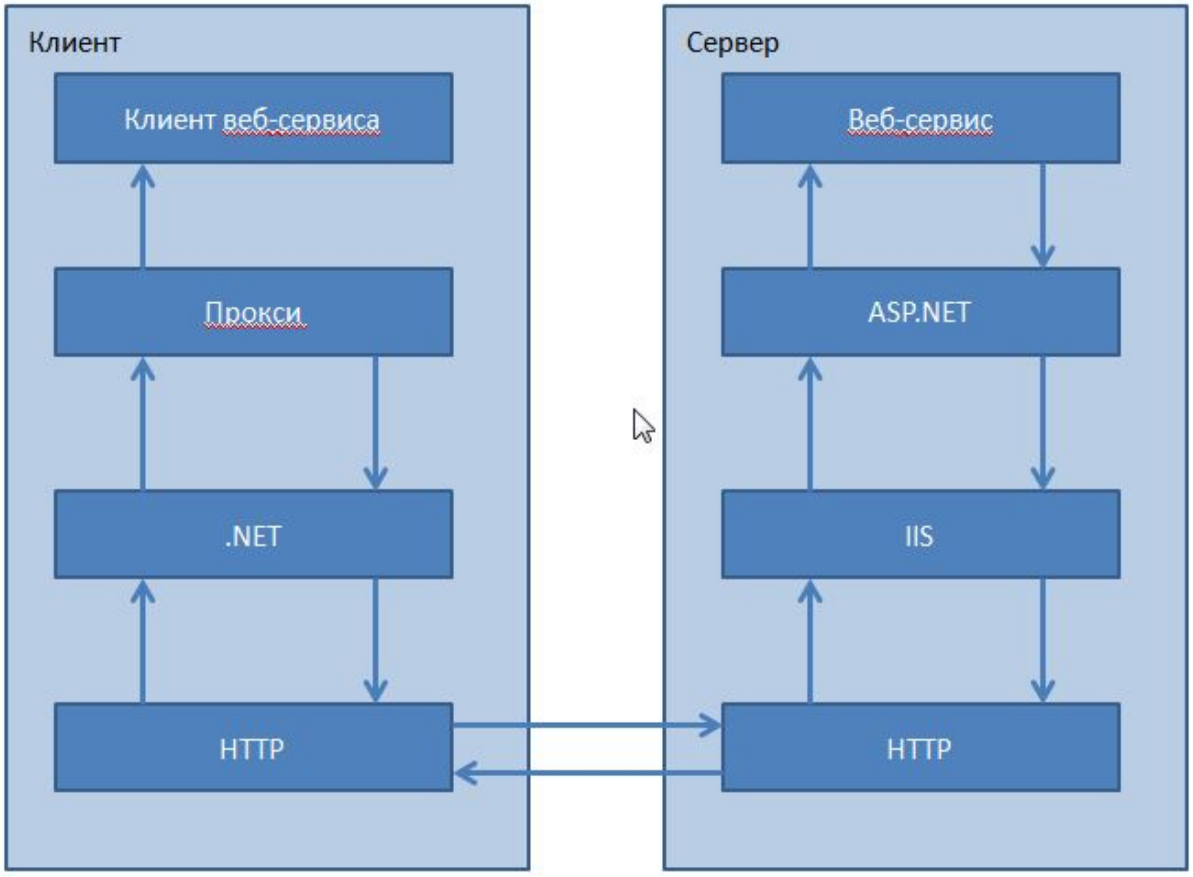
- **UDDI** (Universal Description, Discovery and Integration). Глобальный репозиторий веб-сервисов и технология взаимодействия с репозиторием.

Web-сервис

Характерные особенности:

- Основой веб-сервисов является протокол **SOAP**;
- Формат сериализации данных для веб-сервисов основывается на спецификации **XML**. Данные между клиентским приложением и сервисом передаются в XML-формате;
- В качестве транспортного протокола используется только **HTTP**;
- Не является open source технологией, но с сервисом могут работать клиентские приложения, понимающие **xml**;
- Может быть развернут только на веб-сервере Microsoft, IIS (Internet Information Services);
- Для описания веб-сервисов используется **WSDL** (Web Service Description Language).

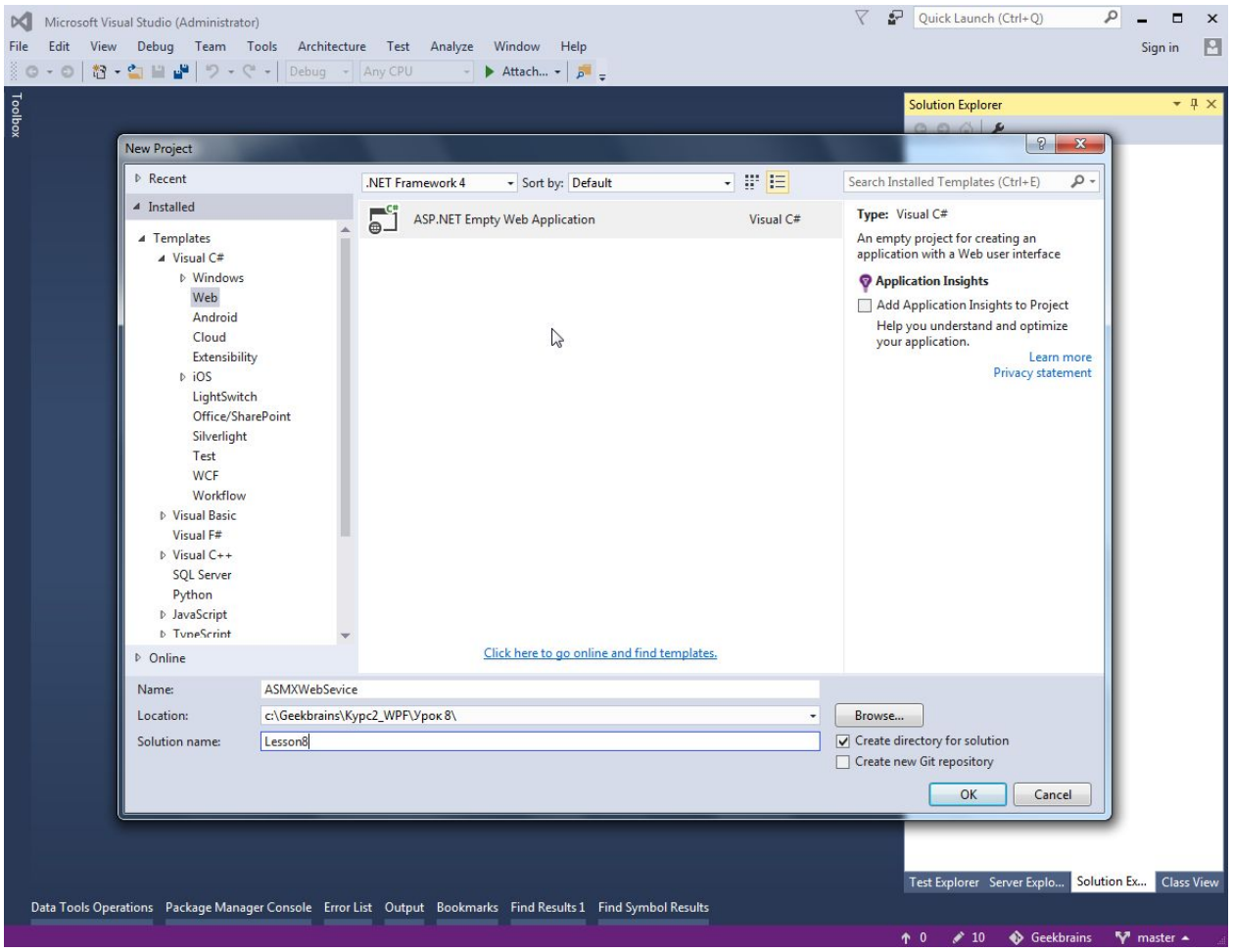
Схема использования веб-сервиса клиентским приложением:



