Урок 8

[Обзор сервис-ориентированной архитектуры приложений (SOA). Web-сервис asmx. WCF-сервис. Web-api сервис. 1](#_Toc487969341)

[1. Обзор сервис-ориентированной архитектуры приложений (SOA) 1](#_Toc487969342)

[2. Web-сервис 3](#_Toc487969343)

[2.1. Создание веб-сервиса 4](#_Toc487969344)

[2.2. Создание WPF-приложения, потребителя веб-сервиса 8](#_Toc487969345)

[3. WCF-сервис 10](#_Toc487969346)

[3.1. Создание WCF-сервиса 11](#_Toc487969347)

[3.2. Создание WPF-приложения потребителя WCF-сервиса 14](#_Toc487969348)

[4. Web- API сервис 15](#_Toc487969349)

[4.1. RESTful 16](#_Toc487969350)

[4.2. Создание Web-API сервиса 17](#_Toc487969351)

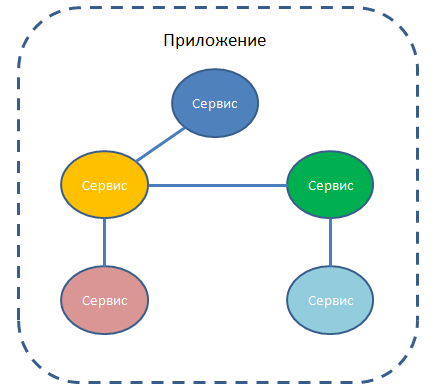
[4.3. Создание WPF-приложения потребителя Web-API сервиса 21](#_Toc487969352)

[5. ДЗ 21](#_Toc487969353)

# Обзор сервис-ориентированной архитектуры приложений (SOA). Web-сервис asmx. WCF-сервис. Web-api сервис.

1. 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 могут быть распределены по разным узлам локальной сети или Интернет. Отдельные сервисы могут разрабатываться разными компаниями, размещаться на разных технологических платформах и находиться в разных часовых поясах. Все эти аспекты скрыты от клиентских приложений, взаимодействующих с сервисами. Клиентские приложения и сервисы могут взаимодействовать друг с другом путем отправки и получения сообщений. Механизм обмена сообщениями между клиентскими приложениями и сервисами сводит на нет разницу между клиентскими приложениями и сервисами, преобразуя входящие и исходящие сообщения к стандартным протоколам сетевого обмена.

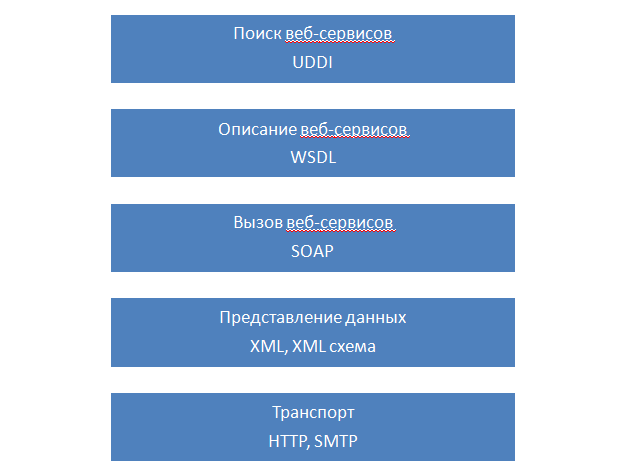
Принципы сервис-ориентированной архитектуры:

* 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 своей 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



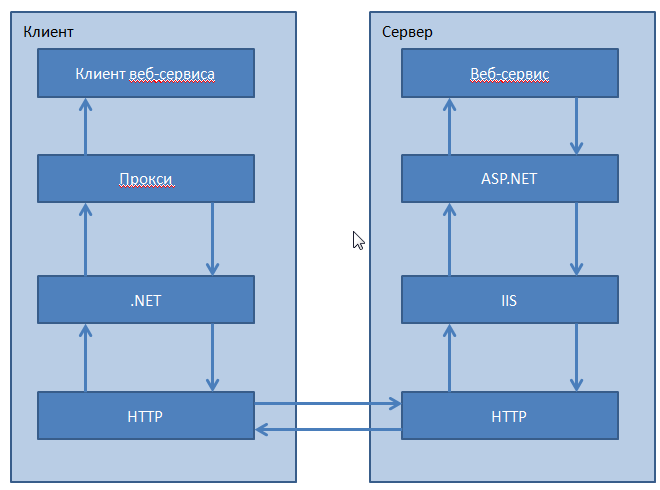
* HTTP, Hypertext Transfer Protocol. HTTP – это протокол, реализующий принцип запрос/ответ.Обычно реализуется поверх TCP/IP протокола. HTTP клиент открывет TCP подключение к серверу и отправляет HTTP запрос. Запрос содержит метод GET или POST. GET используется для получения с сервера файла, например HTML страницы. Для передачи данных используется 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 репозитария веб сервисов и для взаимодействия с репозитарием.
* 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 репозитарий веб сервисов и технология взаимодействия с репозитарием.

1. Web-сервис

Характерные особенности:

