PRACTICAL 1

Write a program to create a web service to covert temperature from Fahrenheit to Celsius and vice a versa.

Steps:

1. We would be using traditional way of creating web services by separating interface ,implementation and publish part
2. We would first create a service as simple java application and then would create a java web client to consume it.

Service part

Every file is in the same package

Creating Interface

```
package temperature;

import javax.jws.WebService;
@WebService
public interface TemperatureInterface {
    double farenheitToCelsius(double f);
    double celsiusToFarenheit(double c);
}
```

Creating Implementation

```
package temperature;
import javax.jws.WebService;
@WebService(endpointInterface="temperature.TemperatureInterface")
public class TemperatureImpl implements TemperatureInterface {
    @Override
    public double farenheitToCelsius(double f) {
        return ((f-32)*0.5556);
    }

    @Override
    public double celsiusToFarenheit(double c) {
       return ((c*1.8)+32);
    }

}
```

Publishing Web service

```
package temperature;
import javax.xml.ws.Endpoint;
public class Temperature {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:9001/conversion",new TemperatureImpl());
        System.out.println("Service published ....");
    }
}
```

Once you have started running your service make sure it generates the wsdl file by going to address where you published your service and appending ?wsdl to it in browser

localhost:9001/conversion?wsdl

localhost:9001/conversion?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▶<!--...-->
▶<!--...-->
▼<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
 xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing
 xmlns:tns="http://temperature/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.o
 name="TemperatureImplService">
  ▶<types>...</types>
  ▶<message name="celsiusToFarenheit">...</message>
  ▶<message name="celsiusToFarenheitResponse">...</message>
  ▶<message name="farenheitToCelsius">...</message>
  ▶<message name="farenheitToCelsiusResponse">...</message>
  ▶<portType name="TemperatureInterface">...</portType>
  ▶<binding name="TemperatureImplPortBinding" type="tns:TemperatureInterface">...</binding>
  ▶<service name="TemperatureImplService">...</service>
 </definitions>
```

Now we will create the next part that is a web client, so just make sure your service is running and then create a new web application.

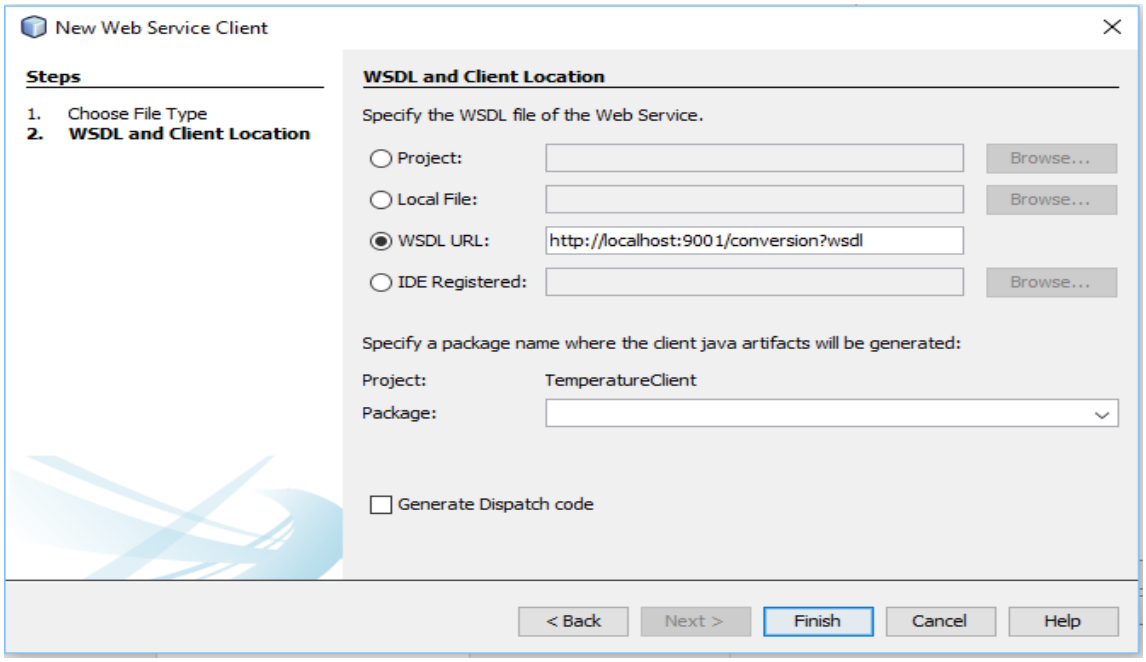
You would require a user interface or html or jsp page to send data to server where you will call the web service created.

Index.html

```html
<html>
    <head>
        <title></title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <form action="TempServlet">
            <input type="text" name="t1">
            <br><br>
            <input type="submit" value="ConvertToFahrenheit" name="b1">
            <input type="submit" value="ConvertToCelsius" name="b1">
        </form>
    </body>
</html>
```

Next you would need a servlet to process your request from html

Before calling your web service methods in servlet you need to add web service client in your project by right clicking on project and adding web service client make sure you provide the wsdl



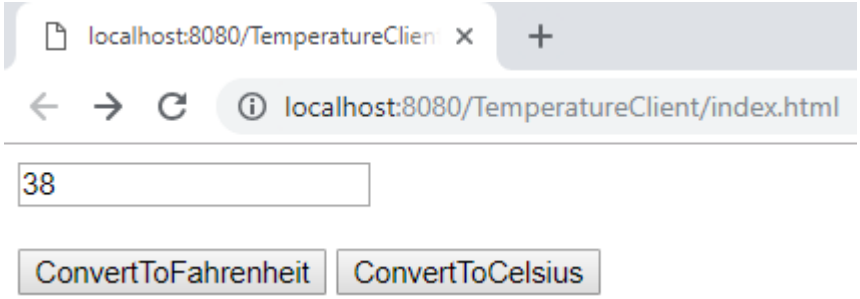
Finally when created your servlet make sure to call web service operation by right clicking on you code part then →insert→call web service operation→select the ones that are specified in your current project.

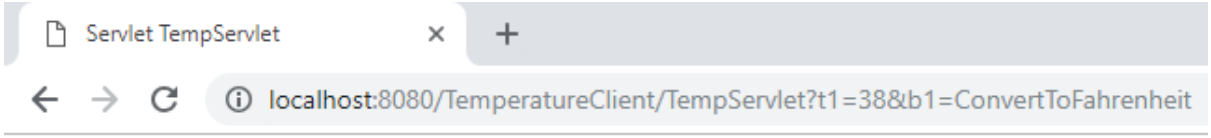
So your final servlet file should be similar to this,

TempServlet.java

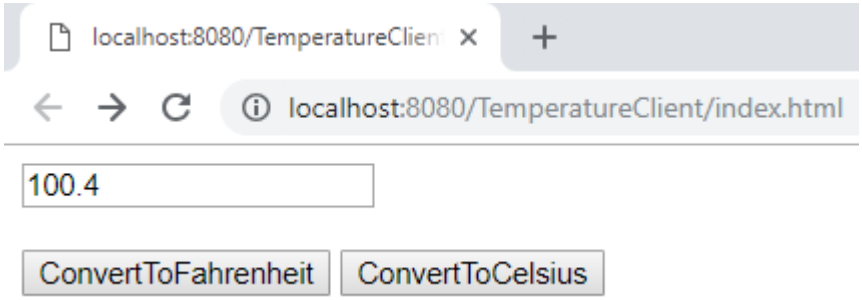
```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet TempServlet</title>");
        out.println("</head>");
        out.println("<body>");
        double value=Double.parseDouble(request.getParameter("t1"));
        if(request.getParameter("b1").equals("ConvertToFahrenheit")){
            out.println("<h1>Temperature in fahrenheit is " +celsiusToFarenheit(value)+ "</h1>");
        }else if(request.getParameter("b1").equals("ConvertToCelsius")){
            out.println("<h1>Temperature in fahrenheit is " +farenheitToCelsius(value)+ "</h1>");
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

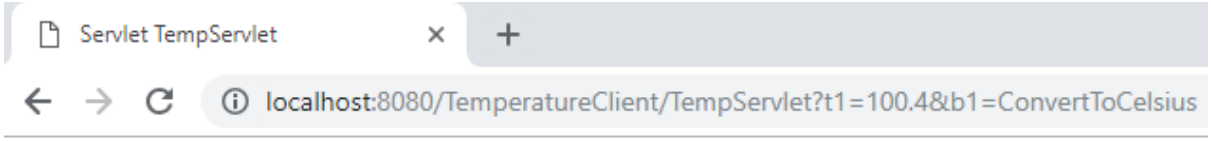
Output:





Temperature in fahrenheit is 100.4





Temperature in fahrenheit is 38.00304

Practical 2

Program to implement web service as in, one-way operation and request-response operation.

This web service will be done in different one than previous we will write and publish our web service in the same class in java application.

In this example we will demonstrate both one way and request-response type operations.

One-way: it is simply web service not designed to respond all you need is annotate the method with @Oneway, such kind of web service can be used to simply pass on messages where the client just indicates its status and does not care about the servers response.

Request-Response: it is the way we usually work with web services.