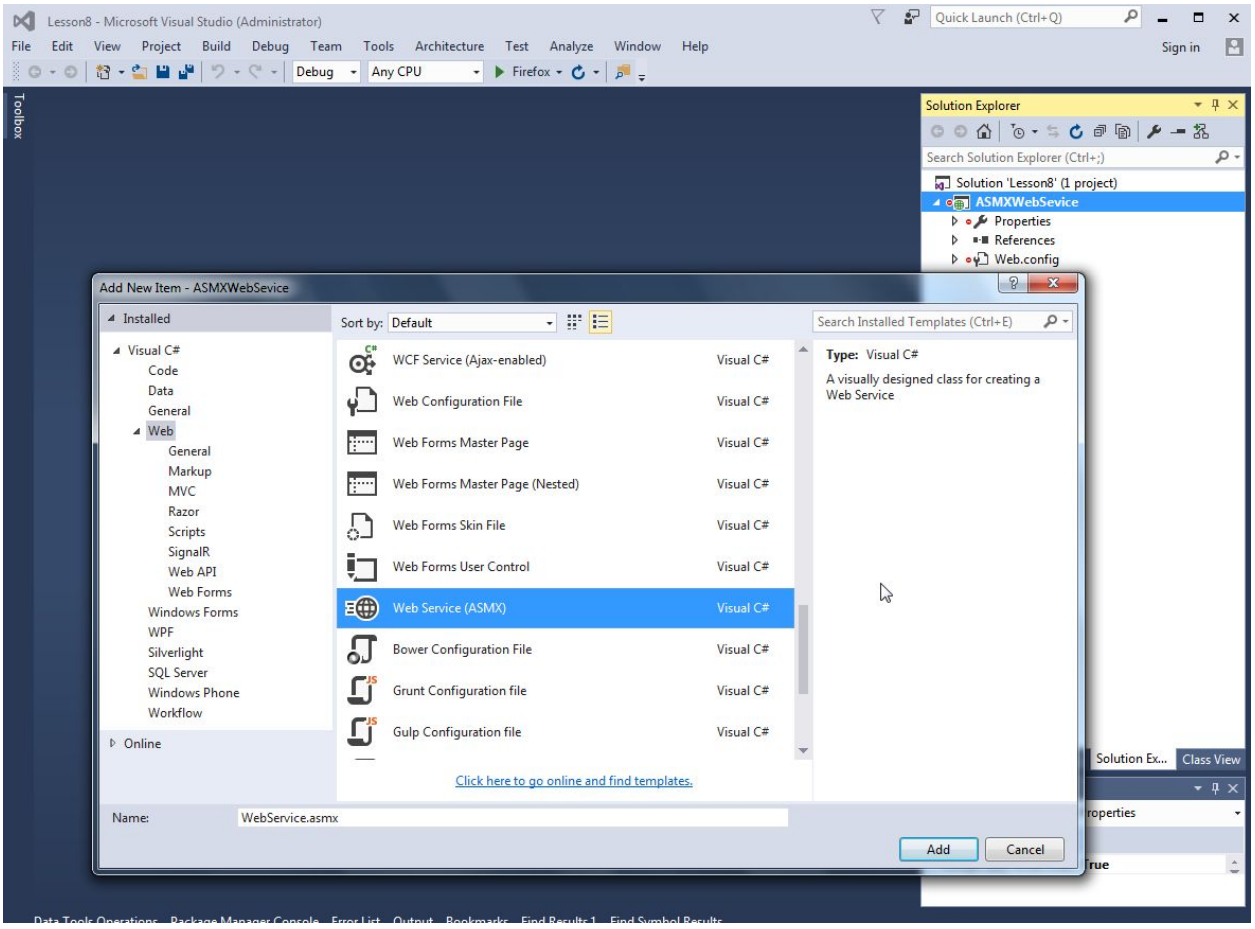
Создание веб-сервиса

Клиентское приложение **WPF** выполняет сложение двух чисел, используя веб-сервис. Клиентское приложение и веб-сервис развернуты на одном компьютере.

Создадим веб-сервис, выполняющий сложение двух чисел. Для этого добавим пустой (Empty) **ASP.Net-проект**:



После этого добавим в него проект (**Add New Item**) **Web-service ASMX**:



Visual Studio сформирует следующий код веб-сервиса:

WebService.asmx.cs

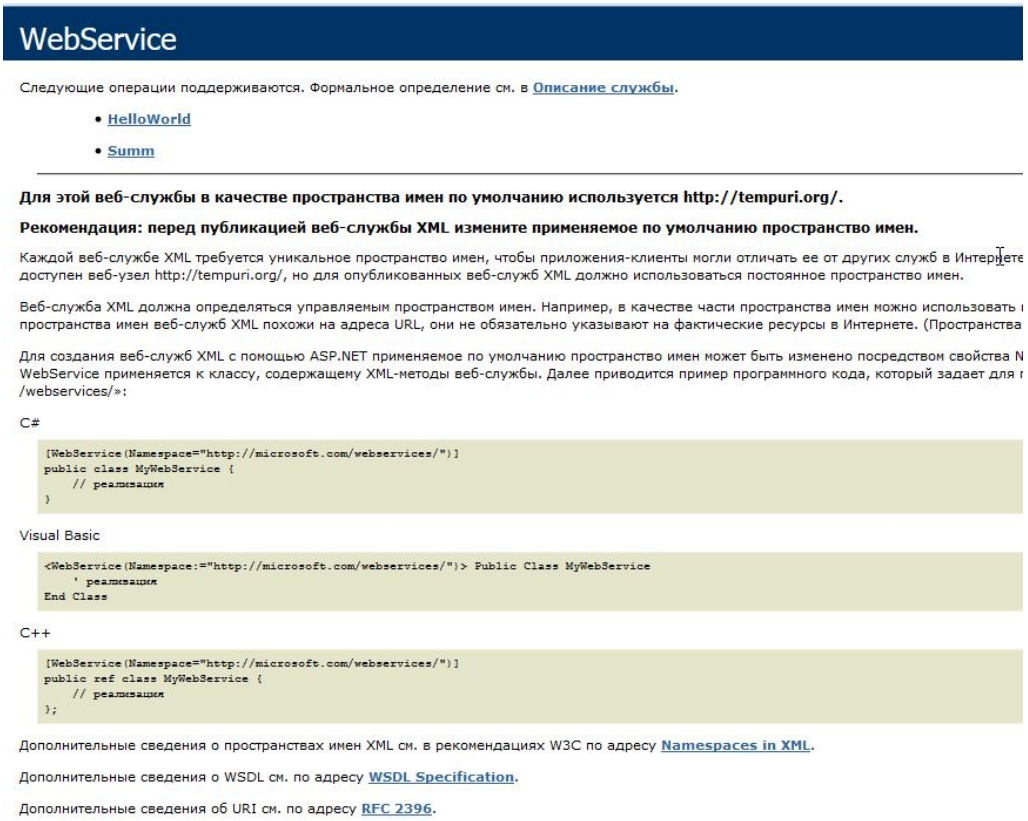
```
using System.Web.Services;
namespace ASMXWebSevice
{
    /// <summary>
    /// Summary description for WebService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX,
uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class WebService : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

В коде веб-сервиса уже реализован метод «**HelloWorld**». Дополним код веб-сервиса методом для расчета суммы чисел.

WebService.asmx.cs

```
using System.Web.Services;
namespace ASMXWebSevice
{
    /// <summary>
    /// Summary description for WebService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
     // To allow this Web Service to be called from script, using ASP.NET AJAX,
uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class WebService : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
        [WebMethod]
        public string Summ(string number1, string number2)
        {
                // Входные параметры объявляем типа String, чтобы принимать от
пользователя любые символы,
                // анализировать их и при «плохом вводе» сообщать по-русски
            float a, b;
                                        bool   flag   =   float.TryParse(number1,
System.Globalization.NumberStyles.Number,
            System.Globalization.NumberFormatInfo.CurrentInfo, out a);
            if (flag == false) return "В первом поле должно быть число";
                                              flag   =   float.TryParse(number2,
System.Globalization.NumberStyles.Number,
            System.Globalization.NumberFormatInfo.CurrentInfo, out b);
            if (flag == false) return "В первом поле должно быть число";
            return "Сумма:" + (a + b).ToString();
        }
    }
}
```

Выберем в обозревателе решений файл с расширением **.asmx** и выполним для него команду «**Отобразить**», **View In Browser**. Должен будет запуститься браузер и, если все сделано правильно, появится следующее окно:



WebService

Следующие операции поддерживаются. Формальное определение см. в Описание службы.

- HelloWorld
- Summ

Для этой веб-службы в качестве пространства имен по умолчанию используется http://tempuri.org/.

Рекомендация: перед публикацией веб-службы XML измените применяемое по умолчанию пространство имен.

Каждой веб-службе XML требуется уникальное пространство имен, чтобы приложения-клиенты могли отличать ее от других служб в Интернете доступен веб-узел http://tempuri.org/, но для опубликованных веб-служб XML должно использоваться постоянное пространство имен.

Веб-служба XML должна определяться управляемым пространством имен. Например, в качестве части пространства имен можно использовать пространства имен веб-служб XML похожи на адреса URL, они не обязательно указывают на фактические ресурсы в Интернете. (Пространства

Для создания веб-служб XML с помощью ASP.NET применяемое по умолчанию пространство имен может быть изменено посредством свойства N WebService применяется к классу, содержащему XML-методы веб-службы. Далее приводится пример программного кода, который задает для /webservices/»:

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // реализация
}
```

Visual Basic

```
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
    ' реализация
End Class
```