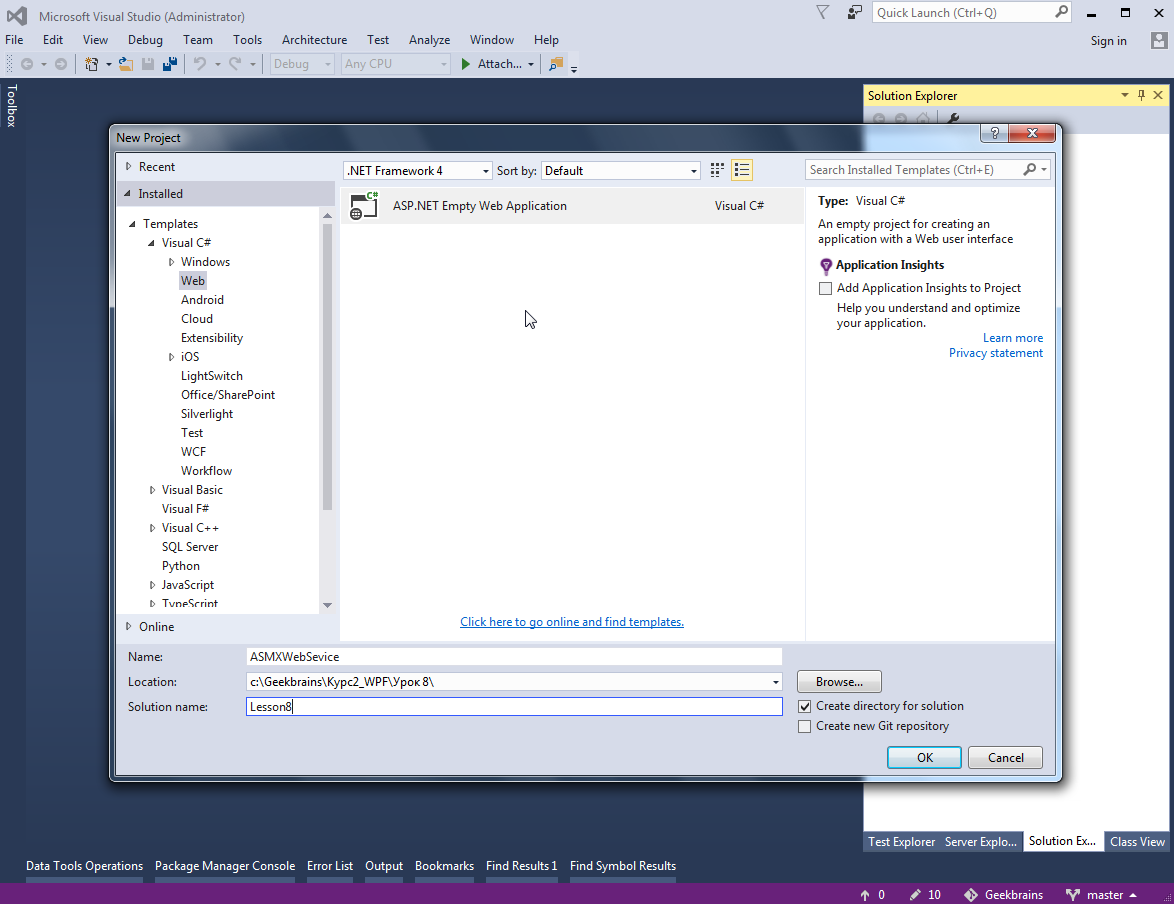
* Основой веб-сервисов является протокол SOAP.
* Формат сериализации данных для веб-сервисов основывается на спецификации XML. Данные, между клиентским приложением и сервисом передаются в XML формате.
* В качестве транспортного протокола используется только HTTP.
* Не является open source технологией, 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-сериса клиентским приложением:

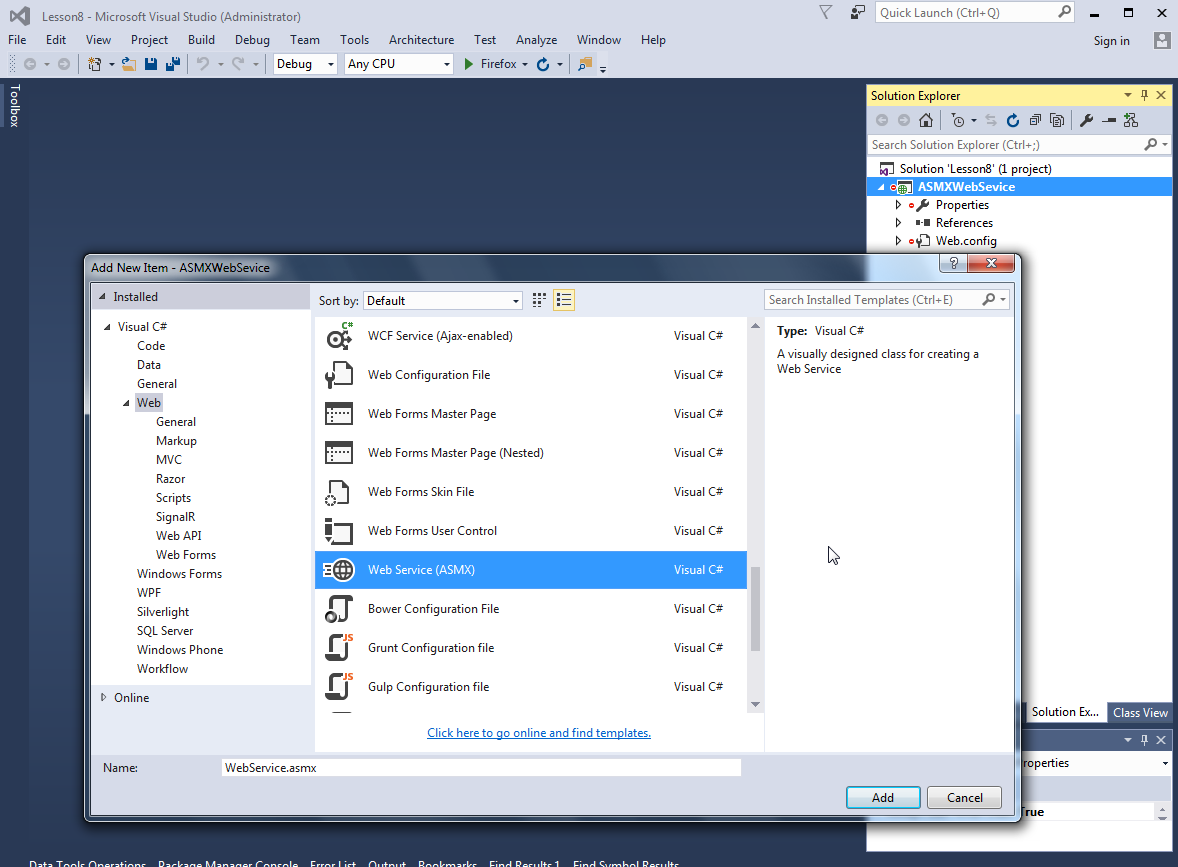


* 1. Создание веб-сервиса

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 создадим пустой (Empty) ASP.Net-проект.   


После этого добавьте в него проект (Add New Item) Web-service ASMX.