Here we create two methods , one for simply passing a name the initializes name globally and the other which responses to client with the name he passed previously and with today's date and time.

```
package webservice;
import java.util.Date;
import javax.jws.Oneway;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.Endpoint;
@WebService
public class WebOperations {
    String name;
    @WebMethod
    @Oneway
    public void sayHello(String n){
        name=n;
    }
    @WebMethod
    public String time(){

        return "Hello "+name+" todays time is "+new Date().toString();
    }
    public static void main(String[] args) {
        // TODO code application logic here
    Endpoint.publish("http://localhost:9000/ws",new WebOperations());
    }
}
```

Next create a simple java application client to utilize the service, do the necessary as mentioned previously and now call the web service operations in our main class,

```
package javaclient;
public class JavaClient {
    public static void main(String[] args) {
        JavaClient.sayHello("Mack");
        System.out.println(JavaClient.time());
    }
    private static void sayHello(java.lang.String arg0) {
        webservice.WebOperationsService service = new webservice.WebOperationsService();
        webservice.WebOperations port = service.getWebOperationsPort();
        port.sayHello(arg0);
    }
    private static String time() {
        webservice.WebOperationsService service = new webservice.WebOperationsService();
        webservice.WebOperations port = service.getWebOperationsPort();
        return port.time();
    }
}
```

Output:

```
run:
Hello Mack todays time is Fri Oct 12 22:25:43 IST 2018
BUILD SUCCESSFUL (total time: 5 seconds)
```

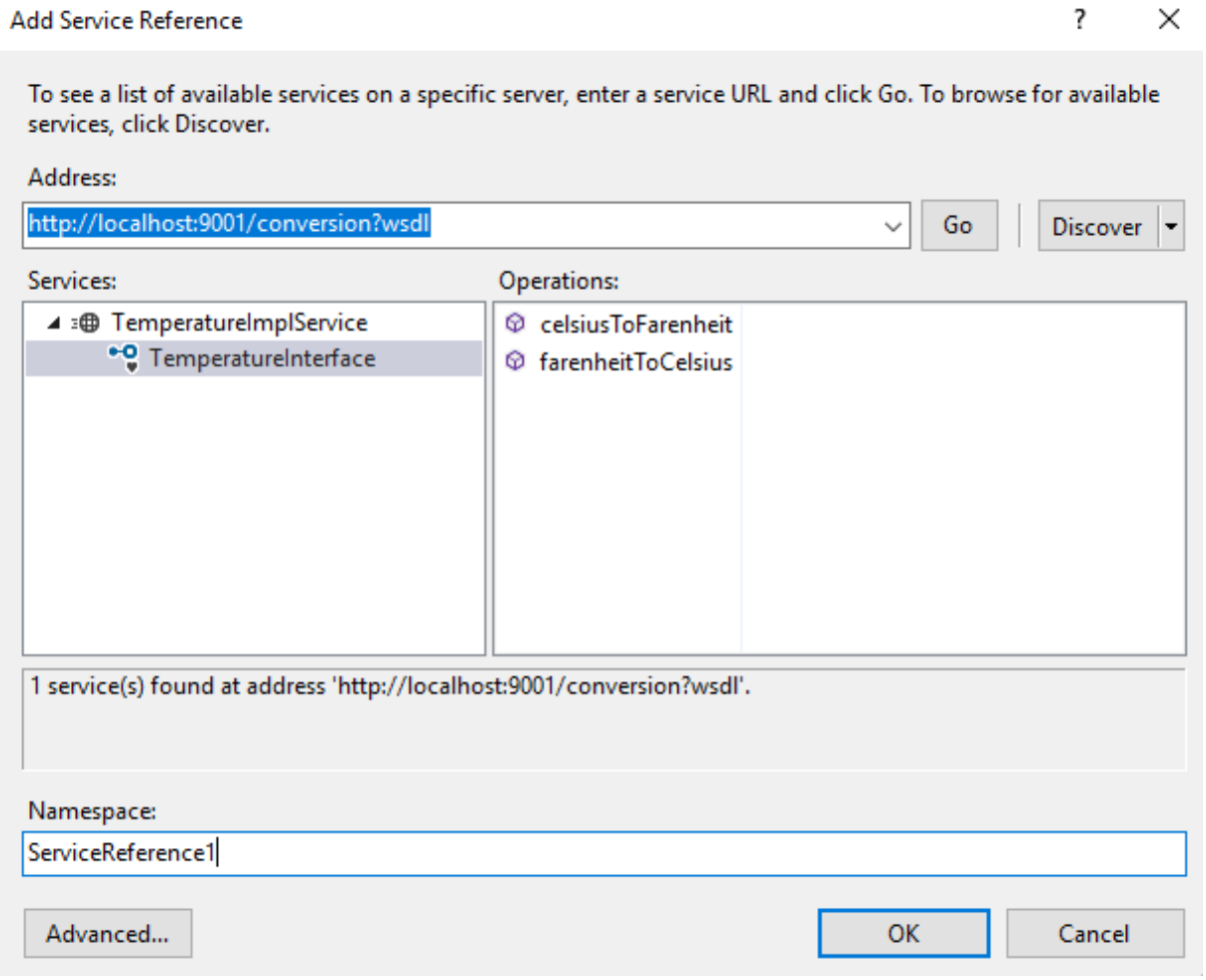
Practical 3

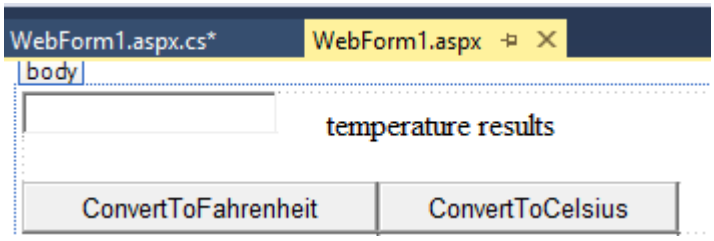
Develop a client that consumes web service in different platform.

For this practical we will utilize the temperature web service we created in practical 1 and use an asp web application client to consume the web service.

So as our web service is ready all we need to do is created asp.net client,

1. Create an empty asp web application
2. Add web reference here you pass in the wsdl generated by web service
3. Now simply design the form according to you work flow logic.

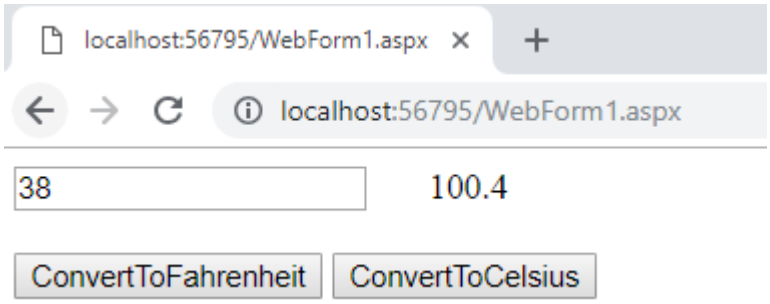




```
WebForm1.aspx.cs   WebForm1.aspx
JavaToAspClient    JavaToAspClient.WebForm1    TextBox1
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.UI;
6   using System.Web.UI.WebControls;
7   using JavaToAspClient.ServiceReference1;
8   namespace JavaToAspClient
9   {
10      public partial class WebForm1 : System.Web.UI.Page
11      {
12          TemperatureInterfaceClient c = new TemperatureInterfaceClient();
13          protected void Page_Load(object sender, EventArgs e)
14          {
15          }
16          protected void Button1_Click(object sender, EventArgs e)
17          {
18              Label1.Text=Convert.ToString(c.celsiusToFarenheit(Convert.ToDouble(TextBox1.Text)));
19          }
20          protected void Button2_Click(object sender, EventArgs e)
21          {
22              Label1.Text = Convert.ToString(c.farenheitToCelsius(Convert.ToDouble(TextBox1.Text)));
23          }
24      }
25  }
```

You can get your web service calling class from the wsdl file here we have added a service reference , you could do same with web reference.

Output:



Practical 4

Creating a “Jax-ws” web service to consume by servlet client.

1.Create a java application and write logic for calculator

2.Create a java web application and prepare it to consume the web service as we done previously with temperature example

Calculator.java

```
package calculator;
import javax.jws.*;
import javax.xml.ws.Endpoint;
@WebService
public class Calculator {
     @WebMethod
     public double add(double a,double b){
        return a+b;
     }
     @WebMethod
     public double sub(double a,double b){
        return a-b;
     }
     @WebMethod
     public double mul(double a,double b){
        return a*b;
     }
     @WebMethod
     public double div(double a,double b){
       return a/b;
     }
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:9999/calculate",new Calculator() );
    }
}
```

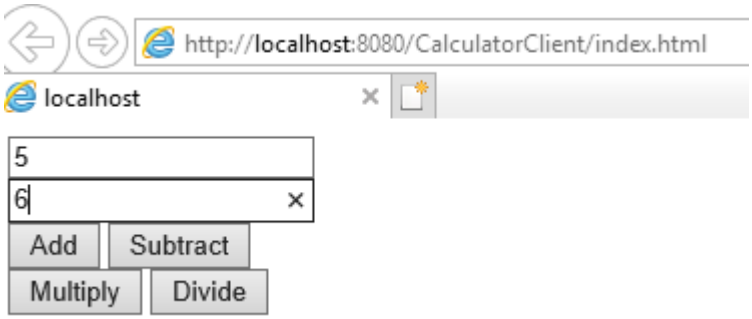
Index.html

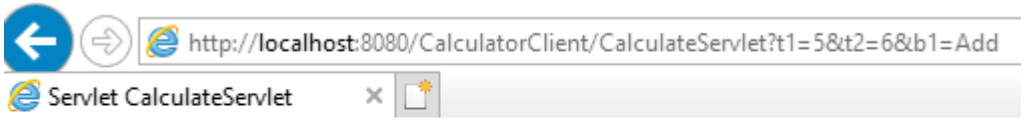
```html
<html>
    <head>
        <title></title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <form action="CalculateServlet">
            <input type="text" name="t1">
            <br>
            <input type="text" name="t2">
            <br>
            <input type="submit" value="Add" name="b1">
            <input type="submit" value="Subtract" name="b1">
            <br>
            <input type="submit" value="Multiply" name="b1">
            <input type="submit" value="Divide" name="b1">

        </form>
    </body>
</html>
```