C++

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public ref class MyWebService {
    // реализация
};
```

Дополнительные сведения о пространствах имен XML см. в рекомендациях W3C по адресу Namespaces in XML.

Дополнительные сведения о WSDL см. по адресу WSDL Specification.

Дополнительные сведения об URI см. по адресу RFC 2396.

Не потребовалось писать дополнительную логику для реализации веб-сервиса (за исключением указания атрибута [**WebMethod**]).

Атрибут [**WebService**] обозначает класс, который будет представлен как веб-сервис. Файл **asmx** может содержать несколько классов, но только один их них может быть представлен как веб-сервис.

Класс веб-сервиса и все его методы, доступные в веб-сервисе, должны быть объявлены с модификаторам доступа **public**. Если добавить атрибут [**WebMethod**] к приватному методу, мы не получим сообщения об ошибке. Данный метод просто будет недоступен в веб-сервисе и его будет невозможно вызвать.

Создание WPF-приложения – потребителя веб-сервиса

Создадим WPF-приложение – потребителя сервиса – **ASMXClient**:

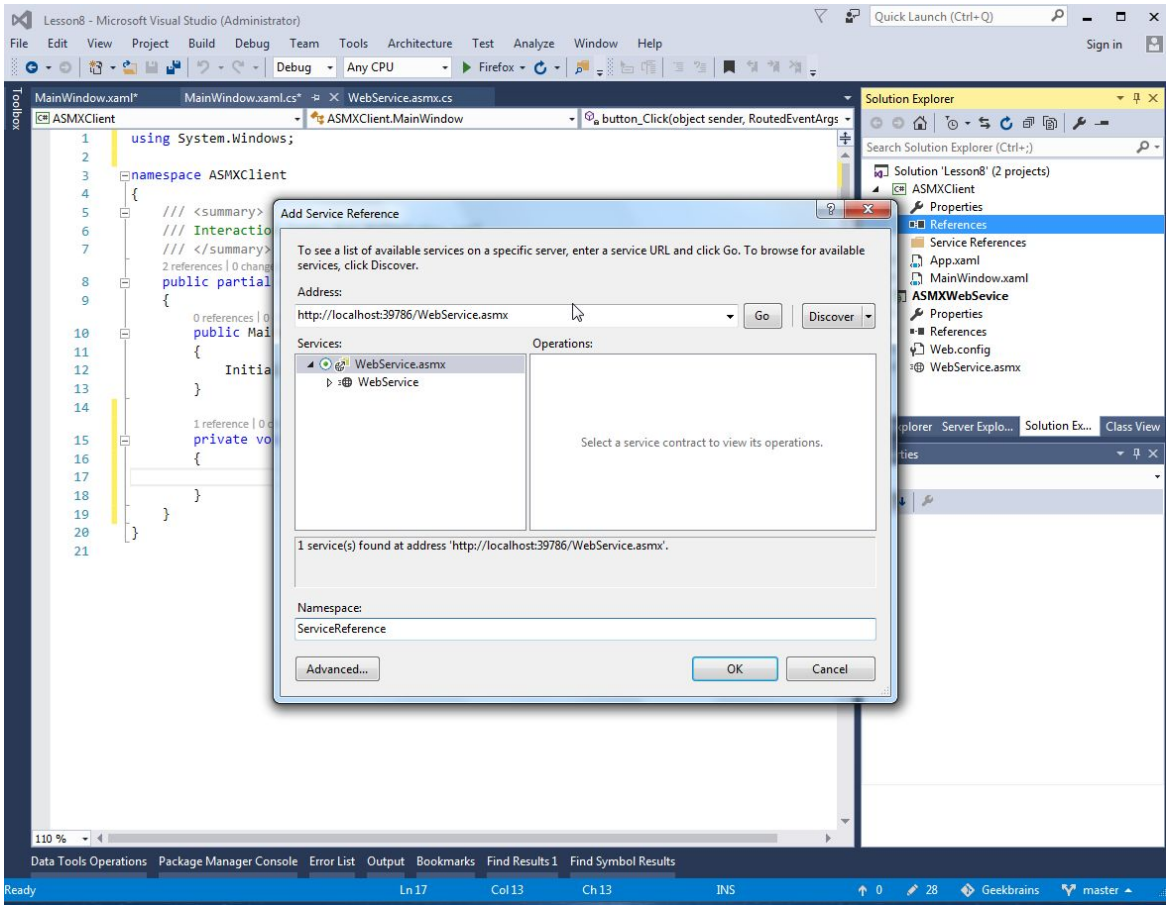
MainWindow.xaml

```
<Window x:Class="ASMXClient.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ASMXClient"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBox x:Name="textBox1" Grid.Column="0" HorizontalAlignment="Center"
Height="23" TextWrapping="Wrap"  VerticalAlignment="Center" Width="120"
Margin="10"/>
        <TextBox x:Name="textBox2" Grid.Column="1" HorizontalAlignment="Center"
Height="23"  TextWrapping="Wrap"  VerticalAlignment="Center" Width="120"
Margin="10"/>
        <Button x:Name="button" Grid.Row="1" Content="Вычислить"
HorizontalAlignment="Center"  VerticalAlignment="Center" Width="75" Margin="10"
Click="button_Click"/>
        <TextBlock x:Name="textBlock" Grid.Column="1" Grid.Row="1"
HorizontalAlignment="Center"  TextWrapping="Wrap" VerticalAlignment="Center"
Margin="10"/>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;
namespace ASMXClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void button_Click(object sender, RoutedEventArgs e)
        {
            ServiceReference.WebServiceSoapClient serviceClient = new
ServiceReference.WebServiceSoapClient();
            string sum = serviceClient.Summ(textBox1.Text, textBox2.Text);
            textBlock.Text = sum;
        }
    }
}
```

Добавим ссылку на веб-сервис в проект:



WCF-сервис

Характерные особенности:

- Основывается на протоколе **SOAP**;
- Передача данных осуществляется в формате **XML**;
- Поддерживает различные транспортные протоколы (**TCP**, **HTTP**, **HTTPS**, **Named Pipes**, **MSMQ**);
- Не является open source-технологией, но с сервисом могут работать клиентские приложения, понимающие **xml**;
- Может быть развернут на веб-сервере **IIS** внутри произвольного приложения или Windows-сервиса.

Все WCF-сервисы предоставляют контракты – это платформонезависимые и стандартные способы описания сервисов. Для **WCF** определены 4 типа контрактов:

- **Контракты сервиса [ServiceContract]** – определяют, какие операции клиент может запрашивать у сервиса;
- **Контракты данных [DataContract]** – определяют, какие типы данных передаются от клиента к сервису и обратно. **WCF** определяет неявные контракты для встроенных типов (**int**, **string** и т.п.), но для пользовательских типов нужно указывать явные контракты;

- **Контракты ошибок [FaultContract]** – определяют, какие ошибки генерируются сервисом и как они обрабатываются и передаются клиенту;

- **Контракты сообщений [MessageContract]** – определяют контракт сообщения для типа (сопоставляют тип и конвертер SOAP).

Значения параметров подключений WCF-сервисов сгруппированы в наборы, которые называются привязками (**bindings**). Они включают в себя определение транспортного протокола, алгоритма кодирования сообщений, способ взаимодействия, защищенность, управление транзакциями (все, что нужно программисту, это выбрать один из вариантов привязки). При этом WCF установит для сервиса множество значений, соответствующих данной привязке.

Основные стандартные варианты привязок

- **Базовая привязка**. Соответствует **ASMX** веб-сервисам и позволяет WCF-сервисам работать аналогично им;

- **TCP-привязка**. Использует протокол **TCP** для обмена сообщениями. Сервис и клиент должны использовать **WCF**;

- **IPC-привязка**. Использует именованные каналы (**named pipes**) при работе клиента и сервиса на одном компьютере;

- **Веб-сервис привязка.** Использует протокол **HTTP** и **HTTPS**. Позволяет организовывать взаимодействие с произвольными клиентами, поддерживающими технологию веб-сервисов;

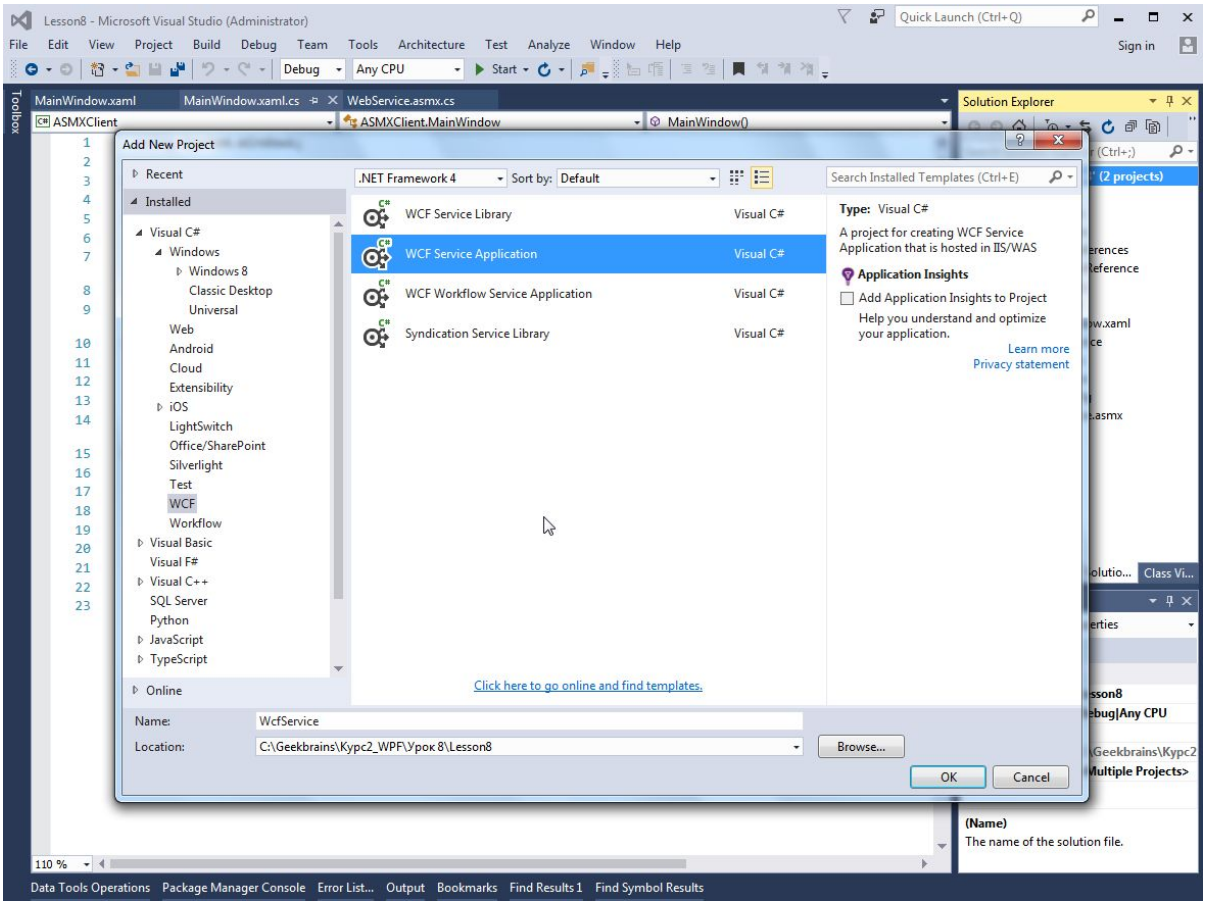
- **MSMQ**-привязка. Использует **MSMQ** в качестве транспортного протокола.

Комбинация из адреса сервиса, его контракта и привязки определяют его работу. Данная комбинация в **WCF** обозначается термином «оконечная точка». Каждая оконечная точка имеет уникальный адрес, и каждый отдельный сервис может предоставлять несколько оконечных точек. Они могут использовать различную привязку и разные контракты.

Создание WCF-сервиса

Клиентское приложение **WPF** выполняет расчет числа дней от указанной даты, используя веб-сервис. Клиентское приложение и веб-сервис развернуты на одном компьютере.

Создадим WCF-сервис, выполняющий расчет числа дней от указанной даты. Для этого выберем шаблон проекта **WCF-service application**:



IService.cs содержит пример описания 3 типов контрактов **WCF**.