  
Visual Studio cформирует 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,

// их анализировать, 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);

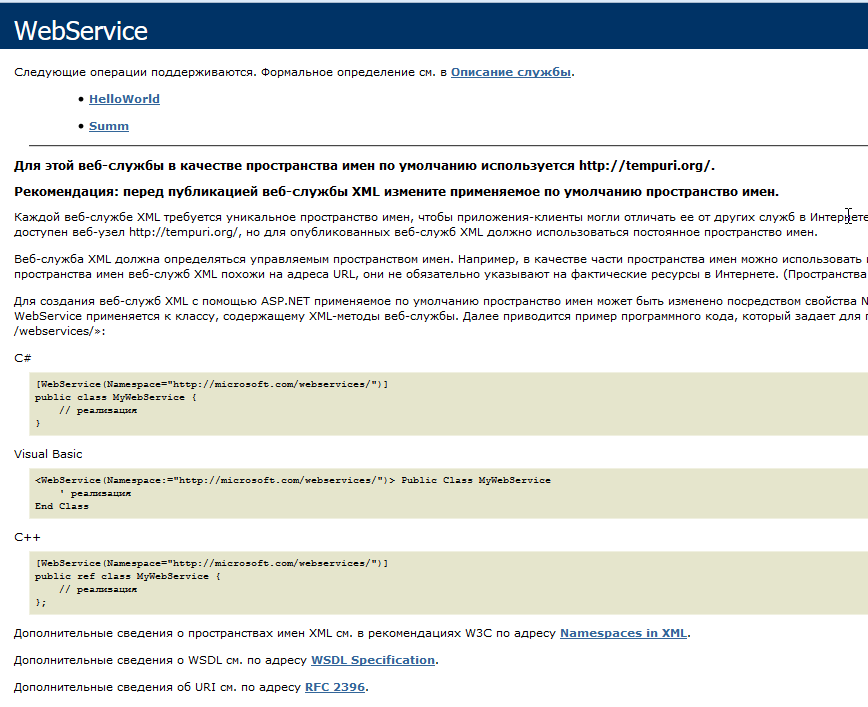
if (flag == false) return "В первом поле должно быть число";

return "Сумма:" + (a + b).ToString();

}

}

}

Выберите в обозревателе решений файл с расширением .asmx и выполните для него команду «Отобразить»,View In Browser . У вас должен будет запуститься браузер и, если всё сделано правильно, появится следующее окно:  


Как можно заметить, нам не потребовалось писать никакой дополнительной логики для реализации веб-сервиса, за исключением указания атрибута [WebMethod].

Атрибут [WebService] обзначает 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, то мы не получим сообщения об ошибке. Данный метод просто будет недоступен в веб-сервисе и его будет невозможно вызвать.

* 1. Создание WPF-приложения, потребителя веб-сервиса

Созадим 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);

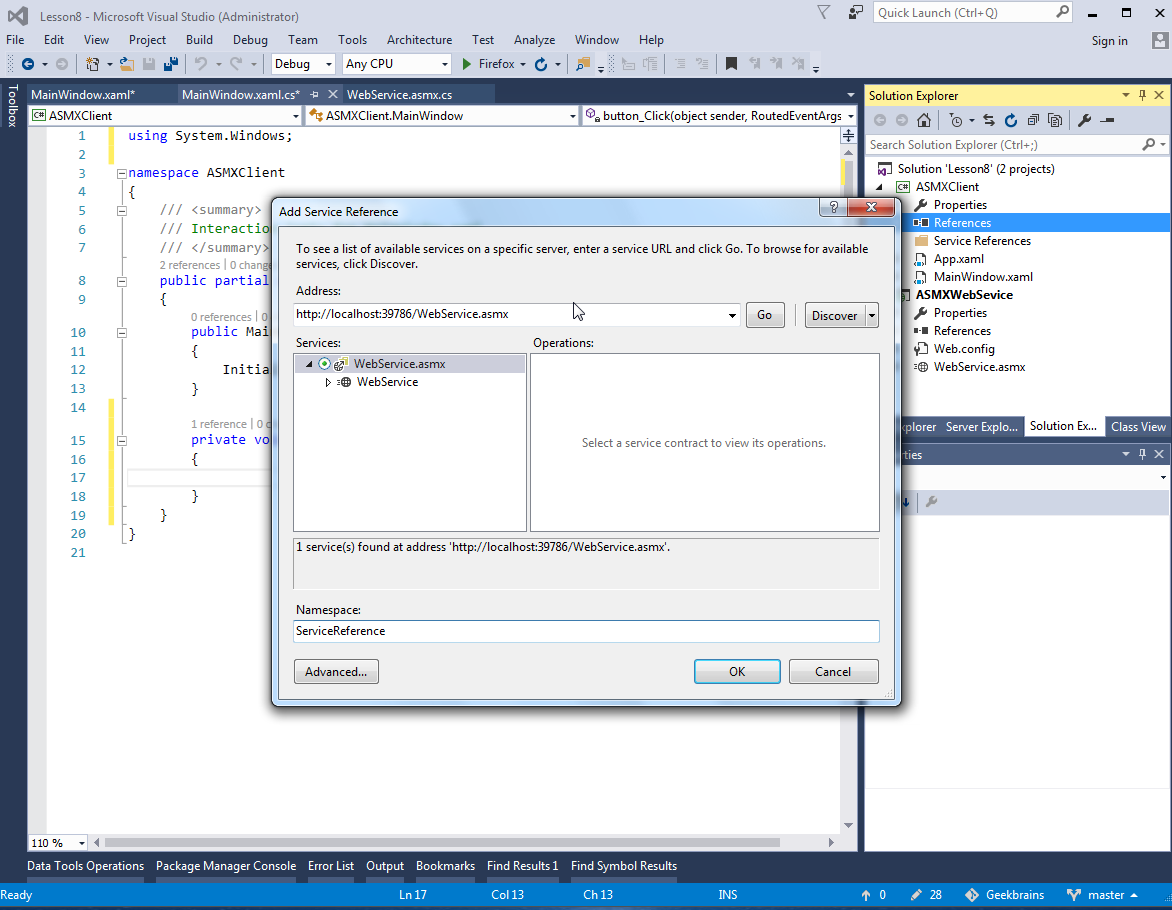
textBlock.Text = sum;

}

}

}

Добавим ссылку на веб-сервис в проект.