CalculateServlet.java

```java
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet CalculateServlet</title>");
        out.println("</head>");
        out.println("<body>");
        double n1=Double.parseDouble(request.getParameter("t1"));
        double n2=Double.parseDouble(request.getParameter("t2"));
        String value=request.getParameter("b1");
        if(value.equals("Add")){
        out.println("<h1>Addition is "+add(n1,n2)+"</h1>");
        }else if(value.equals("Subtract")){
        out.println("<h1>Subtraction is "+sub(n1,n2)+"</h1>");
        }else if(value.equals("Multiply")){
        out.println("<h1>Multiplication is "+mul(n1,n2)+"</h1>");
        }else if(value.equals("Divide")){
        out.println("<h1>Division is "+div(n1,n2)+"</h1>");
        }
        out.println("</body>");
        out.println("</html>");
    }}
```

Output:





Addition is 11.0

Practical 5

Develop a RESTful Api that allows user to perform crud operations

Steps:

1.First we need to setup the java database

2.Then we would need to setup an maven application of jersey archetype to develop Restful Api

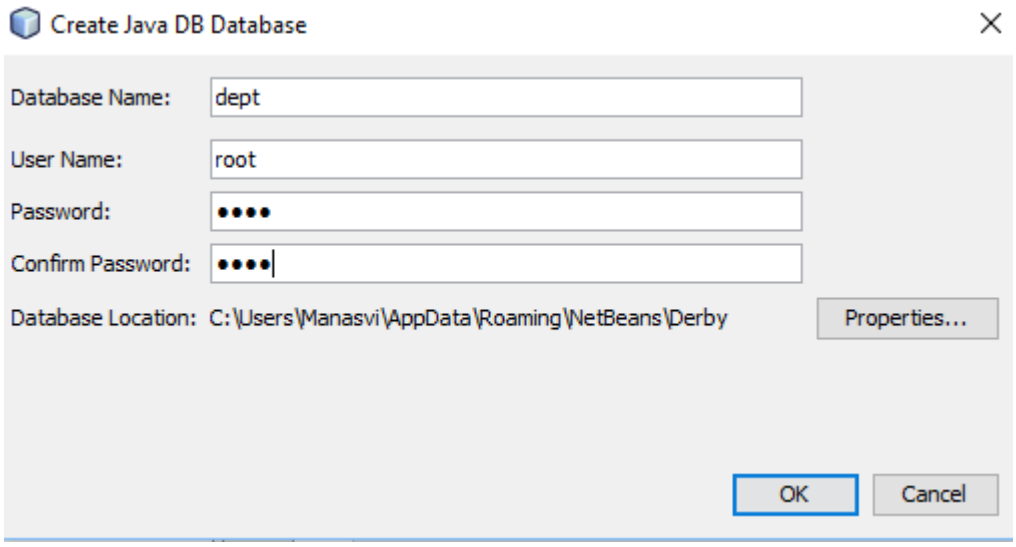
3.At Last we would need a client that helps us perform crud operation (Create,Read,Update,Delete) using get, put, post ,delete http methods.

We will be using postman which is designed to test Rest Api.

Creation of database

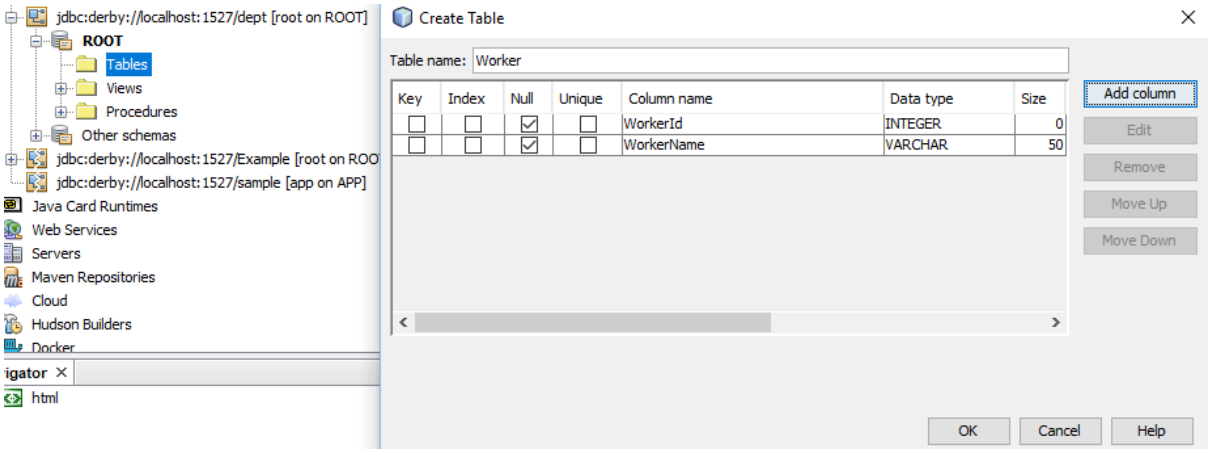
We will use java database for our purpose, so from services tab, right click on your javadb→start server

After starting the server once again do the same as above but this time select create database as we would be doing so.



After creation of your database right click on it and then click on connect.

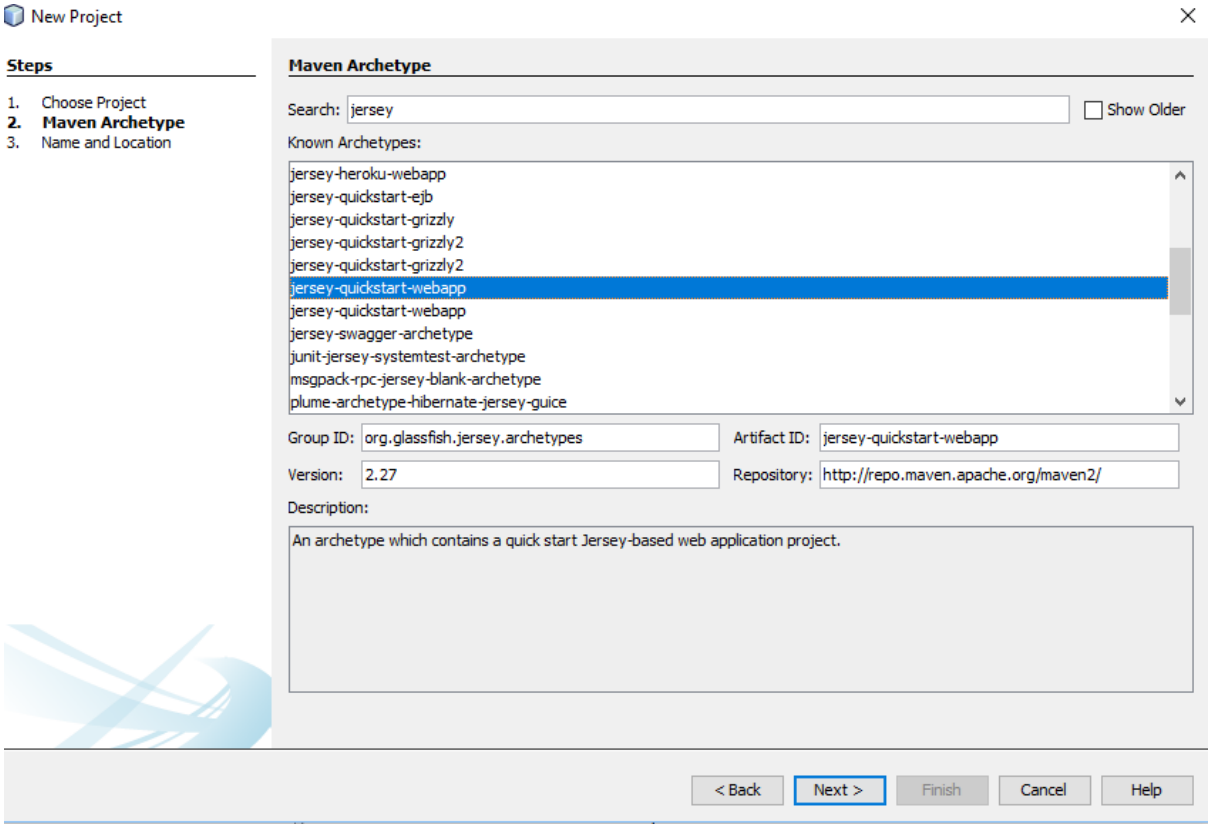
Once you connected to database, create a table name worker, and insert values through queries.



Once we have successfully develop our database now we need to design rest api.

So create a maven application→project from archetype

Type jersey in search bar, the give project name and finish it.