```
[ServiceContract]
    public interface IService1
    {
        [OperationContract]
        string GetData(int value);
        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);
        // TODO: Add your service operations here
    }
      // Use a data contract as illustrated in the sample below to add composite
types to service operations.
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";
        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
            set { boolValue = value; }
        }
        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
```

Service1.svc.cs содержит реализацию интерфейса **IService**.

Добавим в **IServices1.cs** описание метода для расчета количества дней с определенной даты:

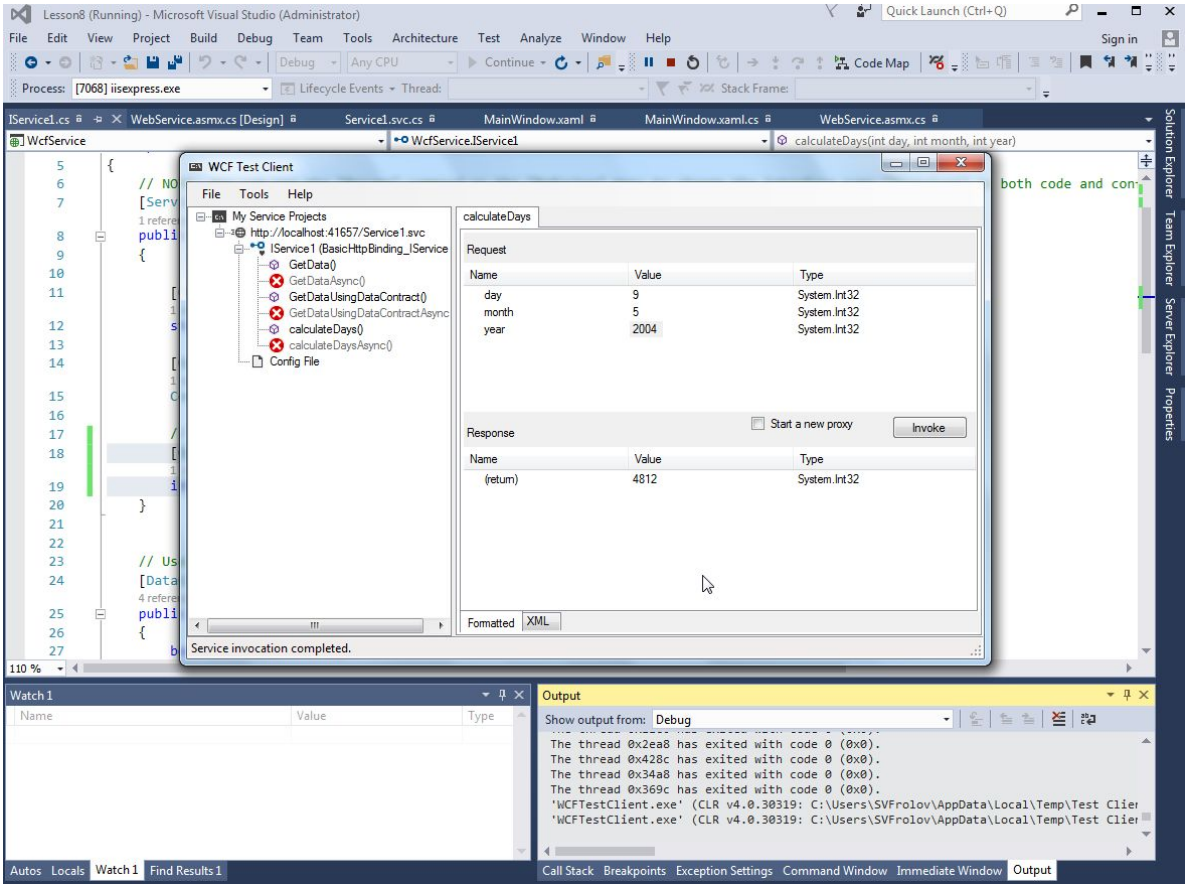
```
        [OperationContract]

        int calculateDays(int day, int month, int year);
```

Реализуем данный метод в файле **Service1.svc.cs**:

```
public class Service1 : IService1
   {
        public int calculateDays(int day, int Month, int year)
        {
            DateTime dt = new DateTime(year, Month, day);
            int datetodays = DateTime.Now.Subtract(dt).Days;
            return datetodays;
        }
```

В дереве проекта выбираем файл **Service.svc** и нажимаем **F5**.



В меню «Файл» – «Добавить службу» можно скопировать путь к службе.

http://localhost:41657/Service1.svc.

Создание WPF-приложения – потребителя WCF-сервиса

Создадим WPF-приложение – потребителя WCF-сервиса – **WcfClient**.

MainWindow.xaml

```
<Window x:Class="WcfClient.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WcfClient"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="250">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBlock Text="День" Grid.Column="0" Grid.Row="0"
HorizontalAlignment="Center"  TextWrapping="Wrap" VerticalAlignment="Center"
Margin="10"/>
        <TextBlock Text="Месяц" Grid.Column="0" Grid.Row="1"
HorizontalAlignment="Center"  TextWrapping="Wrap" VerticalAlignment="Center"
Margin="10"/>
        <TextBlock Text="Год" Grid.Column="0" Grid.Row="2"
HorizontalAlignment="Center"  TextWrapping="Wrap" VerticalAlignment="Center"
Margin="10"/>
        <TextBox x:Name="dayTextBox" Grid.Column="1" Grid.Row="0"
HorizontalAlignment="Center" Height="23" TextWrapping="Wrap"
VerticalAlignment="Center" Width="120" Margin="10"/>
        <TextBox x:Name="monthTextBox" Grid.Column="1" Grid.Row="1"
HorizontalAlignment="Center" Height="23" TextWrapping="Wrap"
VerticalAlignment="Center" Width="120" Margin="10"/>
        <TextBox x:Name="yearTextBox" Grid.Column="1" Grid.Row="2"
HorizontalAlignment="Center" Height="23" TextWrapping="Wrap"
VerticalAlignment="Center" Width="120" Margin="10"/>
        <Button  Grid.Row="3"  Content="Вычислить" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="75" Margin="10" Click="button_Click"/>
        <TextBlock x:Name="resultTextBlock" Text="Дней: " Grid.Column="1"
Grid.Row="3"  HorizontalAlignment="Center"  TextWrapping="Wrap"
VerticalAlignment="Center" Margin="10"/>
    </Grid>
</Window>
```

MainWindow.xaml.cs

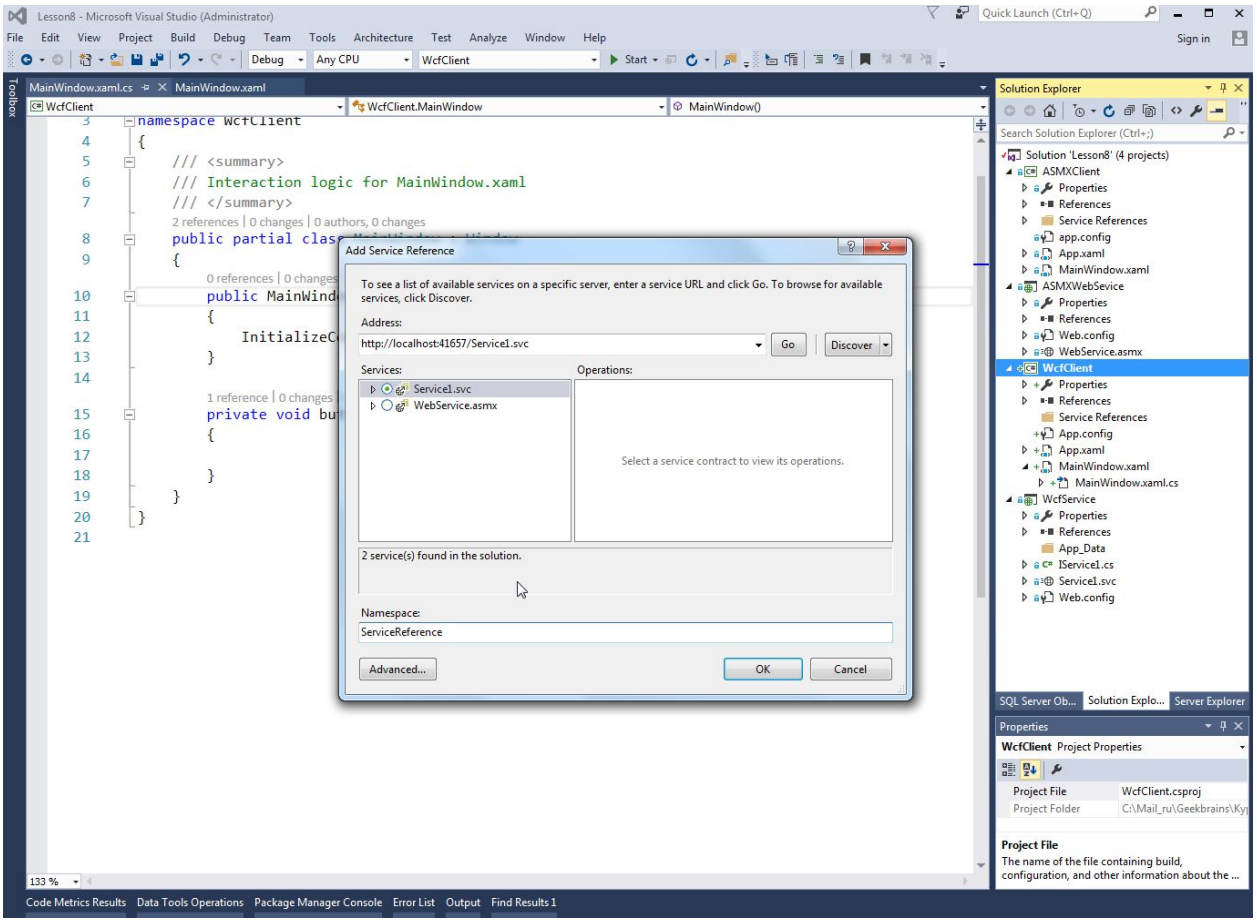
```
using System;
using System.Windows;
namespace WcfClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void button_Click(object sender, RoutedEventArgs e)
        {
                                    ServiceReference.Service1Client   service   =   new
ServiceReference.Service1Client();
            try
            {
                                                                    resultTextBlock.Text    +=
service.calculateDays(Convert.ToInt32(dayTextBox.Text),