1. 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. Контракт – это платформонезависимый и стандартый способ 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 котракты.
* Контракты ошибок [FaultContract]. Определяют какие ошибки генерируются сервисом и как эти ошибки обрабатываются и передаются клиенту.
* Контракты сообщений [MessageContract]. Определяют контракт сообщения для типа, т. е. определяет сопоставление между типом и конвертом SOAP.

Значения параметров подключений WCF-сервисов сгруппированы в наборы, которые называются привязкой(bindings). Привязки 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:

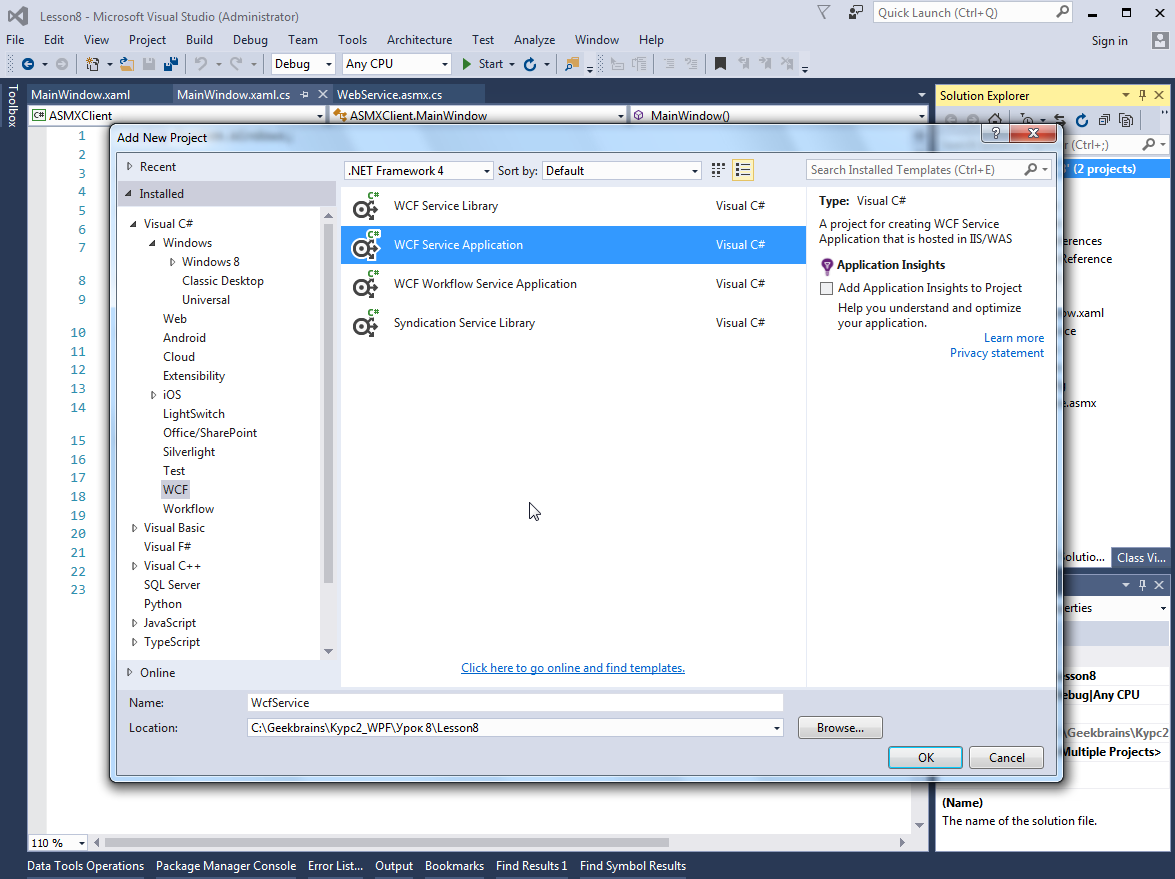
* Базовая привязка. Соответствует ASMX веб-сервисам. Позволяет WCF-сервисам работать аналогично ASMX веб-сервисам.
* TCP привязка. Использует протокол TCP для обмена сообщениями. Сервис и клиент должны использовать WCF.
* IPC привязка. Использует именованные каналы(named pipes) при работе клиента и сервиса 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 работу сервиса. Данная комбинация в WCF обозначается термином «оконечная точка». Каждая оконечная точка имеет уникальный адрес и каждый отдельный сервис может предоставлять несколько оконечных точек. Эти оконечные точки могут использовать различную привязку и различные контракты.

* 1. 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 реализация 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);