Next we need to create three java files in the package where my resource file is,

Worker.java, WorkerResource.java, WorkerService.java

1. Worker.java (representation of our data)

```java
package com.mycompany.worker;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Worker {
    int id;
    String name;
    public Worker(){}

    public Worker(int id,String name){
        this.id=id;
        this.name=name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

2.WorkerService.java(the way we connect our rest service to database)

```java
package com.mycompany.worker;
import java.util.*;
import java.sql.*;
public class WorkerService {
    List<Worker> wlist=new ArrayList<>();
    Connection conn;
    PreparedStatement pst;
    public void connect() throws SQLException{
        conn=DriverManager.getConnection("jdbc:derby://localhost:1527/dept","root","root");
    }
    public List<Worker> getFromDatabase() throws SQLException{
        connect() ;
        pst=conn.prepareStatement("Select * from worker");
        ResultSet rst=pst.executeQuery();
        while(rst.next()){
            wlist.add(new Worker(rst.getInt(1),rst.getString(2)));
        }
    return wlist;
    }

    public void updateInDatabase(int id,Worker wk) throws SQLException{
        connect() ;
        pst=conn.prepareStatement("Update worker set workername=? where workerid=?");
        pst.setString(1, wk.name);
        pst.setInt(2, id);
        pst.executeUpdate();
    }
    public void createInDatabase(int id,Worker wk) throws SQLException{
        connect();
        pst=conn.prepareStatement("insert into worker values(?,?)");
        pst.setInt(1, id);
        pst.setString(2, wk.name);
        pst.executeUpdate();
    }

    public void deleteInDatabase(int id) throws SQLException{
        connect();
        pst=conn.prepareStatement("delete from worker where workerid=?");
        pst.setInt(1, id);
        pst.executeUpdate();
    }
}
```

3.WorkerResource.java(representation of restful api)

```java
package com.mycompany.worker;
import java.sql.SQLException;
import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
@Path("worker")
public class WorkerResource {
    WorkerService ws=new WorkerService();
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Worker> getWorkers() throws SQLException{
        return ws.getFromDatabase();
    }
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @Path("{id}")
    public void update(@PathParam("id")int id,Worker wk) throws SQLException{
        ws.updateInDatabase(id, wk);
    }
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Path("{id}")
    public void create(@PathParam("id")int id,Worker wk) throws SQLException{
        ws.createInDatabase(id, wk);
    }
    @DELETE
    @Path("{id}")
    public void delete(@PathParam("id") int id) throws SQLException{
        ws.deleteInDatabase(id);
    }}
```

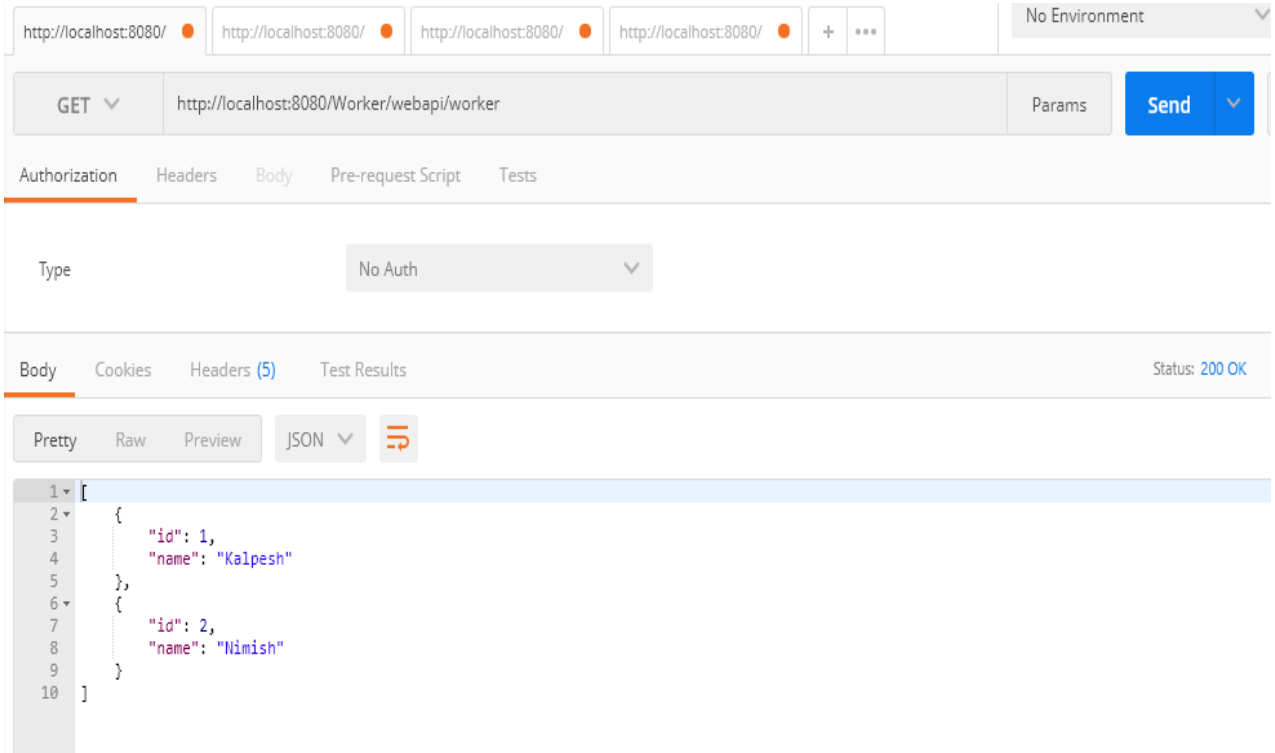
One last thing before running make sure you uncomment json-binding part as we will be using json for crud operation to be performed.

It's in the pom.xml file,

```xml
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</artifactId>
</dependency>
```

Using Postman client

GET Request



POST Request

POST http://localhost:8080/Worker/webapi/worker/3 Params Send

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Content-Type	application/json	
New key	Value	Description

Response

POST http://localhost:8080/Worker/webapi/worker/3 Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "name": "Rekha"
}
```

Body Cookies Headers (3) Test Results Status: 204 No Content

Pretty Raw Preview Text

Then get request again to check whether data is created at id = 3

http://localhost:8080/ http://localhost:8080/ http://localhost:8080/ http://localhost:8080/ No Environment

GET http://localhost:8080/Worker/webapi/worker Params Send

Authorization Headers Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (5) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
[
    {
        "id": 1,
        "name": "Kalpesh"
    },
    {
        "id": 2,
        "name": "Nimish"
    },
    {
        "id": 3,
        "name": "Rekha"
    }
]
```

PUT Request

http://localhost:8080/ | http://localhost:8080/ | http://localhost:8080/ | http://localhost:8080/

PUT http://localhost:8080/Worker/webapi/worker/2 Params Send

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Content-Type	application/json	
New key	Value	Description

http://localhost:8080/ | http://localhost:8080/ | http://localhost:8080/ | http://localhost:8080/ No Environment

PUT http://localhost:8080/Worker/webapi/worker/2 Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "name": "Shanti"
}
```

Body Cookies Headers (3) Test Results Status: 204 No Content

Pretty Raw Preview Text

Then get request again to check whether data is updated at id = 2

http://localhost:8080/ | http://localhost:8080/ | http://localhost:8080/ | http://localhost:8080/ No Environment