Convert.ToInt32(monthTextBox.Text),

Convert.ToInt32(yearTextBox.Text)).ToString();
            }
            catch(Exception)
            {
            }
        }
    }
}
```

Добавим ссылку на WCF-сервис в проект:



Web API сервис

- В качестве транспортного протокола используется только **HTTP**;
- Open source-технология для построения **RESTful**-сервисов;
- Может быть развернут на веб-сервере IIS или внутри произвольного приложения;
- Передача данных может осуществляться в **ASCII**, **XML**, **JSON** или любых других форматах, распознаваемых одновременно и клиентом, и сервером.

RESTful

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем. Системы, поддерживающие **REST**, называются **RESTful**-системами.

Любой объект, с которым работает **RESTful**-сервис, может быть представлен как ресурс. Ресурсом может быть любой объект, который интересен клиентскому приложению, пользующемуся сервисом. Каждому ресурсу соответствует, как минимум, один **URL**.

Доступ к ресурсам осуществляется через стандартный HTTP-интерфейс.

Протокол **HTTP** предназначен не только работы с веб-страницами, но и для построения **API**. **HTTP** прост, гибок и повсеместно распространен. Практически любая платформа содержит поддержку

HTTP-протокола, поэтому HTTP-сервисы доступны в различных клиентах, включая браузеры, мобильные устройства и традиционные десктопные приложения.

Для реализации RESTful-сервиса в протоколе **HTTP** используется 4 основных метода: **GET**, **POST**, **PUT**, **DELETE**.

Вызов метода **GET** – это запрос информации о ресурсе. Ответ возвращается в виде заголовка и представления. Клиентское приложение никогда не посылает представление объекта вместе с вызовом метода **GET**.

Вызов метода **PUT** сообщает об изменении состояния ресурса. Обычно клиентское приложение отправляет представление объекта в методе **PUT**, и сервис в ответ на запрос создает или изменяет ресурс так, чтобы состояние объекта соответствовало представлению.

Вызов метода **DELETE** сообщает, что ресурс должен быть удален. Клиентское приложение никогда не посылает представление объекта в вызове метода **DELETE**.

Вызов метода **POST** является попыткой создания нового ресурса из существующего. Представление объекта в вызове метода **POST** описывает начальное состояние нового ресурса.