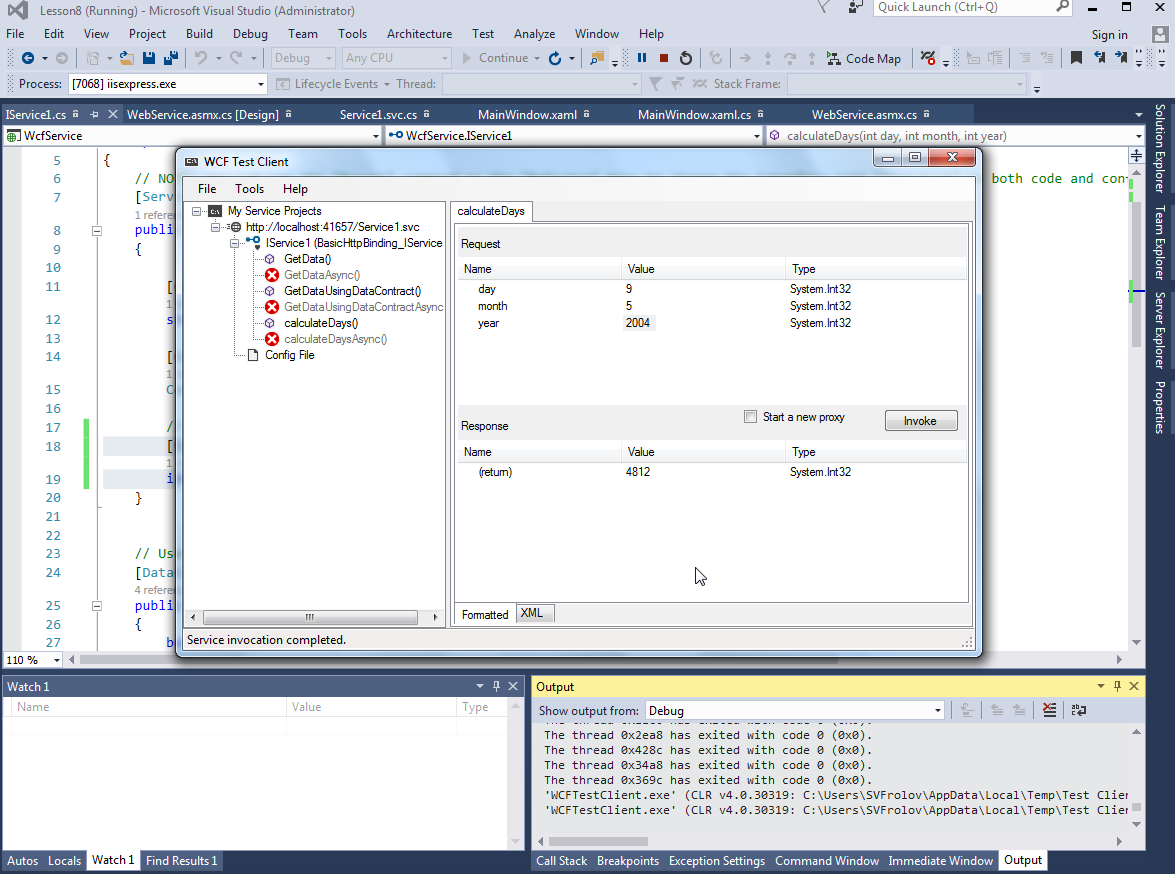
int datetodays = DateTime.Now.Subtract(dt).Days;

return datetodays;

}

…

В дереве проекта выбераем файл Service.svc и нажимаем F5.



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

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

* 1. Создание WPF-приложения потребителя WCF-сервиса

Созадим 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)

{

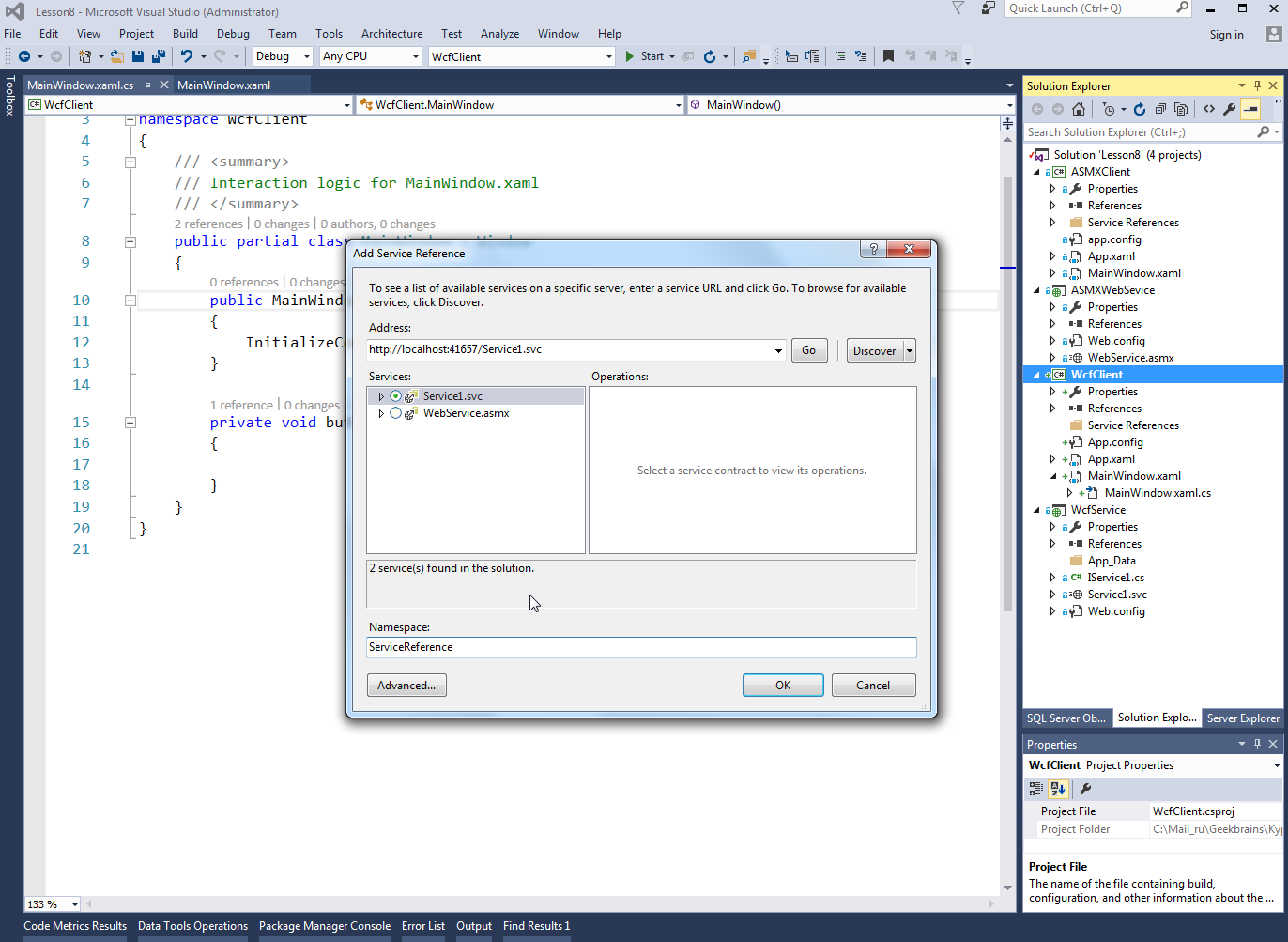
}

}

}

}

Добавим ссылку на WCF-сервис в проект.