GET http://localhost:8080/Worker/webapi/worker Params Send

Authorization Headers Body Pre-request Script Tests

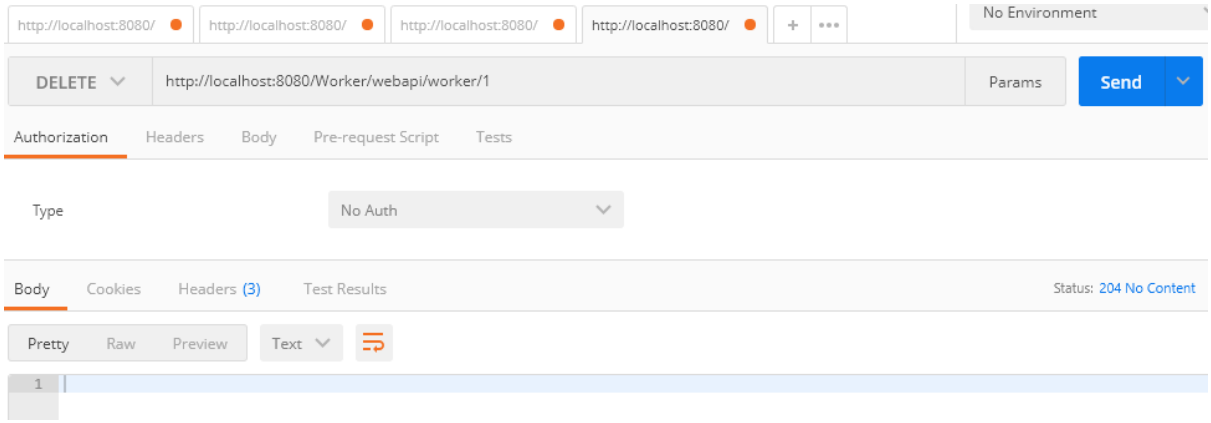
Type No Auth

Body Cookies Headers (5) Test Results Status: 200 OK

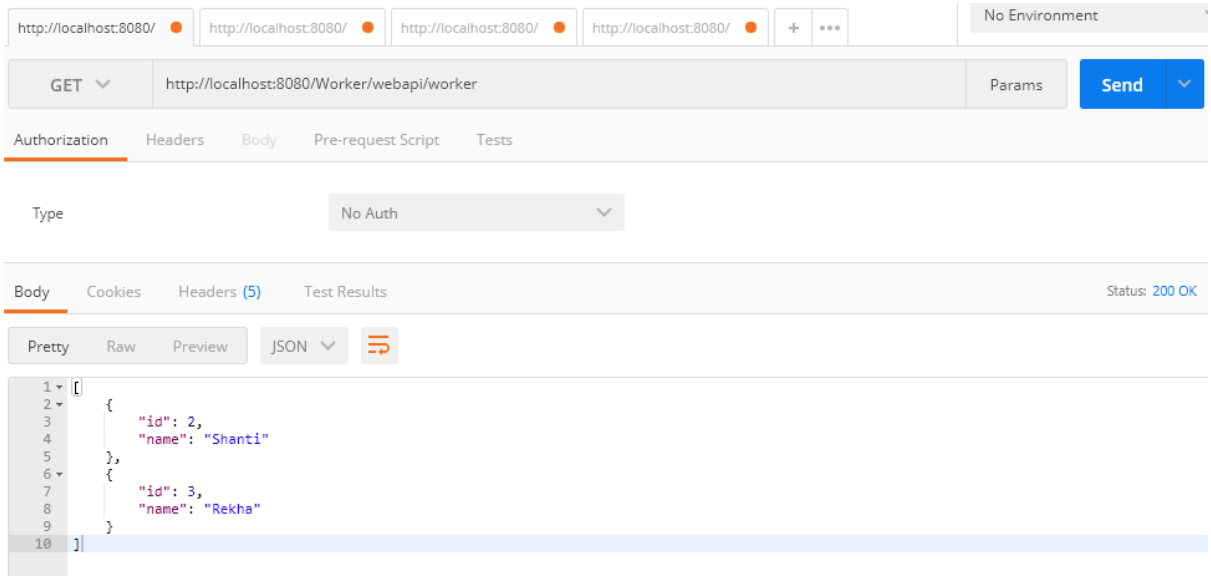
Pretty Raw Preview JSON

```
[
    {
        "id": 1,
        "name": "Kalpesh"
    },
    {
        "id": 2,
        "name": "Shanti"
    },
    {
        "id": 3,
        "name": "Rekha"
    }
]
```

DELETE request



Then get request again to check whether data is deleted at id = 1



Practical 6

Create html client or restful web service , which take json response and represents it in tabular format to user.

Steps:

1. We will require restful api which we develop previously make sure that is running.
2. Next we would need a java web client , so like previous instances create a java web app ad add servlet file to it.
3. Finally we would need a json parser to parse the response send by our rest service, so we will use gson parser(Google's json parser library) to do so.

Develop a Rest api as mentioned in previous practical , then create a new java web application for our client.

Index.html

```
<html>
    <head>
        <title></title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
         <form action="JsonParse">
        <h1>How to display json data in tabular format </h1>
        <input type="Submit" value="GetJson"/>
        </form>
    </body>
</html>
```

JsonParse.java(Make sure to add jar in your library folder)

```java
public class JsonParse extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            URL url = new URL("http://localhost:8080/Worker/webapi/worker");
            URLConnection req = url.openConnection();
            req.connect();
            JsonParser jp = new JsonParser();
            JsonElement root = jp.parse(new InputStreamReader((InputStream) req.getContent()));
            JsonArray rootobj = root.getAsJsonArray();
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet JsonParse</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<table border=2>"
                    + "<tr>"
                    + "<td>ID</td>"
                    + "<td>NAME</td>"
                    + "</tr>");
```

```java
            for(Object o:rootobj){
                out.println("<tr>");
                JsonObject ex=(JsonObject)o;
                out.println("<td>"+ex.get("id")+"</td>");
                out.println("<td>"+ex.get("name")+"</td></tr>");
            }
            //remember to close table
            out.println("</table>");
            out.println("</body>");
            out.println("</html>");

        }
    }
```

Output:

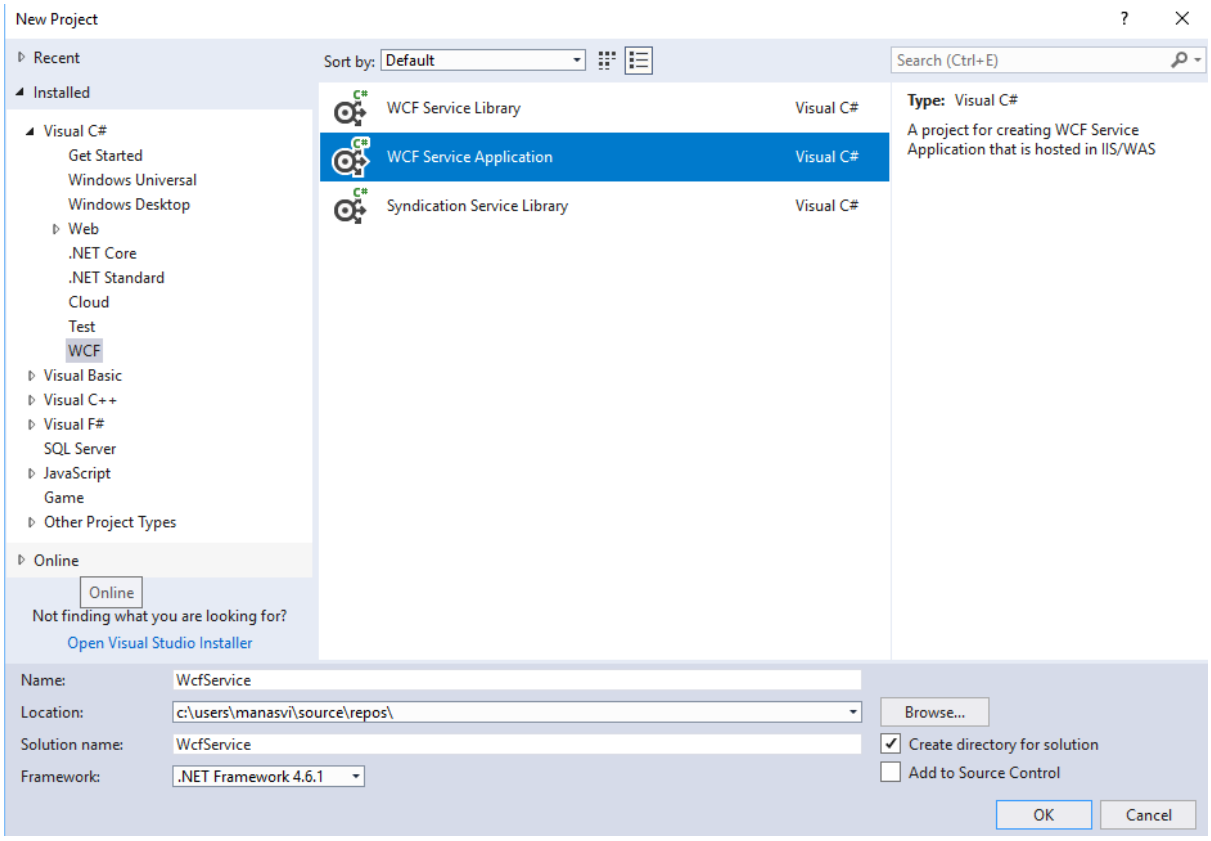
How to display json data in tabular format

GetJson

ID	NAME
2	"Shanti"
3	"Rekha"

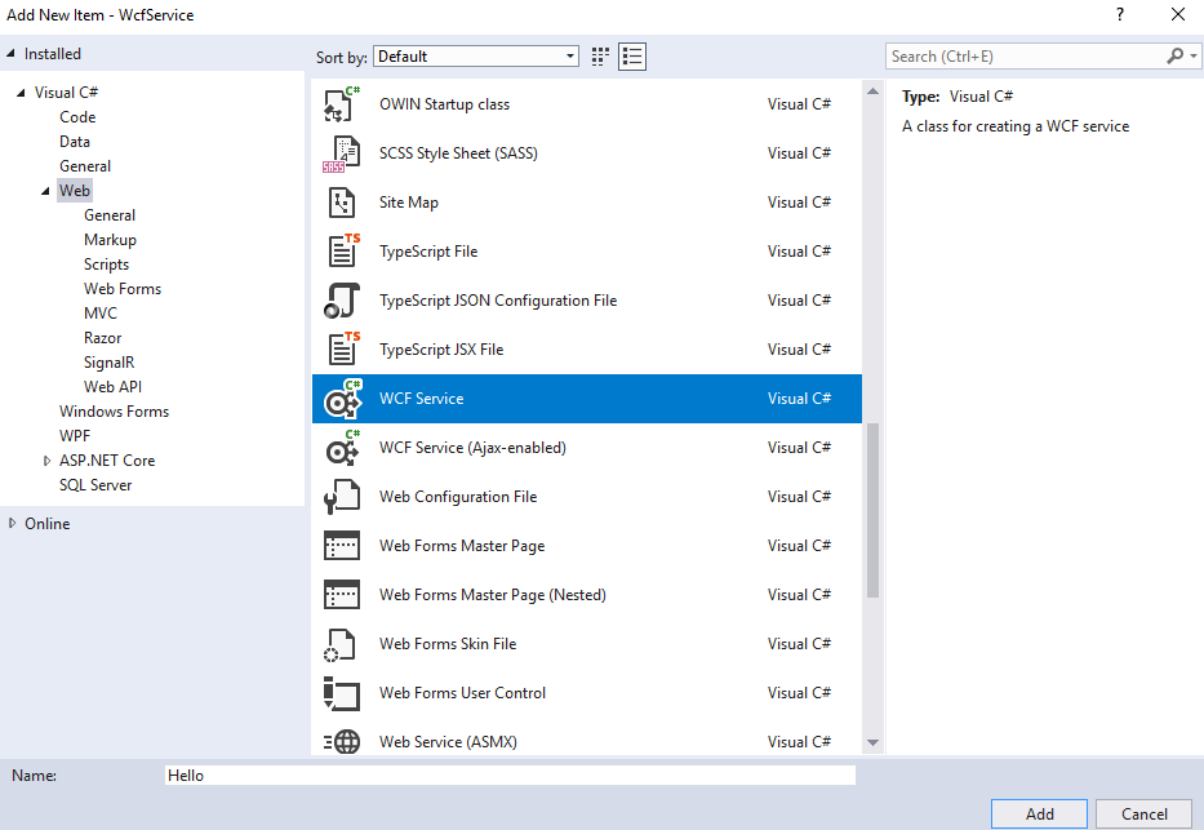
Practical 7

Create a simple WCF service and its client.