Для RESTful-сервисов можно сформулировать два типа состояний объектов: состояние объекта ресурса и состояние объекта приложения. Первое хранится на сервере и направляется клиенту в виде представления. Второе хранится в приложении до передачи ресурсу. Когда состояние объекта приложения отправляется серверу с помощью запросов **POST**, **PUT** и **DELETE**, оно становится состоянием объекта ресурса.

RESTful-сервисы не имеют состояния, если сервер никогда не сохранял состояние объекта приложения. Для приложений, не хранящих состояние объекта, каждый запрос клиентского приложения выполняется независимо от других.

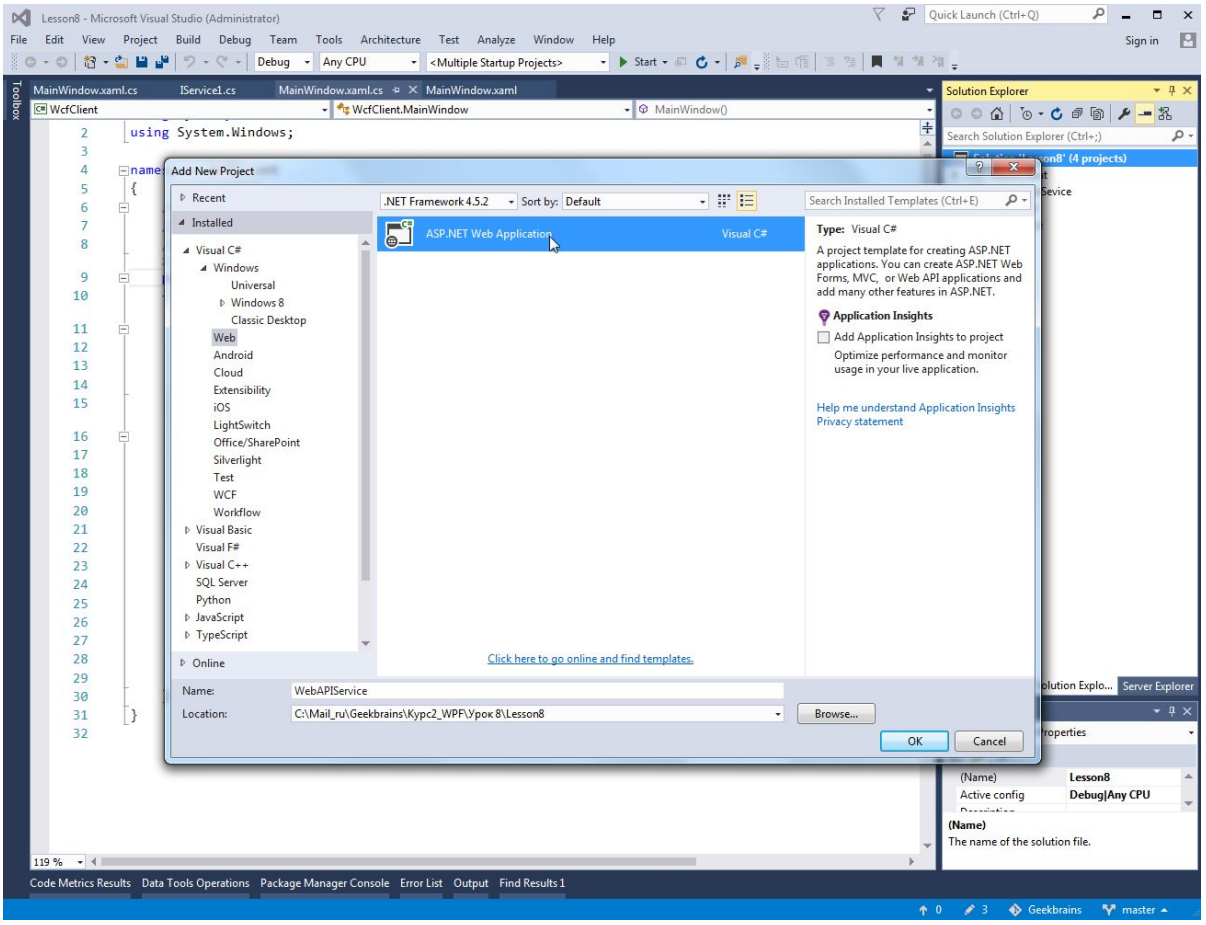
Если клиентское приложение хочет, чтобы учитывалось состояние объекта приложения, необходимо передавать его в каждом вызове (например, передавать так параметры авторизации).

Клиентское приложение может передавать состояние объектов приложения с помощью методов **PUT** и **POST**. Метод **DELETE** работает аналогично, но без передачи представления.

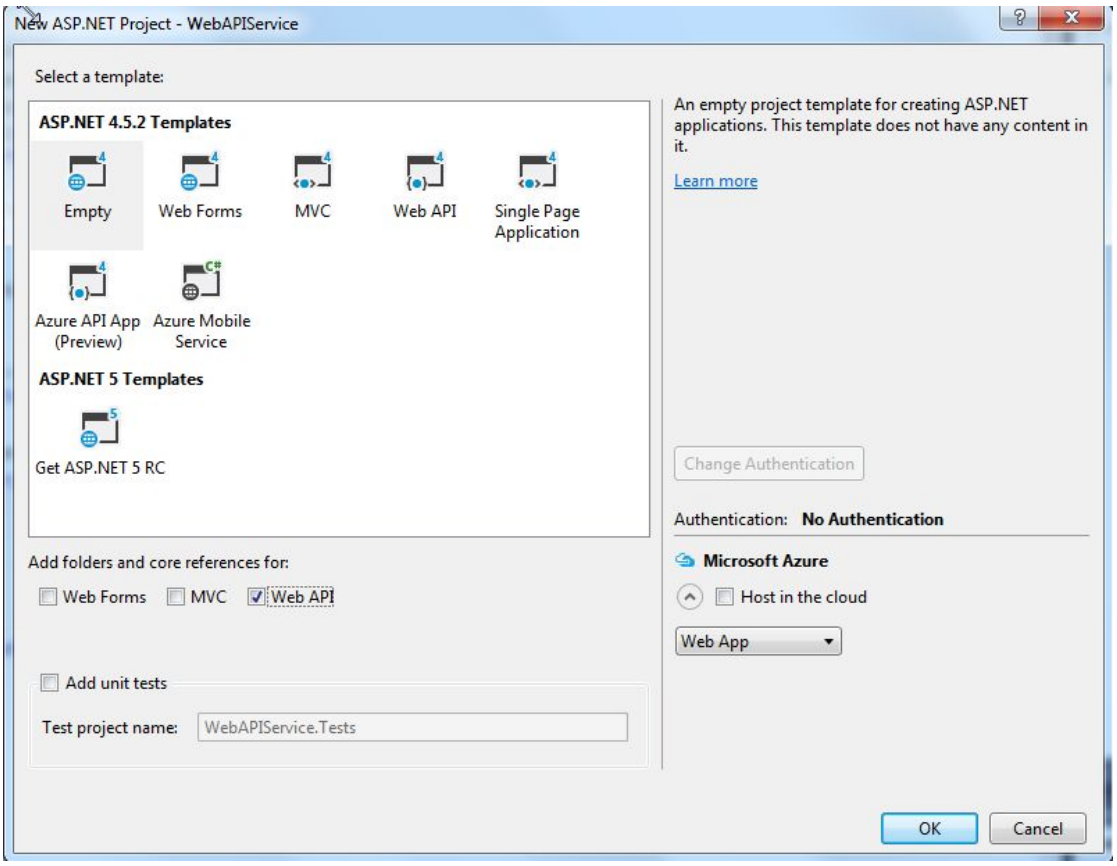
ASP.NET Web API – это фреймворк для построения **Web-API** сервиса на базе **.NET Framework**. В следующем примере **ASP.NET Web API** используется для создания **Web-API** сервиса, который возвращает список товаров.

Создание Web API сервиса

В **VisualStudio** создаем новый проект, используя шаблон **ASP.NET Web Application**. Назовем его **WebAPIService**:



Среди шаблонов **ASP.NET** выбираем «**Empty**» и отмечаем чекбокс «**Web API**» в разделе «**Add folders and core references for**»:



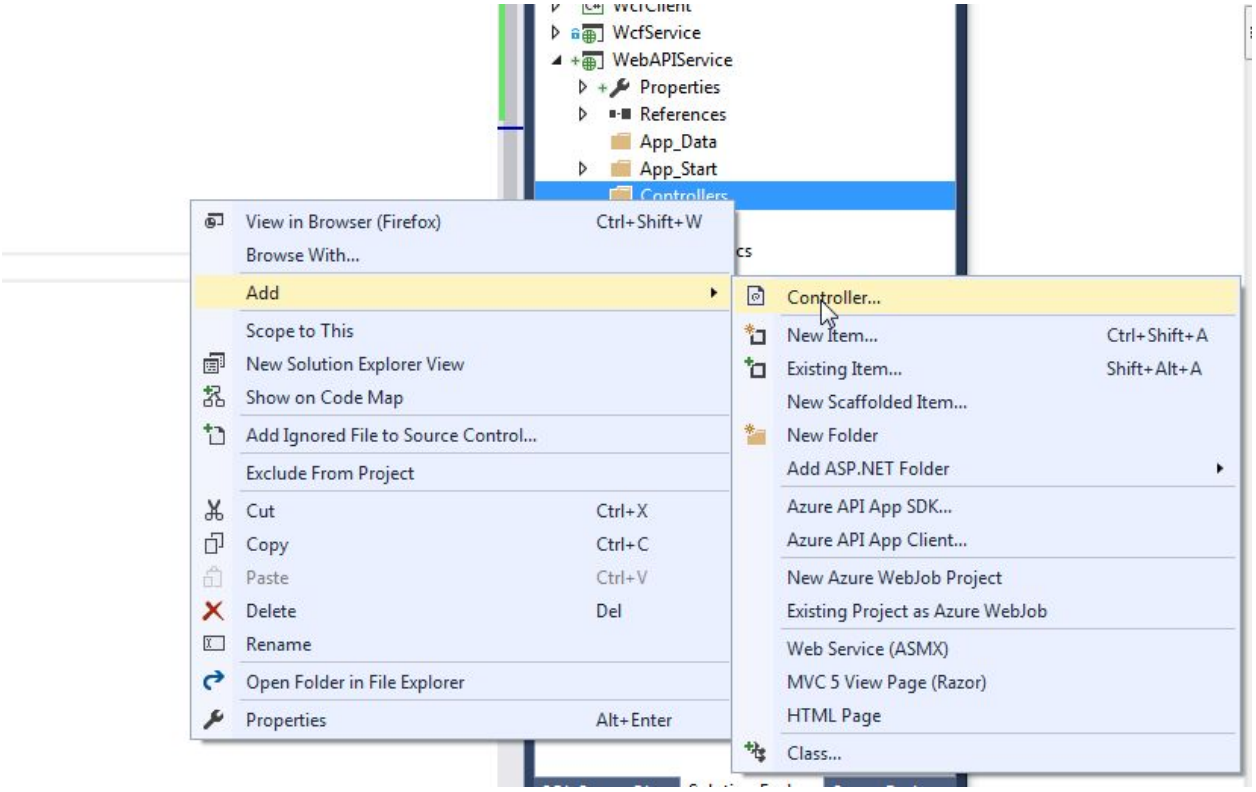
Модель – это объект, который представляет данные в приложении. **ASP.NET Web API** может автоматически сериализовать модель в **JSON**, **XML** и некоторые другие форматы, а затем записать сериализованные данные в тело ответного сообщения **HTTP**. Клиентское приложение может прочитать сериализованные данные и десериализовать его. Большинство клиентских приложений имеет возможность обрабатывать **XML** и **JSON**. При этом клиентское приложение может указывать в заголовке HTTP-сообщения запроса, в каком формате оно ожидает получить ответ.

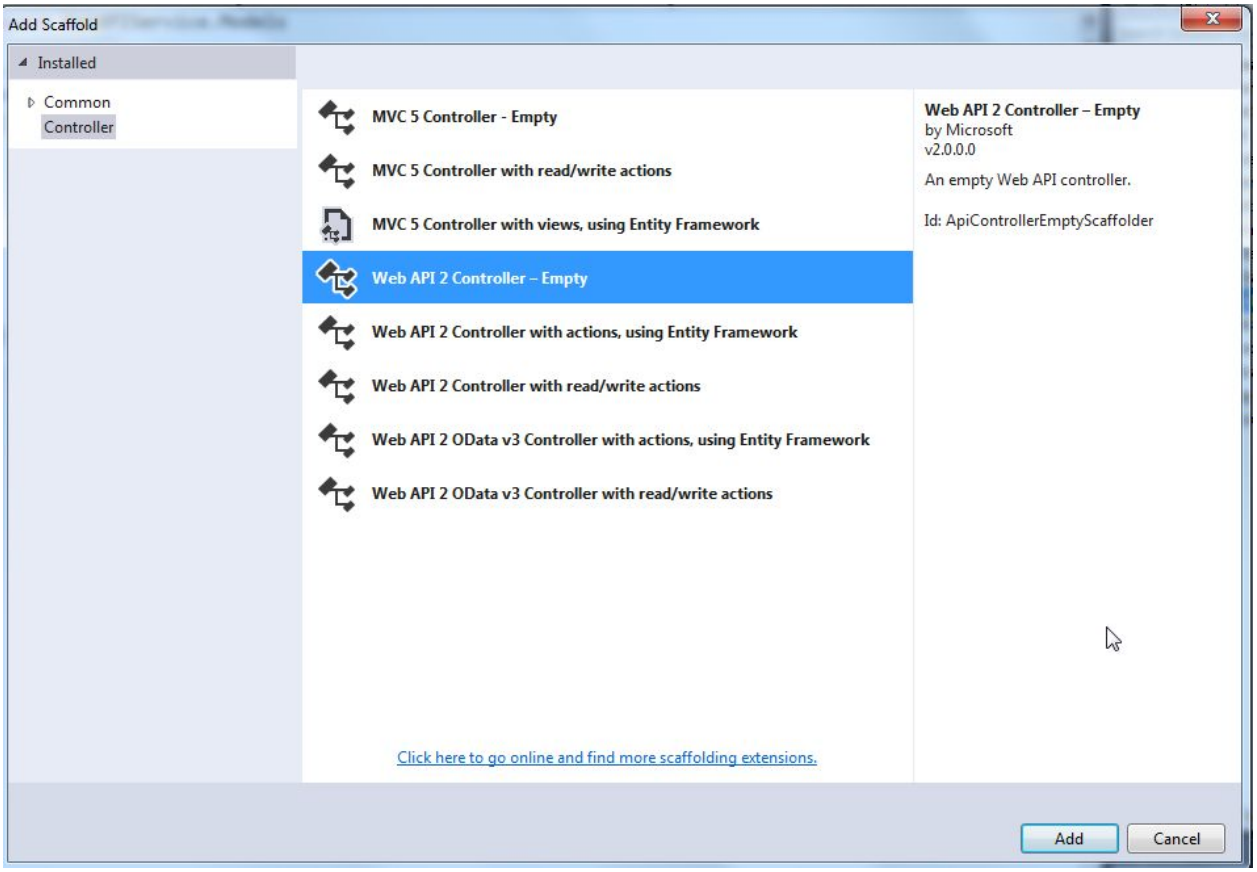
Добавляем класс модели в папку **Models** в проекте **WebAPIService**. Класс называется «**Product**»:

Products.cs

```
namespace WebAPIService.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}
```

В **Web API** контроллером называется объект, который обрабатывает HTTP-запросы. Добавим в проект (в папку Controllers) контроллер, который возвращает либо список товаров, либо товар с заданным ID:





Назовем контроллер **ProductsController**.

Продукты будут храниться в массиве фиксированной длины. Определим в контроллере два метода: **IEnumerable<Product> GetAllProducts()** и **IHttpActionResult GetProduct(int id)**.

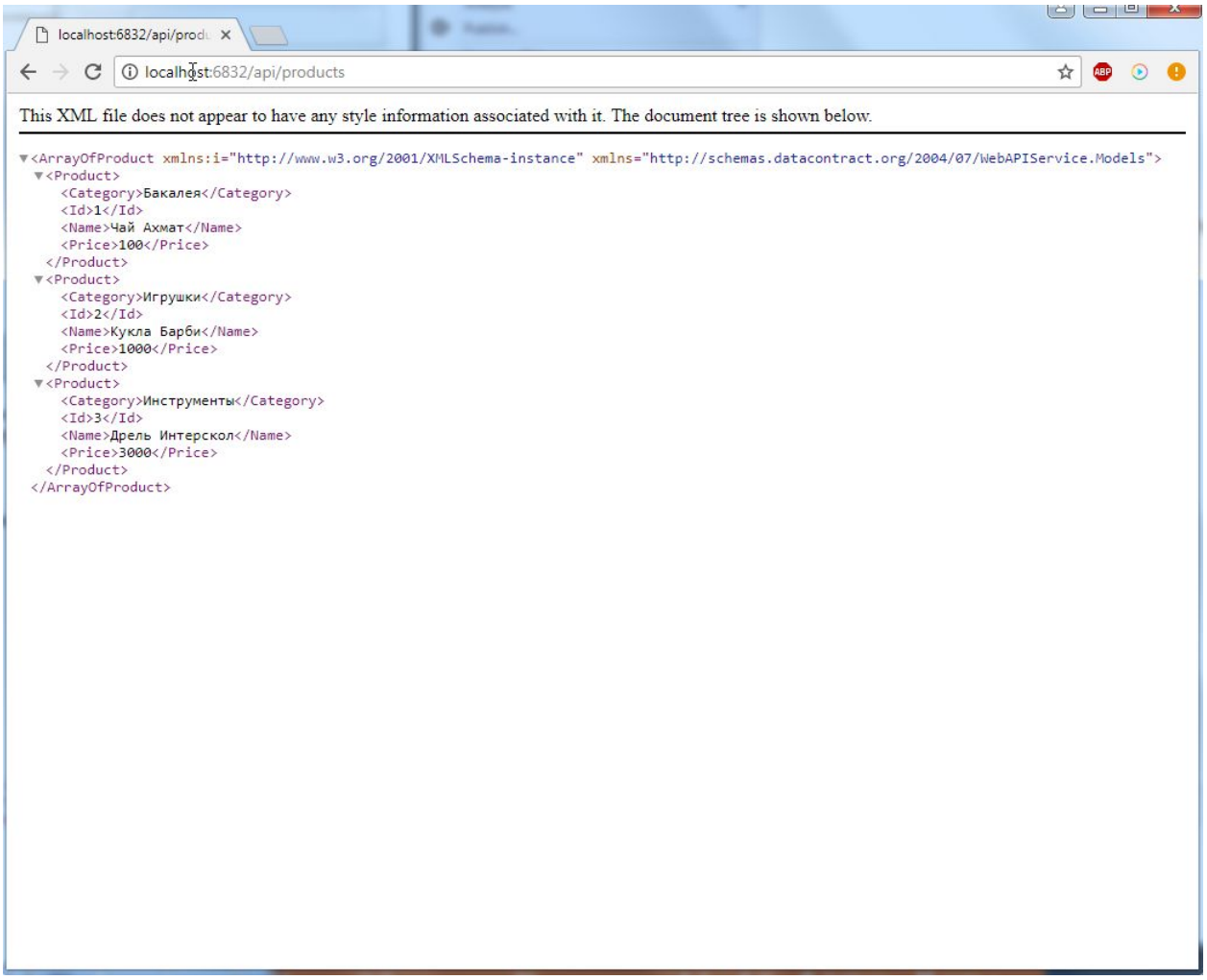
Каждый из методов с одним или более **URL**.

- **IEnumerable<Product> GetAllProducts() – /api/products;**
- **IHttpActionResult GetProduct(int id) – /api/products/id**, где **id** –переменная, соответствующая **id** товара.

ProductsController.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using WebAPIService.Models;
namespace WebAPIService.Controllers
{
    public class ProductsController : ApiController
    {
        Product[] products = new Product[]
        {
          new Product { Id = 1, Name = "Чай Ахмат", Category = "Бакалея", Price =
100 },
         new Product { Id = 2, Name = "Кукла Барби", Category = "Игрушки", Price =
1000 },
         new Product { Id = 3, Name = "Дрель Интерскол", Category = "Инструменты",
Price = 3000 }
        };
        public IEnumerable<Product> GetAllProducts()
        {
            return products;
        }
        public IHttpActionResult GetProduct(int id)
        {
            var product = products.FirstOrDefault((p) => p.Id == id);
            if (product == null)
            {
                return NotFound();
            }
            return Ok(product);
        }
    }
}
```

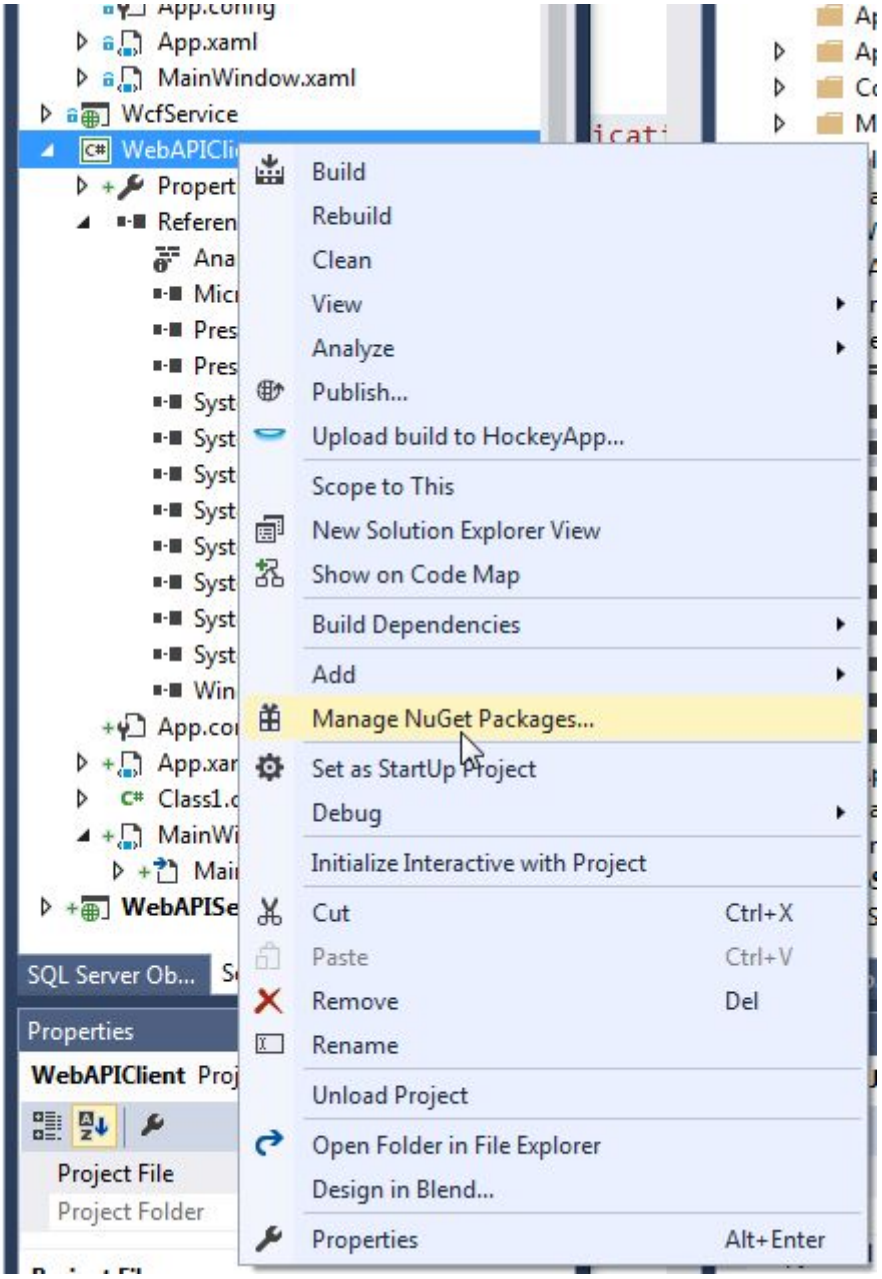
Запустим приложение **WebAPIService**. В результате откроется браузер и попытается обратиться к корневому элементу нашего сервиса. Дополним **URL** в строке адреса браузера одним из реализованных в сервисе URL. Например, **/api/products**. После этого обновим страницу в браузере:

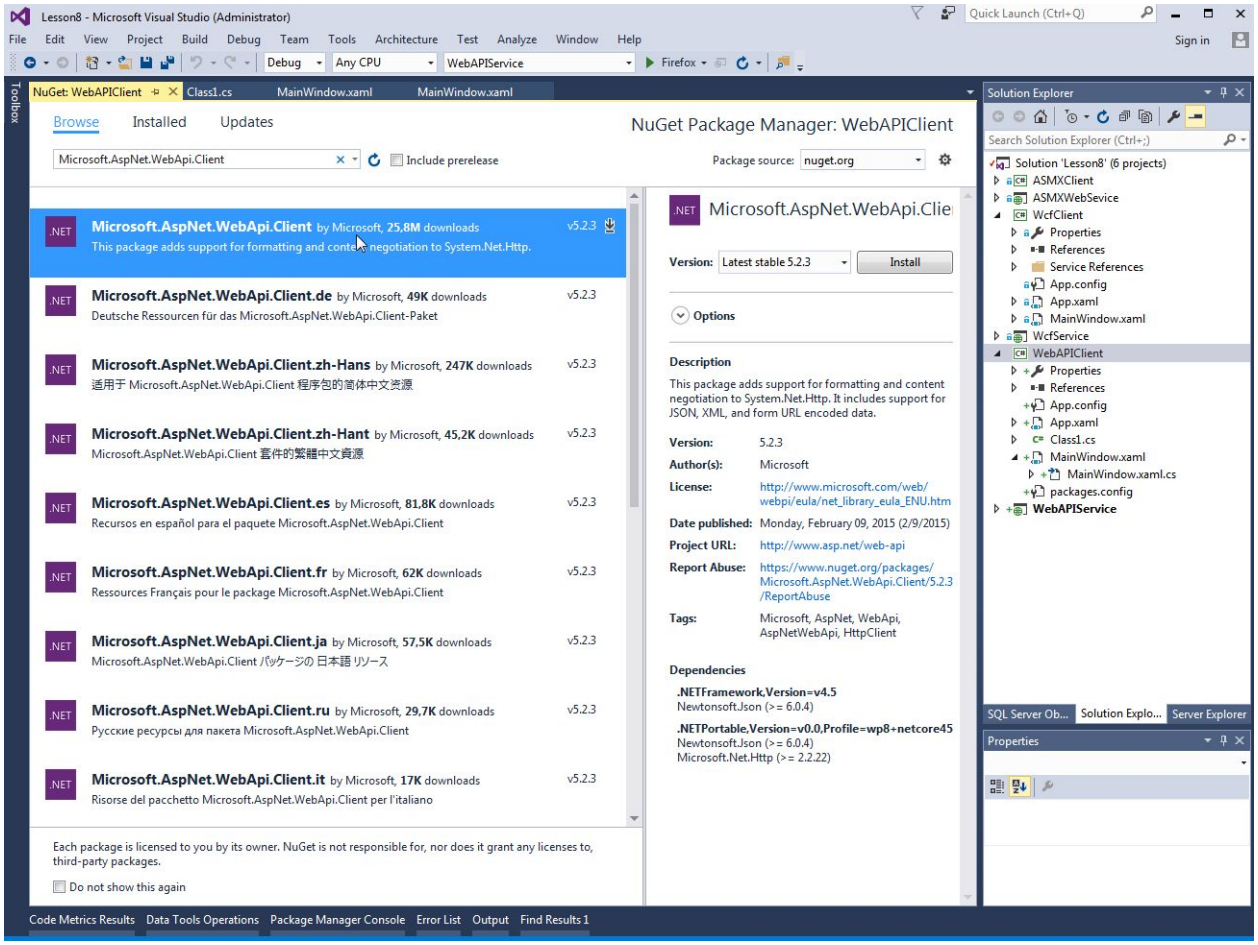

localhost:6832/api/prod
localhost:6832/api/products
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<ArrayOfProduct xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/WebAPIService.Models">
<Product>
<Category>Бакалея</Category>
<Id>1</Id>
<Name>Чай Ахмат</Name>
<Price>100</Price>
</Product>
<Product>
<Category>Игрушки</Category>
<Id>2</Id>
<Name>Кукла Барби</Name>
<Price>1000</Price>
</Product>
<Product>
<Category>Инструменты</Category>
<Id>3</Id>
<Name>Дрель Интерскол</Name>
<Price>3000</Price>
</Product>
</ArrayOfProduct>

Создание WPF-приложения – потребителя Web API сервиса

Создадим WPF-приложение – потребителя Web-API сервиса, **WebAPIClient**.

Используя **NuGet**, добавим в проект пакет **Microsoft.AspNet.WebApi.Client** для поддержки сериализации **XML** и **JSON**:





Products.cs

```
namespace WebAPIClient
{
    public class Product
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
        public string Category { get; set; }
    }
}
```

MainWindow.xaml

```
<Window x:Class="WebAPIClient.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WebAPIClient"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="350">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="5*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <DataGrid x:Name="productDataGrid" Grid.ColumnSpan="2"
AutoGenerateColumns="False" EnableRowVirtualization="True" ItemsSource="{Binding}"
Margin="10" IsReadOnly="True" >
            <DataGrid.Columns>
                <DataGridTextColumn x:Name="idColumn" Binding="{Binding Id}"
Header="Id" IsReadOnly="True" Width="Auto"/>
                <DataGridTextColumn x:Name="nameColumn" Binding="{Binding Name}"
Header="Название" Width="Auto"/>
                <DataGridTextColumn x:Name="priceColumn" Binding="{Binding Price}"
Header="Цена" Width="Auto"/>
                <DataGridTextColumn x:Name="categoryColumn" Binding="{Binding
Category}" Header="Категория" Width="Auto"/>
            </DataGrid.Columns>
        </DataGrid>
        <TextBlock Text="Id товара" Grid.Row="1"  Margin="10"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <StackPanel Grid.Row="1" Grid.Column="1" Orientation="Horizontal" >
            <TextBox x:Name="idproductTextBox" HorizontalAlignment="Left"
Height="23" Margin="10" Grid.Row="1" VerticalAlignment="Center" Width="120"/>
            <Button x:Name="idproductButton" Content="Запросить товар(ы)"
Margin="10" Click="idproductButton_Click"  Grid.Column="1" Grid.Row="1"
Width="Auto" Height="30" HorizontalAlignment="Center" VerticalAlignment="Center"/>
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using System.Windows;
namespace WebAPIClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        static HttpClient client = new HttpClient();
        public MainWindow()
        {
            InitializeComponent();
            client.BaseAddress = new Uri("http://localhost:6832/");
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        }
        private async void allproductButton_Click(object sender, RoutedEventArgs
e)
        {
            IEnumerable<Product> products = await
GetProductsAsync(client.BaseAddress + "api/Products");
            productDataGrid.ItemsSource = products;
        }
        private async void idproductButton_Click(object sender, RoutedEventArgs e)
        {
            List<Product> products = new List<Product>();
            if (idproductTextBox.Text != String.Empty)
            {
                Product product = await GetProductAsync(client.BaseAddress +
"api/Products/" + idproductTextBox.Text);
                if(product != null)
                    products.Add(product);
            }
            else
            {
                products = (List<Product>)await
GetProductsAsync(client.BaseAddress + "api/Products");
            }
            productDataGrid.ItemsSource = products;
        }
        static async Task<IEnumerable<Product>> GetProductsAsync(string path)
```

```
        {
            IEnumerable<Product> products = null;
            try
            {
                HttpResponseMessage response = await client.GetAsync(path);
                if (response.IsSuccessStatusCode)
                {
                    products = await
response.Content.ReadAsAsync<IEnumerable<Product>>();
                }
            }
            catch(Exception) {
            }
            return products;
        }
        static async Task<Product> GetProductAsync(string path)
        {
            Product product = null;
            try {
                HttpResponseMessage response = await client.GetAsync(path);
                if (response.IsSuccessStatusCode)
                {
                    product = await response.Content.ReadAsAsync<Product>();
                }
            }
            catch(Exception) {
            }
            return product;
        }
    }
}
```

Локальный веб-сервер назначает номер порта для веб-сервиса случайным образом. Поэтому необходимо скорректировать номер порта в коде программы – согласно адресной строке браузера, сформированной при первом запуске веб-сервиса.

```
client.BaseAddress = new Uri("http://localhost:6832/");
```

Практическое задание

Изменить WPF-приложение для ведения списка сотрудников компании (из урока 7), **используя веб-сервисы**. Разделите приложение на две части. Первая часть – клиентское приложение, отображающее данные. Вторая часть – веб-сервис и код, связанный с извлечением данных из БД. Приложение реализует только просмотр данных. При разработке приложения допустимо использовать любой из рассмотренных типов веб-сервисов.

1. Создать таблицы **Employee** и **Department** в БД **MSSQL Server** и заполнить списки сущностей начальными данными;
2. Для списка сотрудников и списка департаментов предусмотреть визуализацию (отображение);
3. Разработать формы для отображения отдельных элементов списков сотрудников и департаментов.

Дополнительные материалы

1. Inside Windows Communication Foundation by Justin Smith.
2. Programming WCF Services, 4th Edition by Juval Lowy, Michael Montgomery.
3. Professional ASP.NET Web Services Paperback – November, 2001 by Andreas Eide (Author), Chris Miller (Author), Bill Sempf.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Inside Windows Communication Foundation by Justin Smith.
2. Programming WCF Services, 4th Edition by Juval Lowy, Michael Montgomery.
3. Professional ASP.NET Web Services Paperback – November, 2001 by Andreas Eide (Author), Chris Miller (Author), Bill Sempf.
4. RESTful Web Services by Leonard Richardson, Sam Ruby, December 2008.