1. 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.
  1. 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 основные 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 ресурса. 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. Состояние объекта ресурса хранится на сервере и направляется клиенту в виде представления. Состояние объекта приложения храниться в приложении до передачи ресурсу. Когда состояние объекта приложения отправляется серверу с помощью запросов POST, PUT и DELETE, оно становиться состоянием объекта ресурса.

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

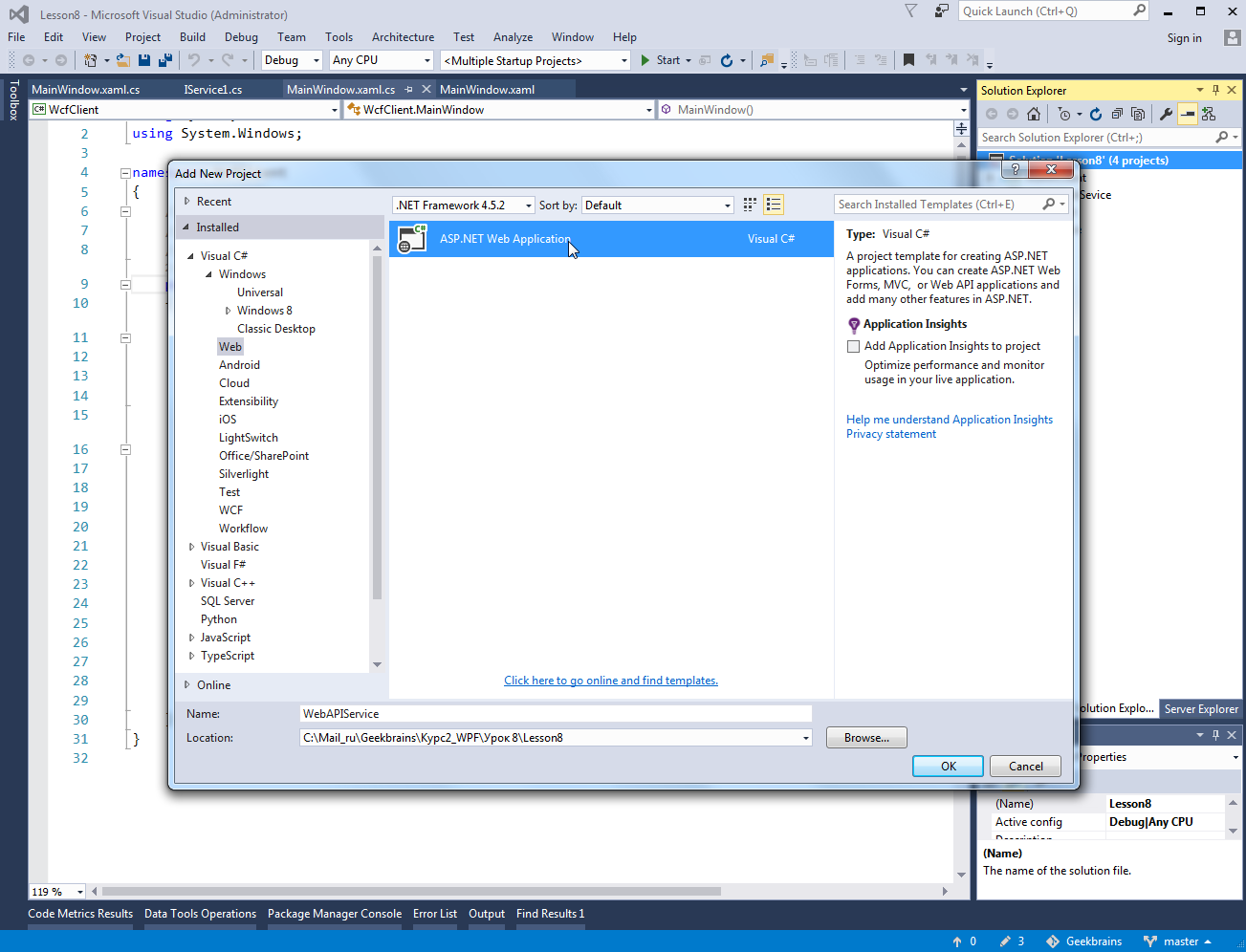
Если клиентское приложение хочет, чтобы учитывалось состояние объекта приложения, необходимо передавать его в каждом вызове. Например, передавать параметры авторизации в каждом вызове.

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

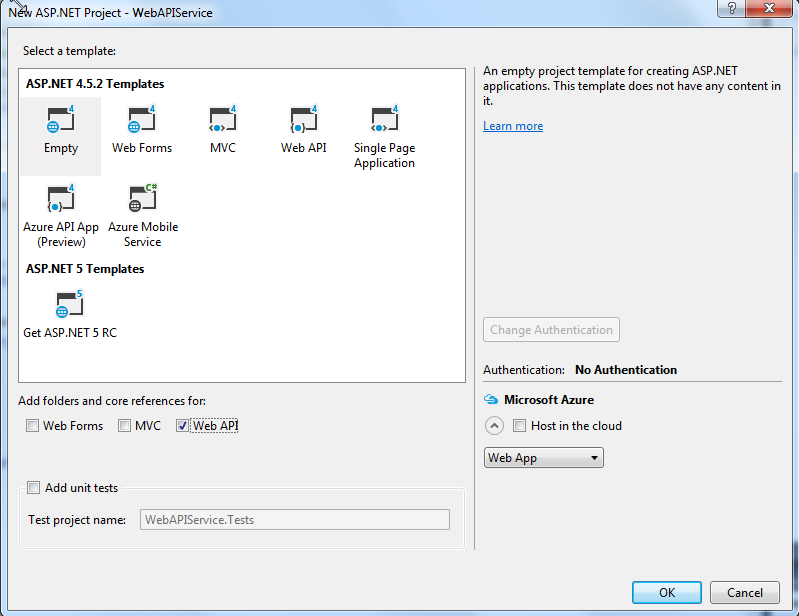
ASP.NET Web API – это фреймворк для построения Web-API сервиса на базе .NET Framework. В следующем примере, ASP.NET Web API используется для создания Web-API сервиса, который возвращает список товаров.

* 1. Создание Web API сервиса

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



Среди шаблонов ASP.NET выбираем шаблон «Empty» и отмечаем чек-бокс «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; }

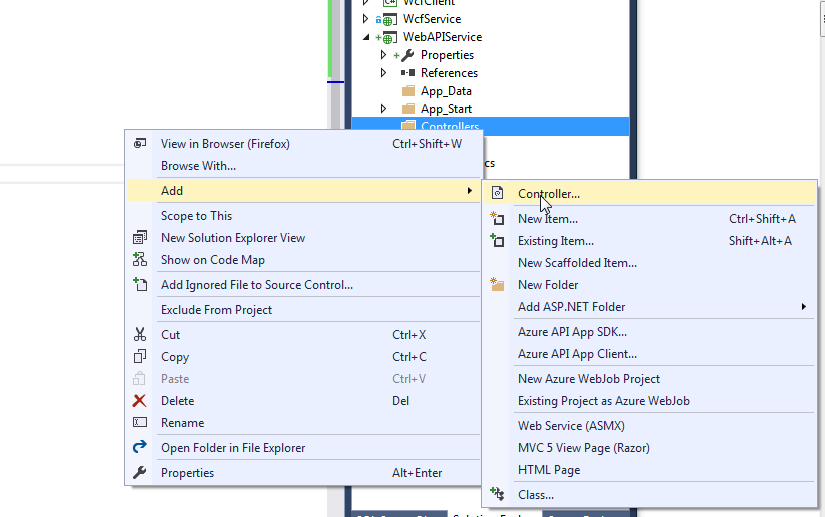
public string Category { get; set; }

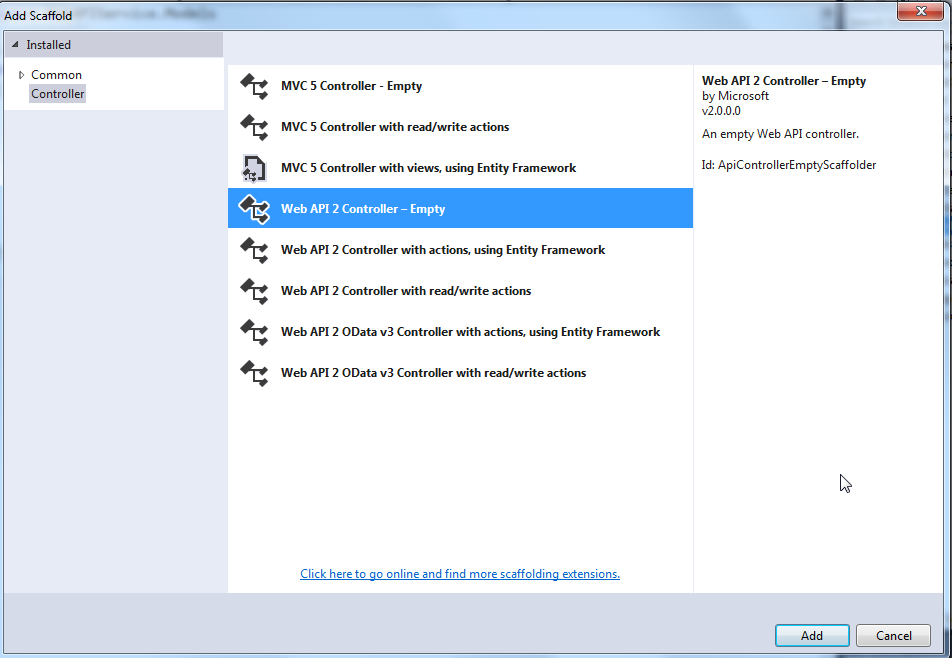
public decimal Price { get; set; }

}

}

В WebAPI, контроллером называется объект, который обрабатывает HTTP запросы. Добавим в проект контроллер, который либо возвращает список товаров, либо товар с заданным ID. В папку Controllers.





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();

}

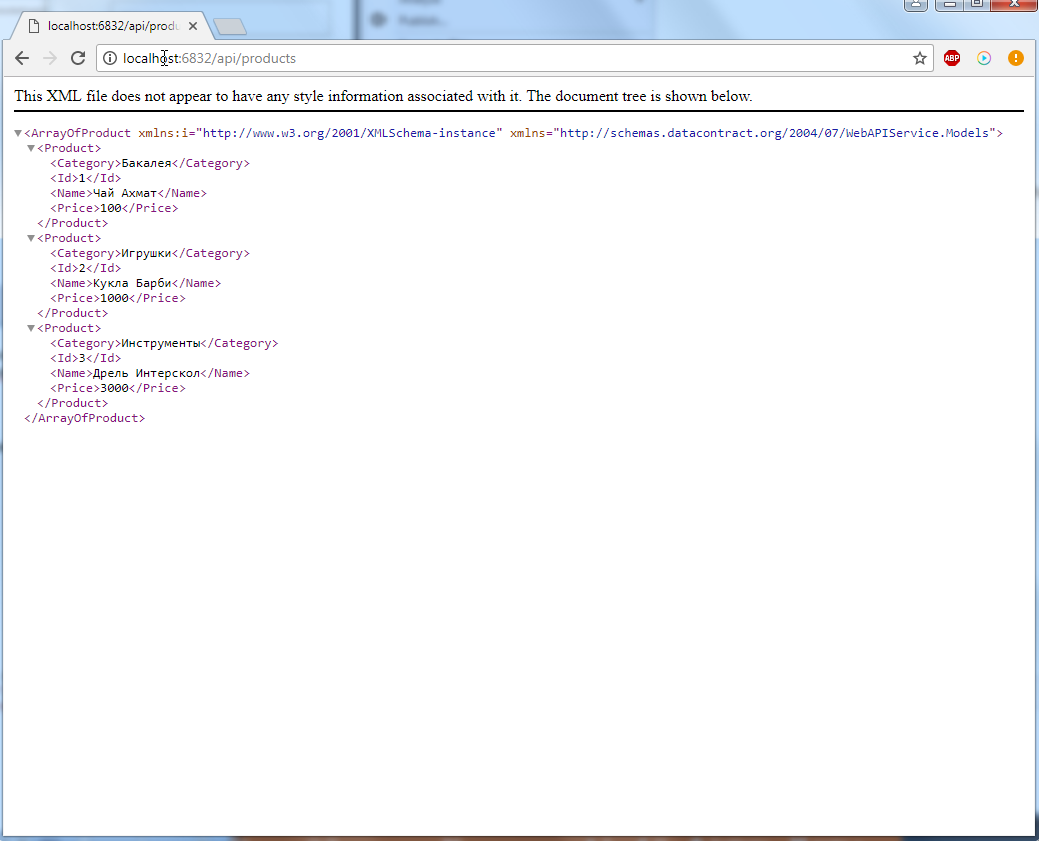
return Ok(product);

}

}

}

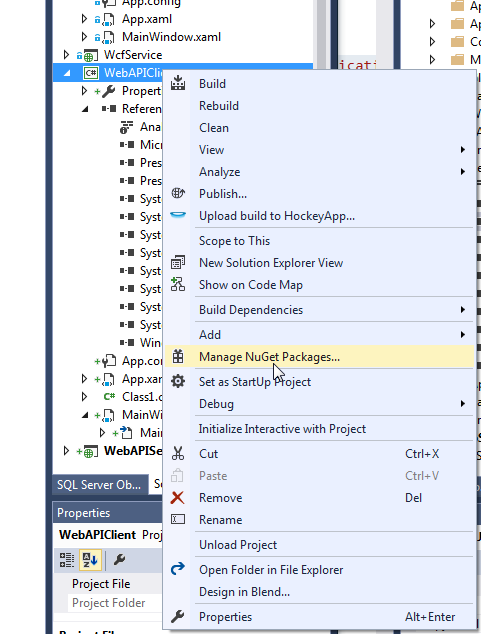
Запустите приложение WebAPIService. В результате откроется браузер и попытается обратиться корневому элементу вашего сервиса. Дополните URL в строке адреса браузера одним из реализованных в сервисе URL . Например, /api/products. После этого обновите страницу в браузере.

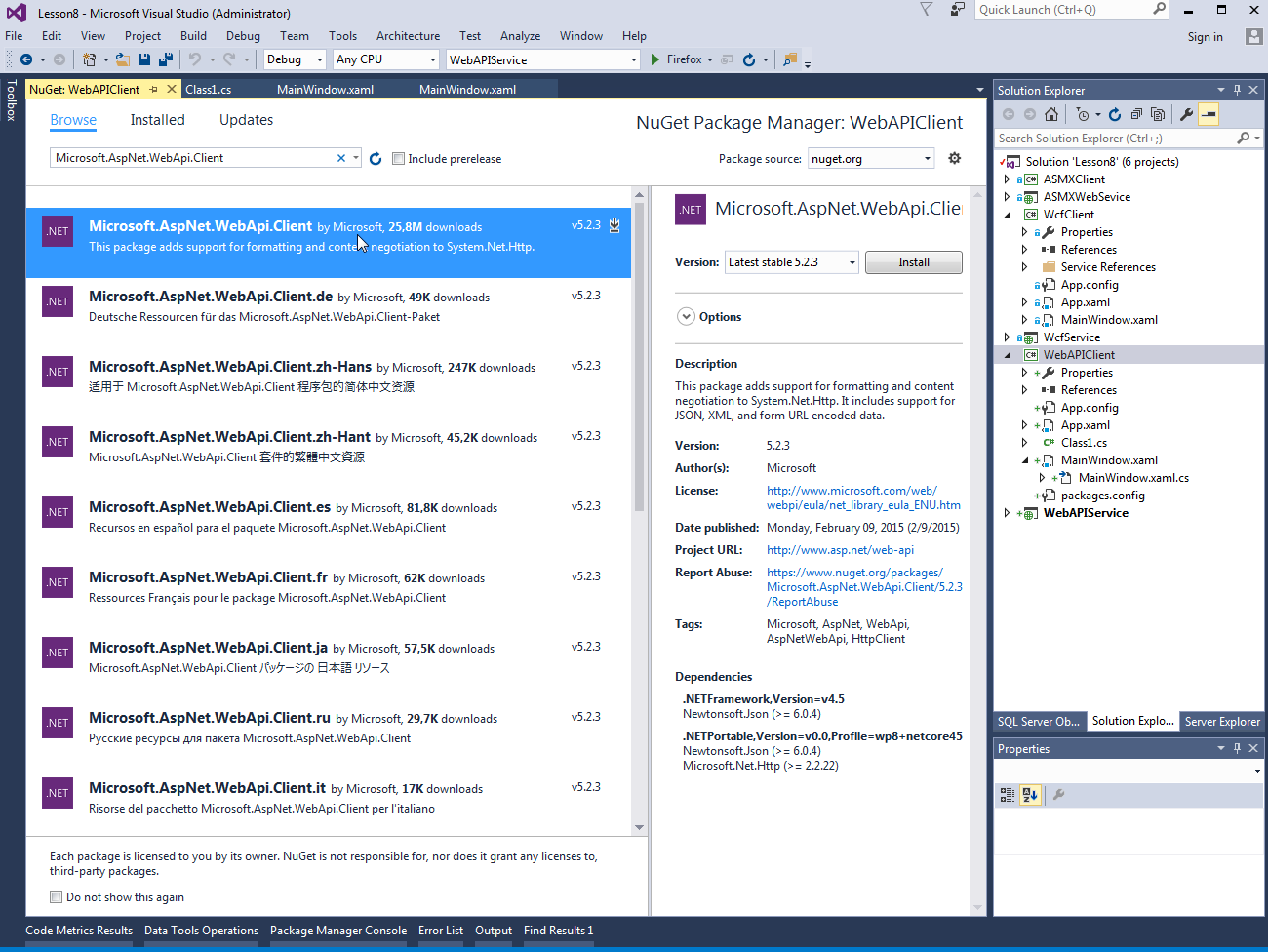


* 1. Создание WPF-приложения потребителя Web API сервиса

Созадим 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/");

1. ДЗ

Измените 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. Создайте таблицы Employee и Department в БД MSSQL Server и заполните списки сущностей начальными данными.  
2. Для списка сотрудников и списка департаментов предусмотрите визуализацию (отображение).

3. Разработайте формы для отображения отдельных элементов списков сотрудников и департаментов.