First delete existing class and interface then add new wcf service

Right click on project→add→new item→wcf service (name it Hello)



Interface (IHello , it is generated automatically just do required changes)

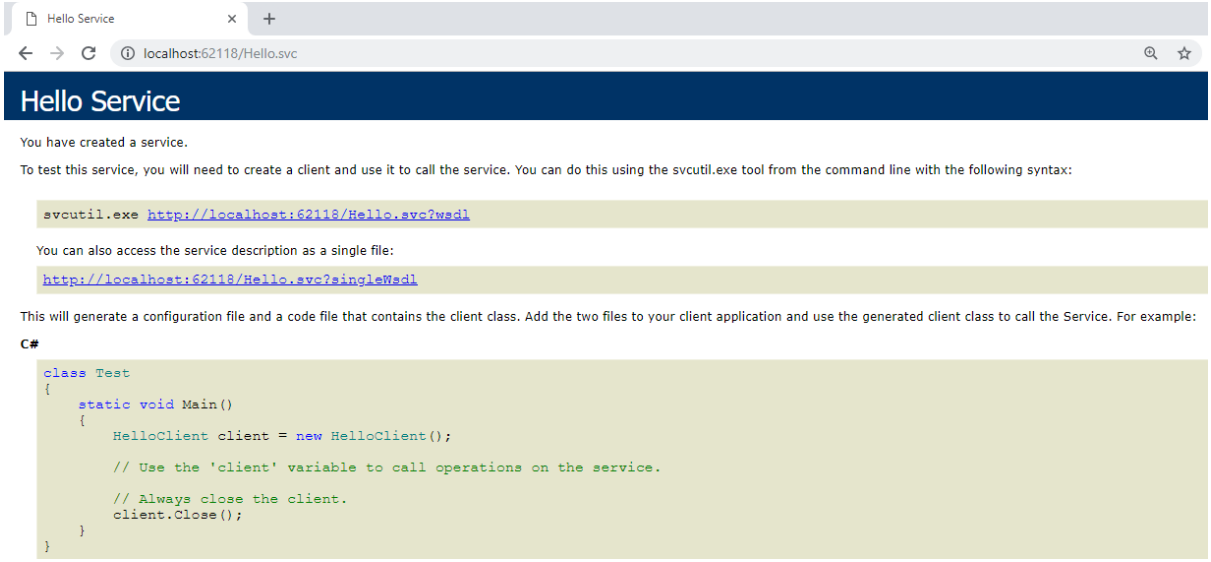
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfService
{
    [ServiceContract]
    public interface IHello
    {
        [OperationContract]
        string sayHello(string name);
    }
}
```

Class (Hello , this is the class you just created)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace WcfService
{
    public class Hello : IHello
    {
        public string sayHello(string name)
        {
            return "Hello " + name + " from Wcf Service";
        }
    }
}
```

Finally right click on project→set as startup project and then run it.

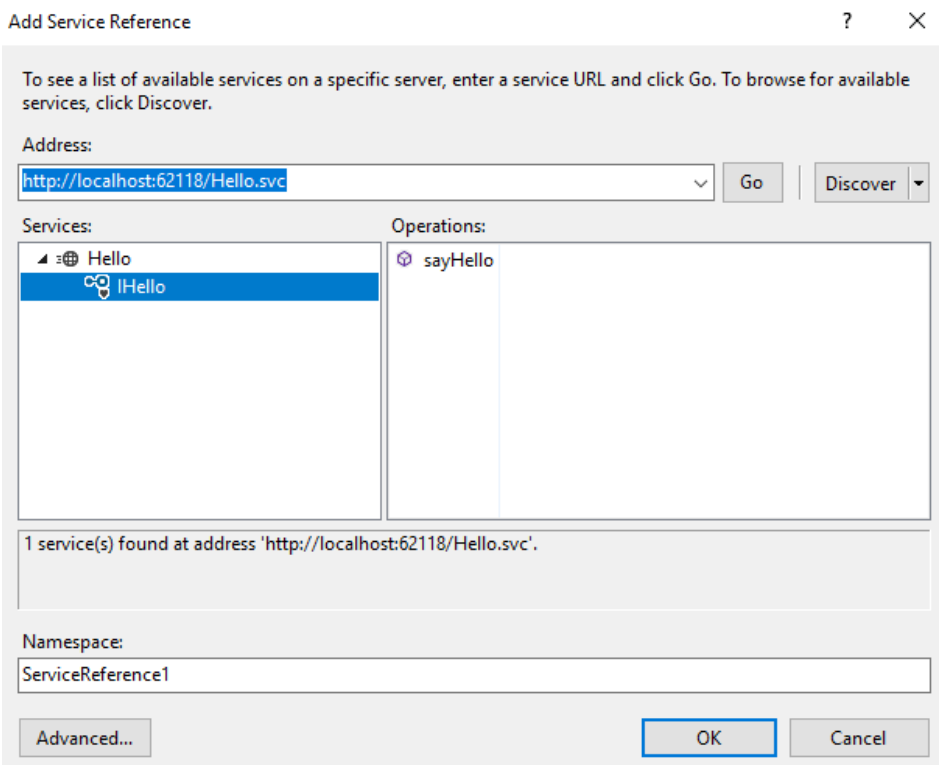
Select Hello.svc



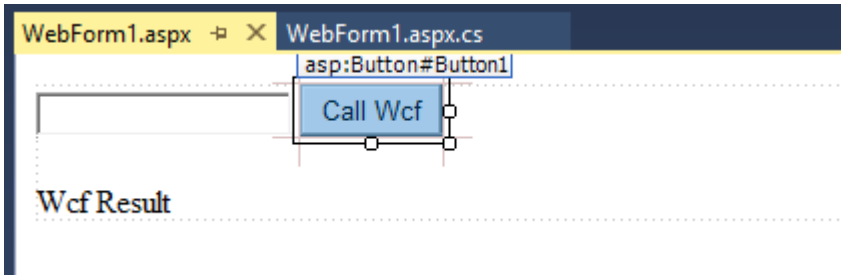
Now all we need is client to consume this service , we'll be using asp web application(keep the wcf service running)

Now first add reference of our wcf service in the project by right clicking on reference→add service reference

Paste the url from the browser and then click on go.



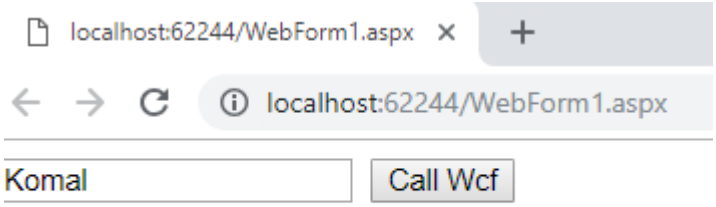
Design of web form



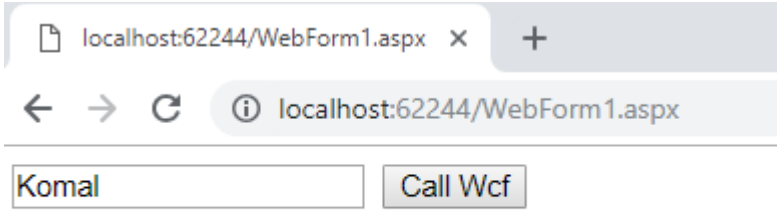
Calling our web service from cs file

```
using AspWcfClient.ServiceReference1;
namespace AspWcfClient
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            HelloClient c = new HelloClient();
            Label1.Text =c.sayHello(TextBox1.Text);
        }
    }
}
```

Output:



Wcf Result



Hello Komal from Wcf Service

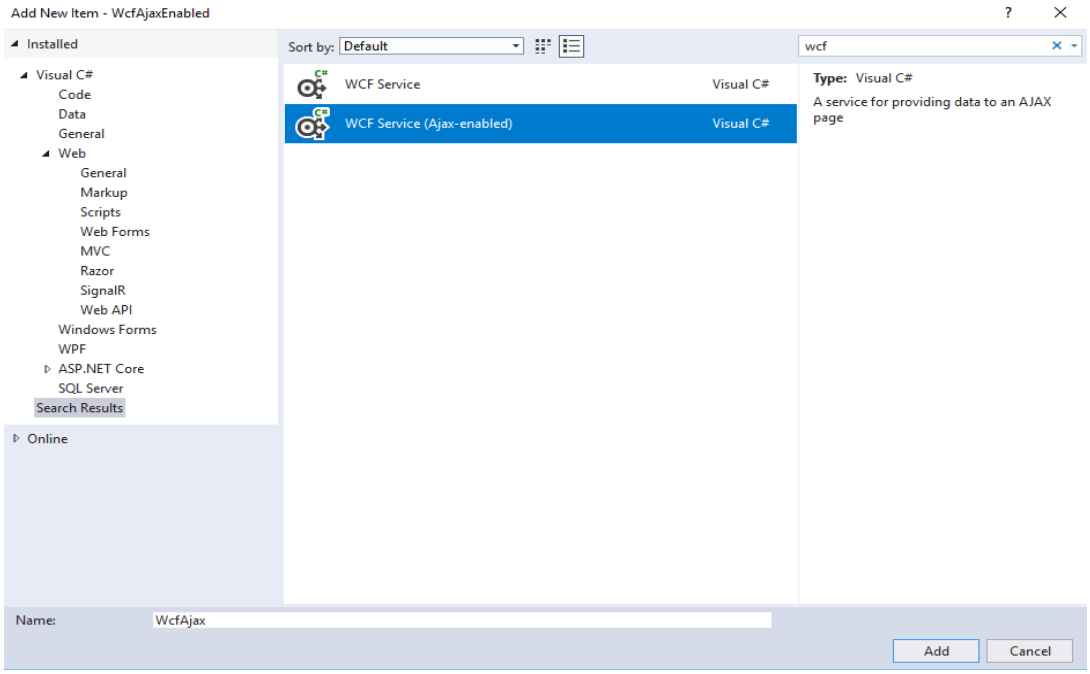
Practical 8

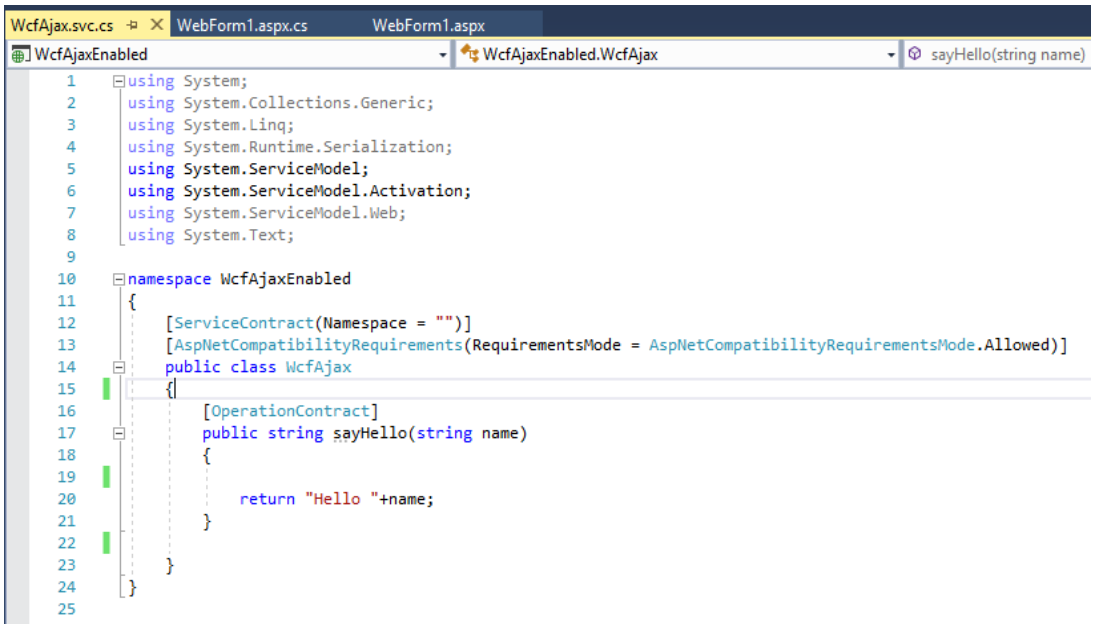
Create a Wcf Service which is Ajax enabled.

Steps:

1. Create a asp web client
2. Create wcf ajax enabled service

When we create empty asp web application , don't do anything for now , we will first define our Wcf ajax enabled web service and then do necessary as previously mentioned.

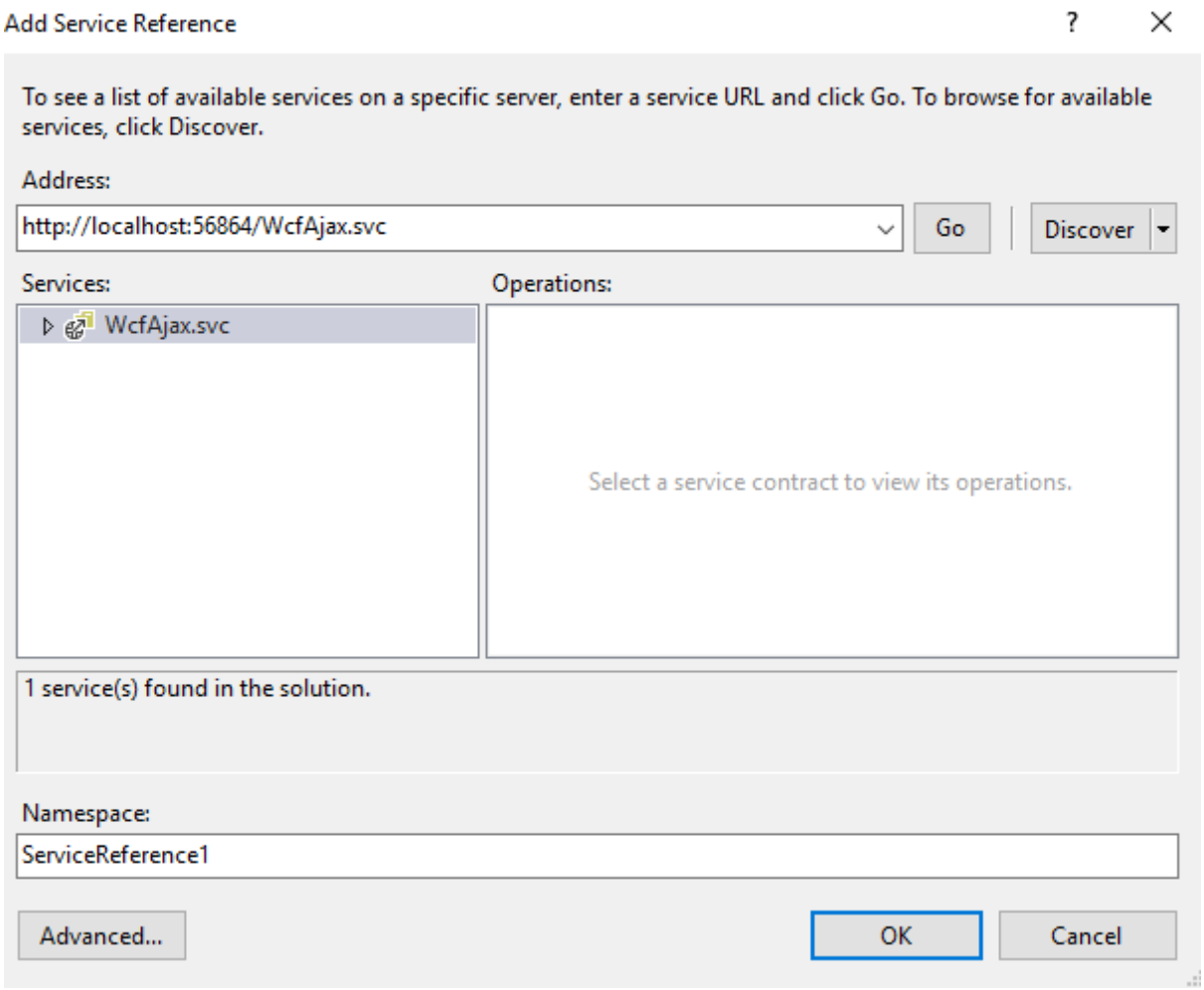




After creating the ajax service now design the client as below:

```
WebForm1.aspx
1   <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WcfAjaxEnabled.WebForm1" %>
2
3   <!DOCTYPE html>
4
5   <html xmlns="http://www.w3.org/1999/xhtml">
6   <head runat="server">
7       <title></title>
8   </head>
9   <body>
10      <form id="form1" runat="server">
11          <div>
12              <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
13              <br /><br />
14              <asp:ScriptManager ID="ScriptManager1" runat="server">
15              </asp:ScriptManager>
16
17              <asp:UpdatePanel ID="UpdatePanel1" runat="server">
18                  <ContentTemplate>
19                      <asp:Timer ID="Timer1" runat="server" Interval="5000" OnTick="Timer1_Tick"></asp:Timer>
20              <br /><br />
21                      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
22                  </ContentTemplate>
23
24              </asp:UpdatePanel>
25
26          </div>
27      </form>
28  </body>
29  </html>
```

Then add service reference same as previous, this time click on discover you will see the service as it is defined locally so you won't need url.

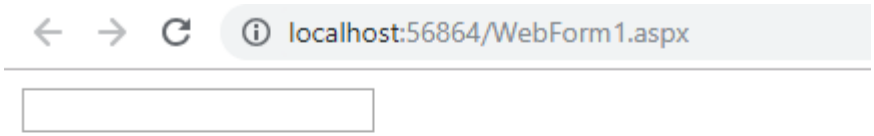


WebForm1.aspx.cs* | WebForm1.aspx

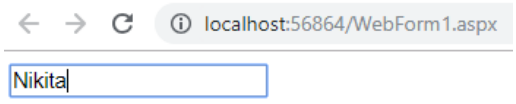
WcfAjaxEnabled | WcfAjaxEnabled.WebForm1

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.UI;
6   using System.Web.UI.WebControls;
7   using WcfAjaxEnabled;
8   namespace WcfAjaxEnabled
9   {
10      public partial class WebForm1 : System.Web.UI.Page
11      {
12          protected void Page_Load(object sender, EventArgs e)
13          {
14
15          }
16
17          protected void Timer1_Tick(object sender, EventArgs e)
18          {
19
20              WcfAjax w = new WcfAjax();
21              Label1.Text = w.sayHello(TextBox1.Text);
22
23          }
24      }
25  }
```

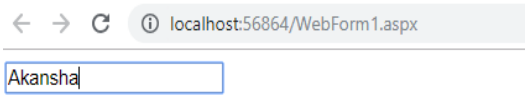
Output:



Label



Hello Nikita



Hello Akansha

Practical 9

Demonstrate the binding attribute of wcf service.

Steps:

1.We will write an C# assembly or library of wcf service

2.We will need to create a C# Console app for hosting wcf service.

3.Next we'll need two clients one for http and other for tcp,

so we will create one C# Asp web application and another

C# Windows form application.

(Run Visual studio with admin privilages)

First create new project→Class Library (this is where we will define our wcf service)

After creating Class Library delete the existing class and then add new Wcf Service.

Functionality of this service is as follows:

IHelloService.cs | HelloService.cs

HelloServiceWcf | HelloServic

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Runtime.Serialization;
5   using System.ServiceModel;
6   using System.Text;
7
8   namespace HelloServiceWcf
9   {
10      [ServiceContract]
11      public interface IHelloService
12      {
13          [OperationContract]
14          string getMessage(string name);
15      }
16  }
```

```
IHelloService.cs    HelloService.cs*
HelloServiceWcf                                  HelloServiceWcf.HelloService
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Runtime.Serialization;
5   using System.ServiceModel;
6   using System.Text;
7
8   namespace HelloServiceWcf
9   {
10      public class HelloService : IHelloService
11      {
12          public string getMessage(string name)
13          {
14              return "Hello " + name;
15          }
16      }
17  }
```

Once created the service in the same solution add new console application.

Here you need to add two references

Right Click on reference→Add reference→ .Net →System.ServiceModel

Right Click on reference→Add reference→ Projects→HelloServiceWcf(Your class library project name)

Then edit Program.cs as follows:

```
Program.cs
HelloHost                                        HelloHost.Program
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.ServiceModel;
6   namespace HelloHost
7   {
8       class Program
9       {
10          static void Main()
11          {
12              using (System.ServiceModel.ServiceHost host = new
13                  System.ServiceModel.ServiceHost(typeof(HelloServiceWcf.HelloService)))
14              {
15                  host.Open();
16                  Console.WriteLine("Wcf started running....");
17                  Console.ReadLine();
18              }
19
20          }
21      }
22  }
```

Further add Application Configuration file to the console project and edit it as follows:

App.config Program.cs

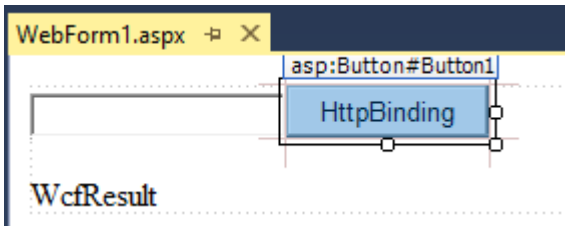
```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="mexBehaviour">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="HelloServiceWcf.HelloService" behaviorConfiguration="mexBehaviour">
        <endpoint address="HelloService" binding="basicHttpBinding" contract="HelloServiceWcf.IHelloService">
        </endpoint>
        <endpoint address="HelloService" binding="netTcpBinding" contract="HelloServiceWcf.IHelloService">
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:9080/" />
            <add baseAddress="net.tcp://localhost:9090"/>
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Once done with run the Console app to host your service:

```
Wcf started running....
```

Now we need to create two separate clients a web and other windows form application.

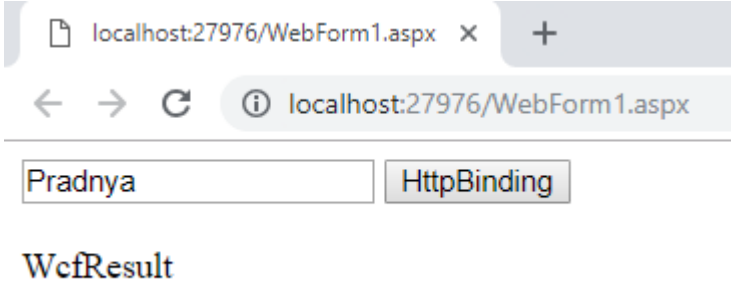
Asp Web Client:

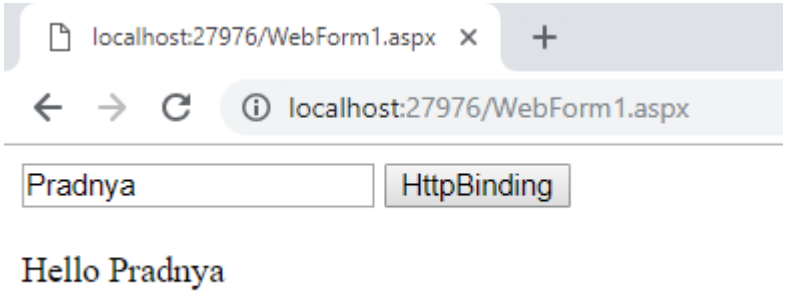


Add reference to your service similar but this time the url should be the one you specifeid in the configuration file (the http one). At last do changes in button click event.

```
WebForm1.aspx.cs*   WebForm1.aspx
WcfHttpClient   WcfHttpClient.WebForm1   TextBox1
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.UI;
6   using System.Web.UI.WebControls;
7   using WcfHttpClient.ServiceReference1;
8   namespace WcfHttpClient
9   {
10      public partial class WebForm1 : System.Web.UI.Page
11      {
12          protected void Page_Load(object sender, EventArgs e)
13          {
14
15          }
16
17          protected void Button1_Click(object sender, EventArgs e)
18          {
19              HelloServiceClient c = new HelloServiceClient("BasicHttpBinding_IHelloService");
20
21              Label1.Text=c.getMessage(TextBox1.Text);
22          }
23      }
24  }
```

When you run the web form:

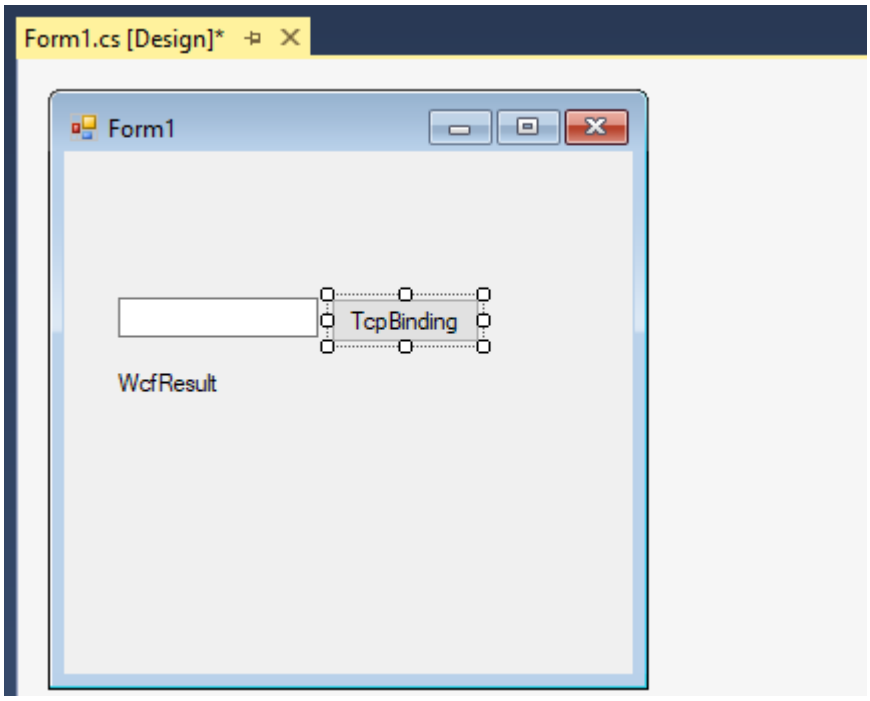




Now for next client create windows form application:

Make sure you add reference similar as previous

Design



On button click edit as follow:

```
Form1.cs   Form1.cs [Design]
WindowsTcpClient                    WindowsTcpClient.Form1                    butt
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Windows.Forms;
9   using WindowsTcpClient.ServiceReference1;
10  namespace WindowsTcpClient
11  {
12      public partial class Form1 : Form
13      {
14          public Form1()
15          {
16              InitializeComponent();
17          }
18
19          private void button1_Click(object sender, EventArgs e)
20          {
21              HelloServiceClient c = new HelloServiceClient("NetTcpBinding_IHelloService");
22              label1.Text = c.getMessage(textBox1.Text);
23          }
24      }
25  }
```

Finally when you run the project: