

CREACIÓN DE WEB SERVICE empleando NETBEANS para JAVA

Utilizando NETBEANS,

Requisitos Previos: (tener instalado el JEE y un servidor Web como GlassFish).

En la Página de descargas de Netbeans (<https://netbeans.org/downloads/>), descargar aquella versión que contiene el JEE y el GlassFish.

Supported technologies *	Java SE	Java EE
NetBeans Platform SDK	•	•
Java SE	•	•
Java FX	•	•
Java EE		•
Java ME		
HTML5/JavaScript		•
PHP		
C/C++		
Groovy		
Java Card™ 3 Connected		
Bundled servers		
GlassFish Server Open Source Edition 4.1.1		•
Apache Tomcat 8.0.27		•

Download
Download

1. Archivo, Nuevo Proyecto.
2. En Categoría, seleccionamos “Java Web”, “Web application”.
3. Seleccionamos “siguiente”,
4. Colocamos un nombre al proyecto, indicamos la ubicación del proyecto,
5. Seleccionamos “siguiente”,
6. Elegimos el servidor Web con el cual queremos trabajar. (Ej. Glassfish Server, Tomcat).
7. Elegimos la versión más actual de Java Enterprise Edition,
8. Escribimos la ruta de despliegue,
9. Seleccionamos “siguiente”,
10. Seleccionamos algún framework en caso es deseado programar aplicaciones Web.
11. Finalizar.

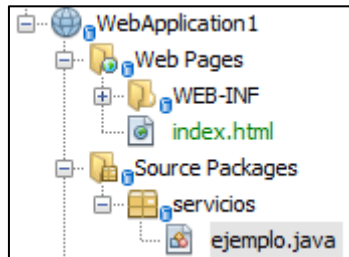
El proyecto presenta cuatro secciones:

- (1) Web Pages, dedicado a las páginas Web con extensión .html,
- (2) Source Packages, para las clases .java
- (3) Libraries, para las librerías y
- (4) Configuration files, para los archivos de configuración.

Dentro de la sección de (2) Source Packages, vamos a crear los archivos .java relacionados al servicio. En esta carpeta agregamos un paquete y posteriormente un elemento de tipo “Web Service”, que es posible encontrar dentro de la categoría “Web Services”.



Se obtiene una estructura similar a la siguiente:



La clase tiene la siguiente estructura:

```
package servicios;

import javax.ws.WebService;
import javax.ws.WebMethod;
import javax.ws.WebParam;

@WebService(serviceName = "ejemplo")
public class ejemplo {

    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```

Básicamente, el IDE nos muestra la estructura que debe contener la programación de un Servicio Web:

- La clase que pone a disposición de los clientes un servicio debe tener un encabezado con la siguiente notación:
@WebService(serviceName = "[nombre del servicio]").
Bajo el nombre especificado en ese formato será conocido el servicio por los clientes.
- Para cada uno de los métodos contenidos en el servicio, es necesario utilizar otro encabezado con la siguiente notación:
@WebMethod(operationName = "[nombre del método]"). Bajo el nombre especificado en este formato será conocido el método por los clientes.
- Para cada uno de los parámetros de cada método, es necesario utilizar otra notación:
@WebParam(name = "[nombre del parámetro]"). Bajo el nombre especificado en este formato será conocido en la parte cliente el nombre del parámetro que recibe el método.

Ejemplo de una clase donde está programado un servicio Web con cinco funciones:

- (1) sumar(a,b): que devuelve la suma de dos valores.
- (2) restar(a,b): que devuelve la resta de dos valores.
- (3) multiplicar(a,b): que devuelve la multiplicación de dos valores.
- (4) dividir(a,b): que devuelve la división de dos valores.
- (5) saludar(nombre): que devuelve un saludo en base a nombre.

```

package servicios;

import javax.ws.WebService;
import javax.ws.WebMethod;
import javax.ws.WebParam;

@WebService(serviceName = "ejemplo")
public class ejemplo {

    @WebMethod(operationName = "sumar")
    public double sumar(@WebParam(name = "a") double a,
                        @WebParam(name = "b") double b) {

        return a + b;
    }

    @WebMethod(operationName = "restar")
    public double restar(@WebParam(name = "a") double a,
                        @WebParam(name = "b") double b) {

        return a - b;
    }

    @WebMethod(operationName = "multiplicar")
    public double multiplicar(@WebParam(name = "a") double a,
                        @WebParam(name = "b") double b) {

        return a * b;
    }

    @WebMethod(operationName = "dividir")
    public double dividir(@WebParam(name = "a") double a,
                        @WebParam(name = "b") double b) {

        return a / b;
    }

    @WebMethod(operationName = "saludar")
    public String saludar(@WebParam(name = "nombre") String nombre) {
        return "Hola como estas? " + nombre;
    }
}

```

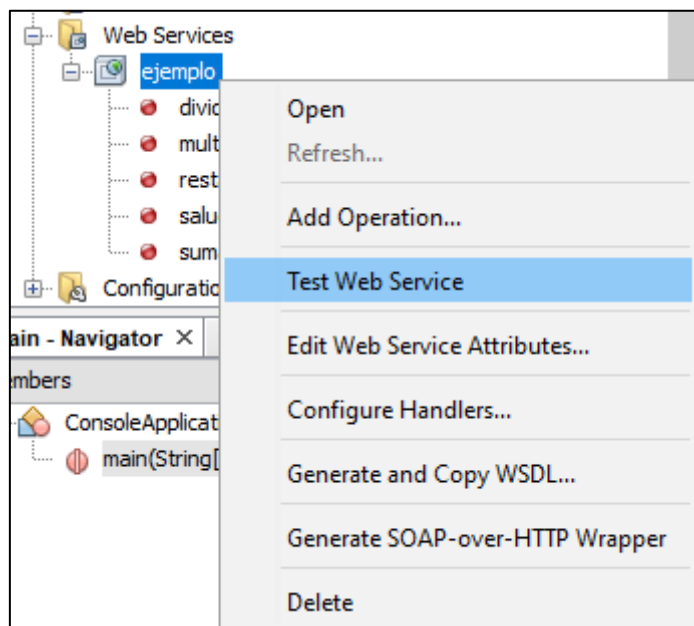
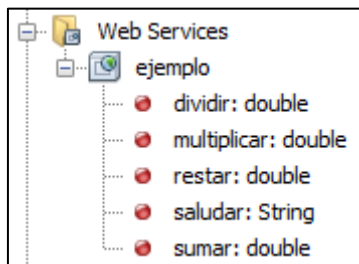
Procedemos a compilar el proyecto y a desplegarlo en el servidor Web (deploy). Si el despliegue fue exitoso, el servidor Web mostrará un mensaje similar al siguiente:

```

GlassFish Server 4.1.1 × WebApplication1 (run-deploy) ×
información: webservice endpoint deployed ejemplo
listening at address at http://PCCIX:8080/WebApplication1/ejemplo.
Información: Loading application [WebApplication1] at [/WebApplication1]
Información: WebApplication1 was successfully deployed in 704 milliseconds.

```

Para realizar pruebas a los Web services creados, ingresamos a la sección de “Web Services” dentro del proyecto. Debe existir un servicio creado con el nombre que hemos colocado en la notación serviceName de la clase. Click derecho al nombre del servicio (en este caso “ejemplo”) y a continuación “Test Web Service”.



También es posible acceder directamente a través del siguiente formato de URL:

```
http://[IP del Servidor]:[Puerto de ejecución del Servidor Web]/[nombre del proyecto]/[nombre del servicio]?Tester
```

Para mi caso específico es:

```
http://localhost:8080/WebApplication1/ejemplo?Tester
```

Muestra una pantalla como la siguiente:

ejemplo Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.lang.String servicios.Ejemplo.saludar(java.lang.String)

saludar ()

public abstract double servicios.Ejemplo.sumar(double,double)

sumar (,)

public abstract double servicios.Ejemplo.restar(double,double)

restar (,)

public abstract double servicios.Ejemplo.multiplicar(double,double)

multiplicar (,)

public abstract double servicios.Ejemplo.dividir(double,double)

dividir (,)

Procedemos a probar uno de los métodos: `saludar(nombre)`, para verificar que funciona correctamente:

Methods :

public abstract java.lang.String servicios.Ejemplo.saludar(java.lang.String)

saludar (Freddy Paz)

Verificamos que el valor retornado sea correcto:

Method returned

java.lang.String : **"Hola como estas? Freddy Paz"**

Asimismo, procedemos a probar alguna de las funciones matemáticas:

public abstract double servicios.Ejemplo.sumar(double,double)

sumar (23.4 , 12.98)

Observamos el resultado de la operación:

Method returned

double : **"36.379999999999995"**

Habiendo comprobado que funciona correctamente, procedemos a programar el cliente:

Crearemos un cliente en consola que consuma los servicios que ofrece nuestro servidor, para esto es necesario tener en cuenta que los servicios se encuentran disponibles en la siguiente dirección URL:

`http://[IP del Servidor]:[Puerto de salida del GlassFish]/[nombre del proyecto]/[nombre del servicio]?wsdl`

En mi caso, se ha generado la siguiente URL:

`http://localhost:8080/WebApplication1/ejemplo?WSDL`

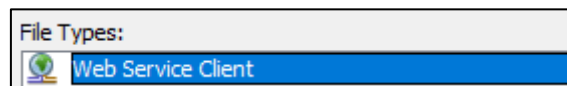
Esta URL será utilizada para crear el cliente.

También es posible obtener la dirección del WSDL haciendo click en el enlace “WSDL File” que aparece en la página de Testing (`http://localhost:8080/WebApplication1/ejemplo?Tester`)

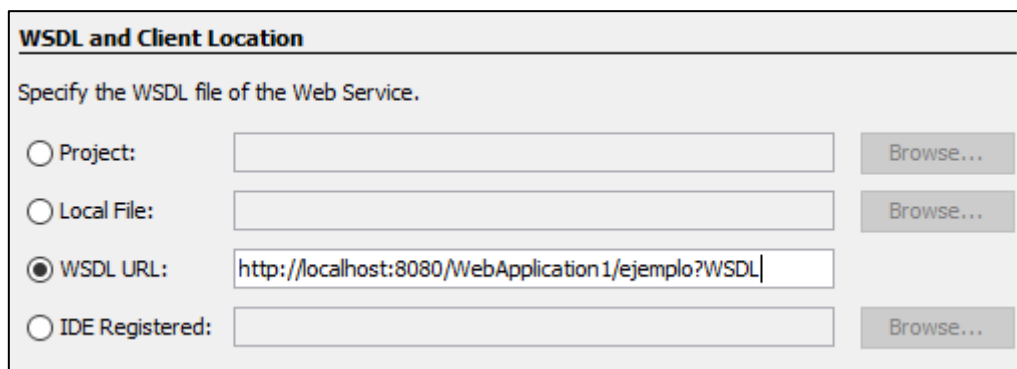
Creando el Cliente

1. Archivo -> Nuevo proyecto -> JAVA Application.
2. Seleccionamos “Siguiente”
3. Colocamos un nombre al proyecto.
4. Indicamos la ruta del proyecto.
5. Terminar.

En el proyecto autogenerado, tenemos que agregar un elemento de tipo “Web Service Client”.

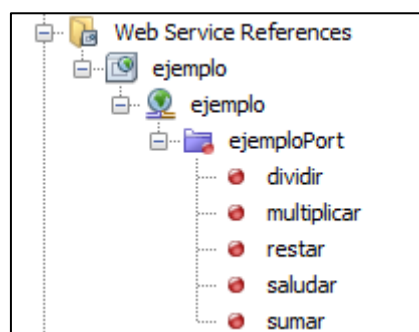


Nos solicitará ingresar la dirección WSDL:

A screenshot of the 'WSDL and Client Location' dialog box. The title is 'WSDL and Client Location'. Below the title, it says 'Specify the WSDL file of the Web Service.' There are four radio button options: 'Project:', 'Local File:', 'WSDL URL:', and 'IDE Registered:'. The 'WSDL URL:' option is selected. To the right of each option is a text input field and a 'Browse...' button. The 'WSDL URL:' field contains the text 'http://localhost:8080/WebApplication1/ejemplo?WSDL'.

Finalizamos y el proyecto recogerá del servidor toda la meta-data necesaria para la ejecución de los métodos que nos provee el servidor.

Podemos observar que en el cliente se ha generado la siguiente carpeta: “Web Service References”:



Asimismo, podemos observar que se han generado una serie de clases sobre el servicio:



Para hacer uso de los métodos que nos ofrece el servidor como servicio es necesario hacer uso de las clases:

<code>[nombre_del_servicio].java</code> <code>[nombre_del_servicio]_Service.java</code>
--

En este caso específico, el servicio fue codificado con el nombre de “ejemplo” en el servidor. Por tanto, en el cliente, las clases generadas y a ser utilizadas serían:

Ejemplo.java

Ejemplo_Service.java

La codificación necesaria para hacer uso de los métodos que nos ofrece el servicio del servidor son las siguientes:

1. Instanciamos un objeto de tipo Ejemplo_Service haciendo referencia al paquete al cual pertenece dentro de la carpeta “Generated Source”:

<code>servicios.Ejemplo_Service service = new servicios.Ejemplo_Service();</code>

2. Instanciamos un objeto de tipo Ejemplo a partir del objeto creado anteriormente:

<code>servicios.Ejemplo port = service.getEjemploPort();</code>

Con el objeto port es posible llamar a los métodos del servicio. Veamos la siguiente implementación:

```

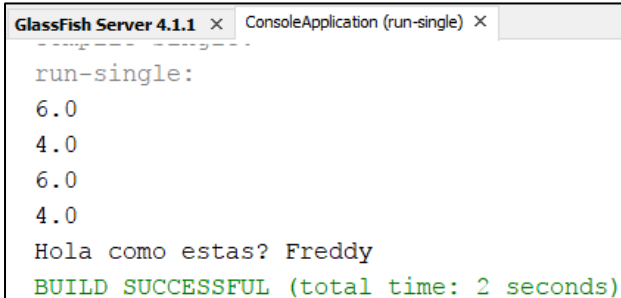
package consoleApplication;

public class ConsoleApplication {

    public static void main(String[] args) {
        servicios.Ejemplo_Service service = new servicios.Ejemplo_Service();
        servicios.Ejemplo port = service.getEjemploPort();
        double suma = port.sumar(2.5,3.5);
        double resta = port.restar(4.5, 0.5);
        double multiplicacion = port.multiplicar(2.0, 3.0);
        double division = port.dividir(8.0, 2.0);
        String saludo = port.saludar("Freddy");
        System.out.println(suma);
        System.out.println(resta);
        System.out.println(multiplicacion);
        System.out.println(division);
        System.out.println(saludo);
    }
}

```

Observemos el resultado:



The screenshot shows an IDE window titled 'GlassFish Server 4.1.1' and 'ConsoleApplication (run-single)'. The console output is as follows:

```

run-single:
6.0
4.0
6.0
4.0
Hola como estas? Freddy
BUILD SUCCESSFUL (total time: 2 seconds)

```

Ejercicio Propuesto:

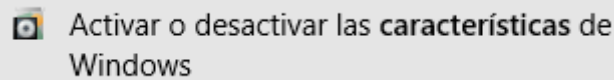
Utilizando la tecnología Web Service, implementar dos programas (1 programa Servidor que contenga la capa de modelo y acceso a datos para la gestión de registros en BD) y (1 programa Cliente que contenga la capa de presentación y lógica del negocio que consuma la capa de acceso a datos y el modelo del servidor para insertar, modificar, eliminar y listar registros almacenados en BD).

CREACIÓN DE WEB SERVICE empleando VISUAL STUDIO 2017 para C#

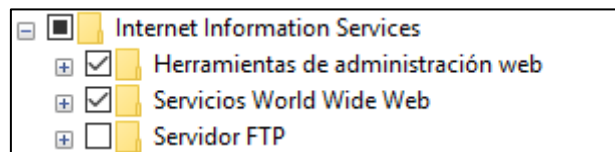
Requisitos Previos:

- Tener instalado el IIS (Internet Information Service).
- Tener instalado el paquete de desarrollo Web de Visual Studio 2017.

Para instalar el IIS, en Windows: Ingresar a la opción “Activar o desactivar las características de Windows”:



En la carpeta “Internet Information Services”, seleccionar todos las características de las carpetas: “Herramientas de administración web” y “Servicios World Wide Web”:

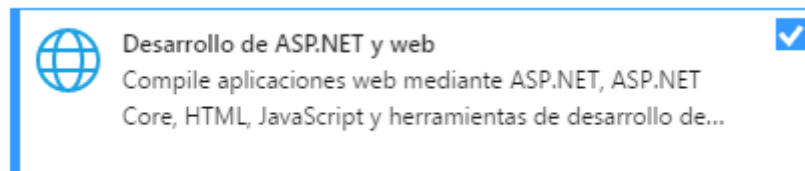


Aceptamos y Windows realizará la instalación del IIS (Servidor Web).

Para instalar las herramientas de desarrollo Web de Visual Studio, ejecutamos el instalador de Visual Studio:



Seleccionamos “Desarrollo de ASP.NET y web”:



Creando el Servidor:

1. Archivo -> Nuevo -> Proyecto
2. Elegimos “Aplicación de servicios WCF” dentro de la categoría de WCF.
3. Colocamos un nombre al proyecto
4. Elegimos la ubicación
5. Aceptar

Por defecto el proyecto crea una clase con extensión *.scv para la generación de un servicio de tipo Windows Communication Foundation. Sin embargo, en este caso no vamos a utilizar la tecnología WCF sino Web Service. Por lo tanto, procedemos a borrar tanto la interfaz como el archivo .scv creados por defecto. En vez de ello, procedemos a agregar un nuevo elemento de tipo “Servicio Web (ASMX)” [español] o “Web Service (ASMX)” [inglés].



Colocamos un nombre al elemento, para este ejemplo en particular le llamaré: ejemploWS.asmx

Se autogenera una clase con las siguientes etiquetas:

```
[WebService(Namespace = "http://tempuri.org/")]
```

Que hace referencia al espacio de nombres al cual pertenecerá el servicio. Y también la siguiente etiqueta:

```
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
```

Que hace referencia al estándar de comunicación “Basic Profile 1.1” por será utilizado para el envío de información.

Asimismo, cada método debe estar acompañado de la siguiente etiqueta: [WebMethod]

Procedemos a codificar un pequeño ejemplo de un Servicio que ofrece cinco funciones:

- suma(a, b): que devuelve la suma de dos valores.
- resta(a, b): que devuelve la resta de dos valores.
- multiplicacion(a, b): que devuelve la multiplicación de dos valores.
- division(a, b): que devuelve la división de dos valores.
- saludar(nombre): que devuelve un saludo en base a nombre.

La clase quedaría de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace ServidorWeb
{
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class ejemploWS : System.Web.Services.WebService
    {
        [WebMethod] public double sumar(double a, double b) {return a + b;}
        [WebMethod] public double restar(double a, double b) {return a - b;}
        [WebMethod] public double multiplicar(double a, double b) {return a * b;}
        [WebMethod] public double dividir(double a, double b) {return a / b;}
        [WebMethod] public String saludar(String nombre) {return "Hola "+nombre;}
    }
}
```

Para desplegar el Web Service, es necesario indicar el servidor Web. Damos click derecho al proyecto, propiedades, y en la pestaña Web, seleccionamos “IIS local”. Es mejor utilizar la versión local del Internet Information Service (IIS) que la versión limitada Express de IIS.

Aplicación

Compilación

Web*

Empaquetar/publicar web

Empaquetar/publicar SQL

Eventos de compilación

Recursos

Configuración

Rutas de acceso de referencias

Firma

Análisis de código

Configuración: N/A Plataforma: N/A

Argumentos de la línea de comandos

Directorio de trabajo

☐ Dirección URL de inicio

☐ No abrir una página. Esperar una solicitud de una aplicación externa.

Servidores

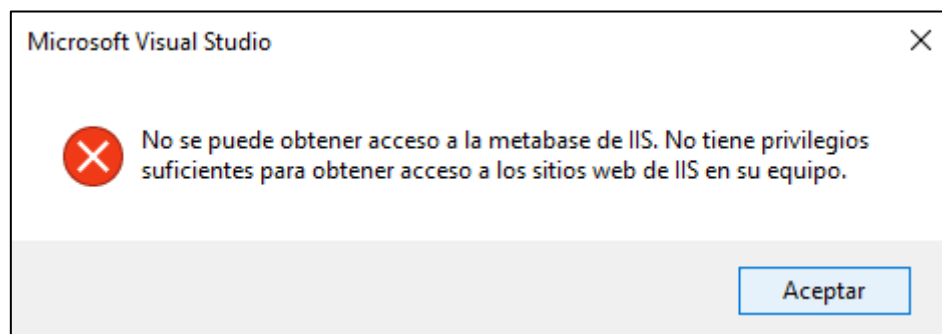
☒ Aplicar configuración del servidor a todos los usuarios (almacenar en archivo de proyecto)

IIS local

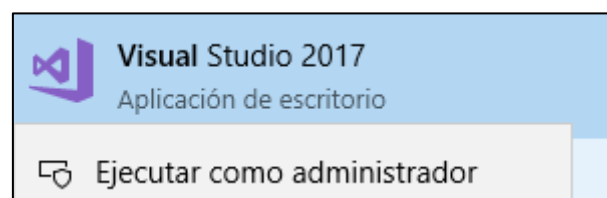
URL del proyecto

☐ Reemplazar dirección URL raíz de la aplicación.

Si nos aparece el siguiente mensaje de error, únicamente es necesario ejecutar el Visual Studio 2017 con privilegios de administrador.



Ejecutamos como administrador:



Volvemos a abrir el proyecto, seleccionamos “IIS local”,

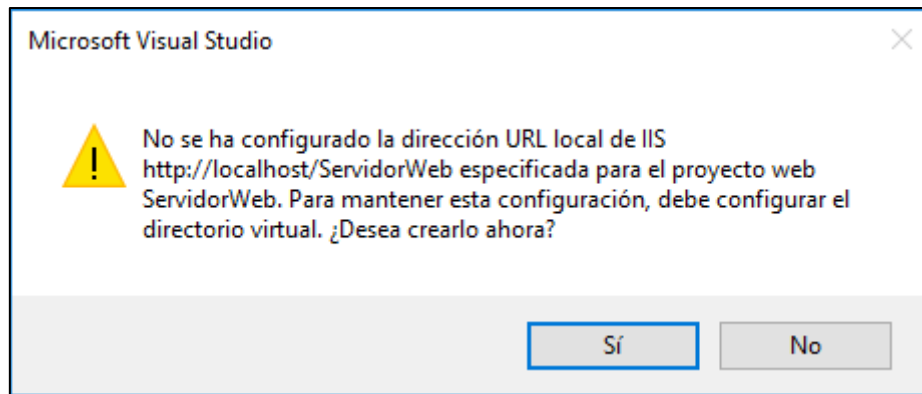
☒ Aplicar configuración del servidor a todos los usuarios (almacenar en archivo de proyecto)

IIS local

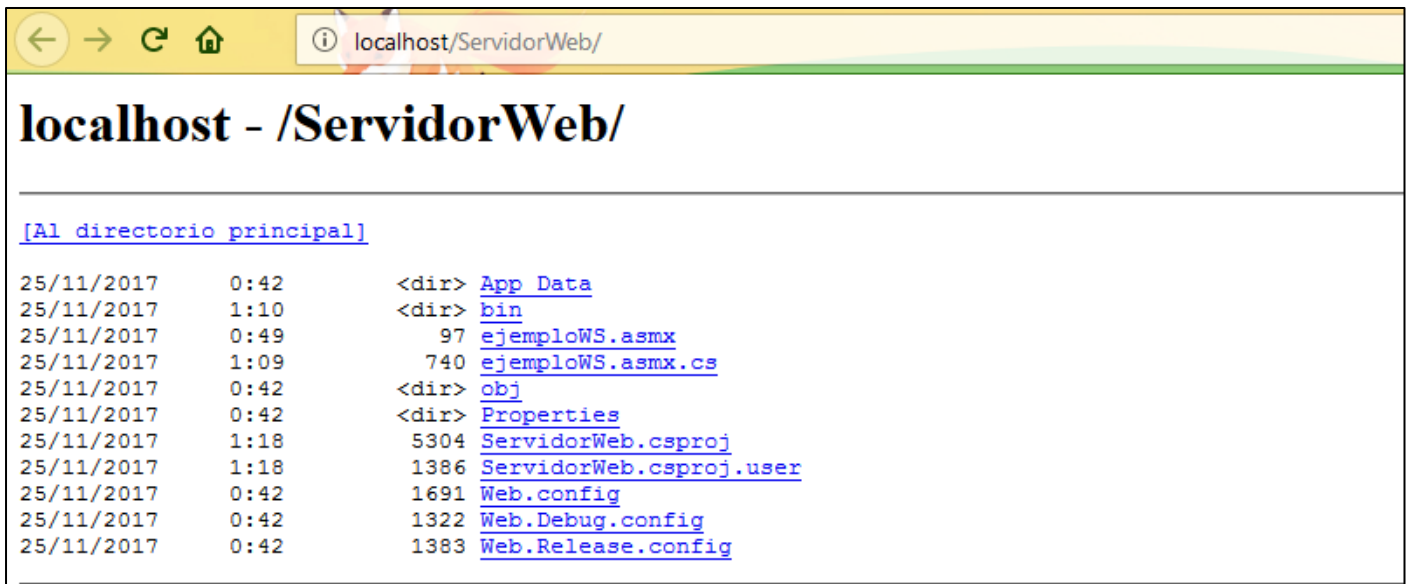
URL del proyecto

Compilamos y ejecutamos:

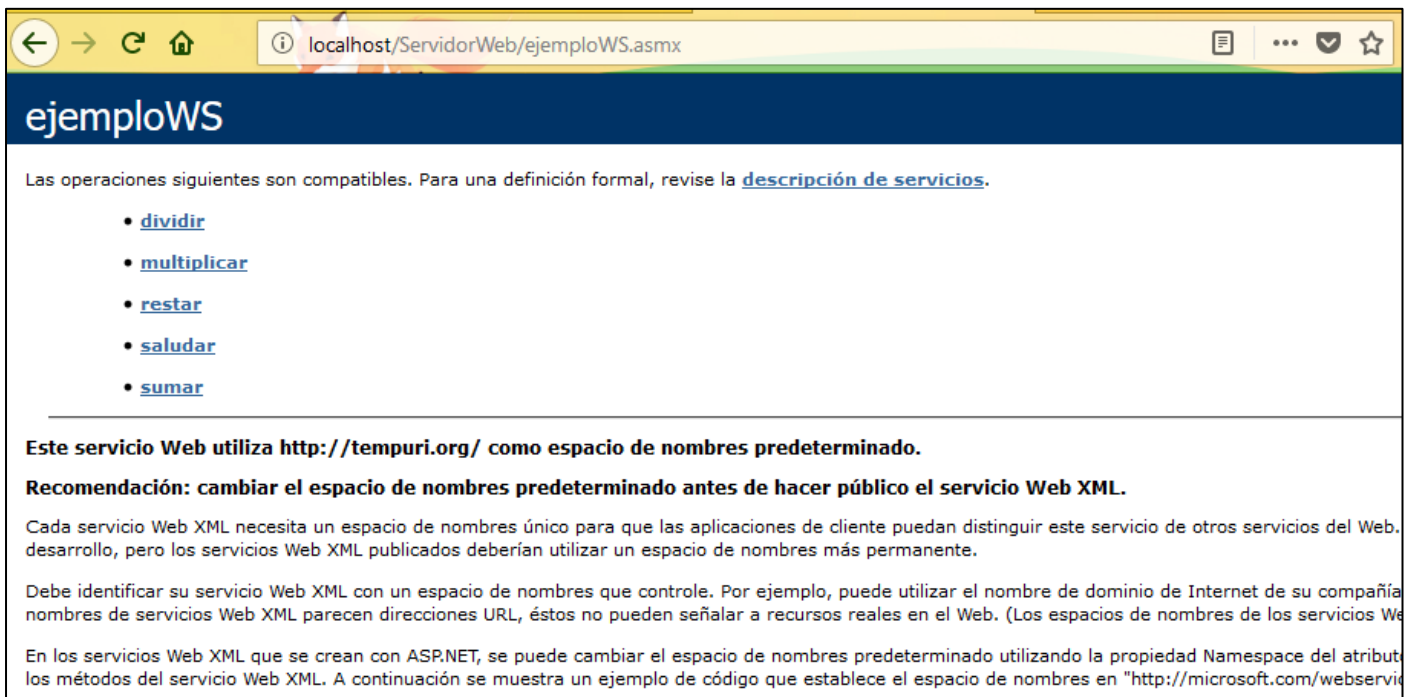
Nos va a solicitar permiso para crear la estructura de directorios para desplegar el servicio dentro del Servidor Web:



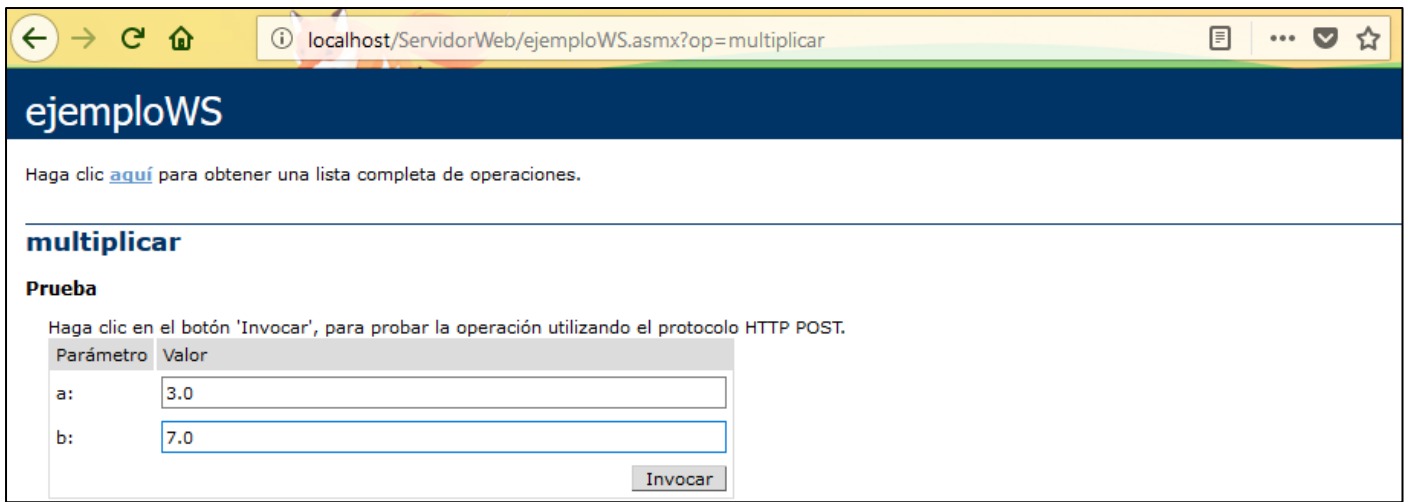
Aceptamos y nos va a mostrar en un explorador de páginas Web, la dirección Web donde está desplegado el proyecto:



Ingresamos al servicio: En mi caso: ejemploWS.asmx. Nos aparece la siguiente pantalla, donde se listan todos los métodos que están disponibles en el servicio. Si hacemos click en “descripción de servicios” es posible obtener la dirección WSDL asociada al servicio.



Para probar que los métodos se están ejecutando apropiadamente, hacemos click en alguno de ellos:



← → ↺ 🏠 ⓘ localhost/ServidorWeb/ejemploWS.asmx?op=multiplicar

ejemploWS

Haga clic [aquí](#) para obtener una lista completa de operaciones.

multiplicar

Prueba

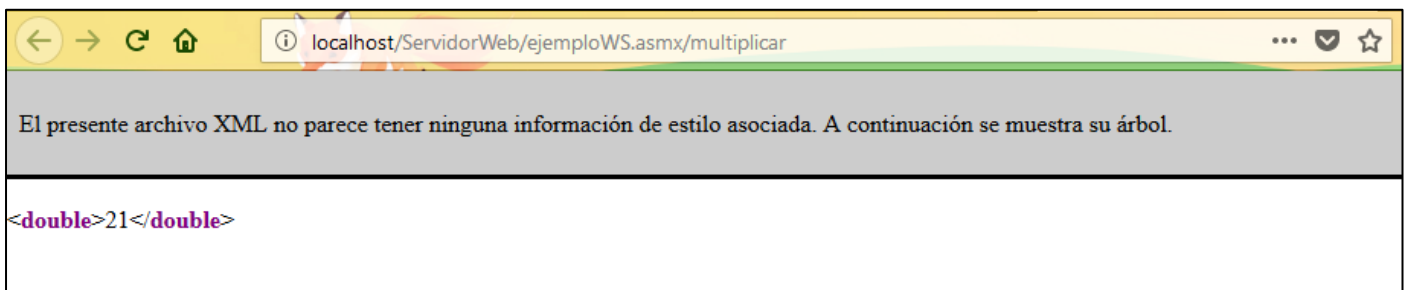
Haga clic en el botón 'Invocar', para probar la operación utilizando el protocolo HTTP POST.

Parámetro	Valor
a:	<input type="text" value="3.0"/>
b:	<input type="text" value="7.0"/>

Invocar

Por ejemplo: “multiplicar”. Escribimos los valores e invocamos el método.

Verificamos que los valores retornados sean correctos:



← → ↺ 🏠 ⓘ localhost/ServidorWeb/ejemploWS.asmx/multiplicar

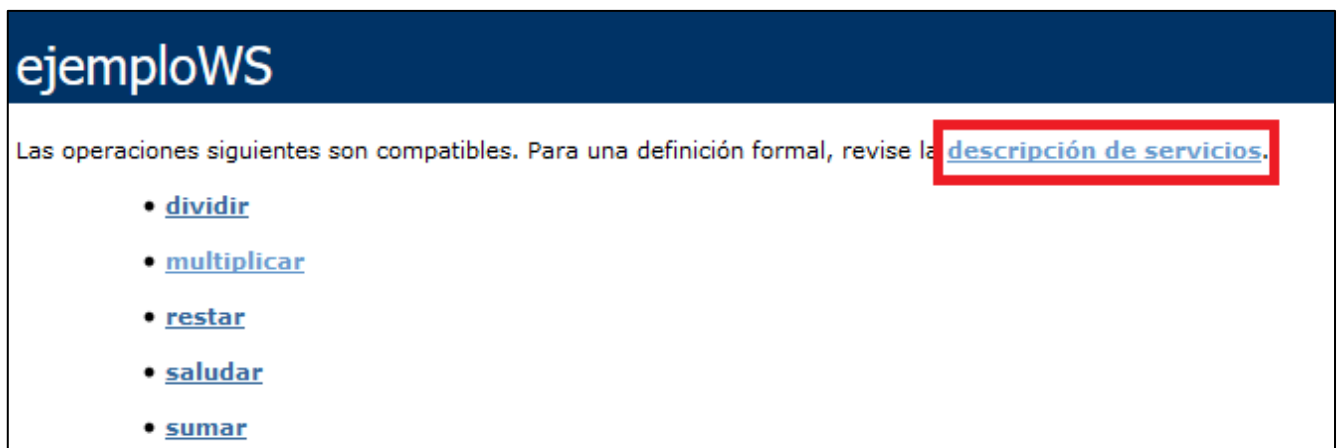
El presente archivo XML no parece tener ninguna información de estilo asociada. A continuación se muestra su árbol.

```
<double>21</double>
```

La dirección WSDL tiene el siguiente formato:

http://[IP del Servidor]:80/[Nombre del proyecto]/[Nombre de la clase que contiene la codificación del servicio].asmx?WSDL

También es posible obtener la dirección WSDL haciendo click en “descripción de servicios”.



ejemploWS

Las operaciones siguientes son compatibles. Para una definición formal, revise la [descripción de servicios](#).

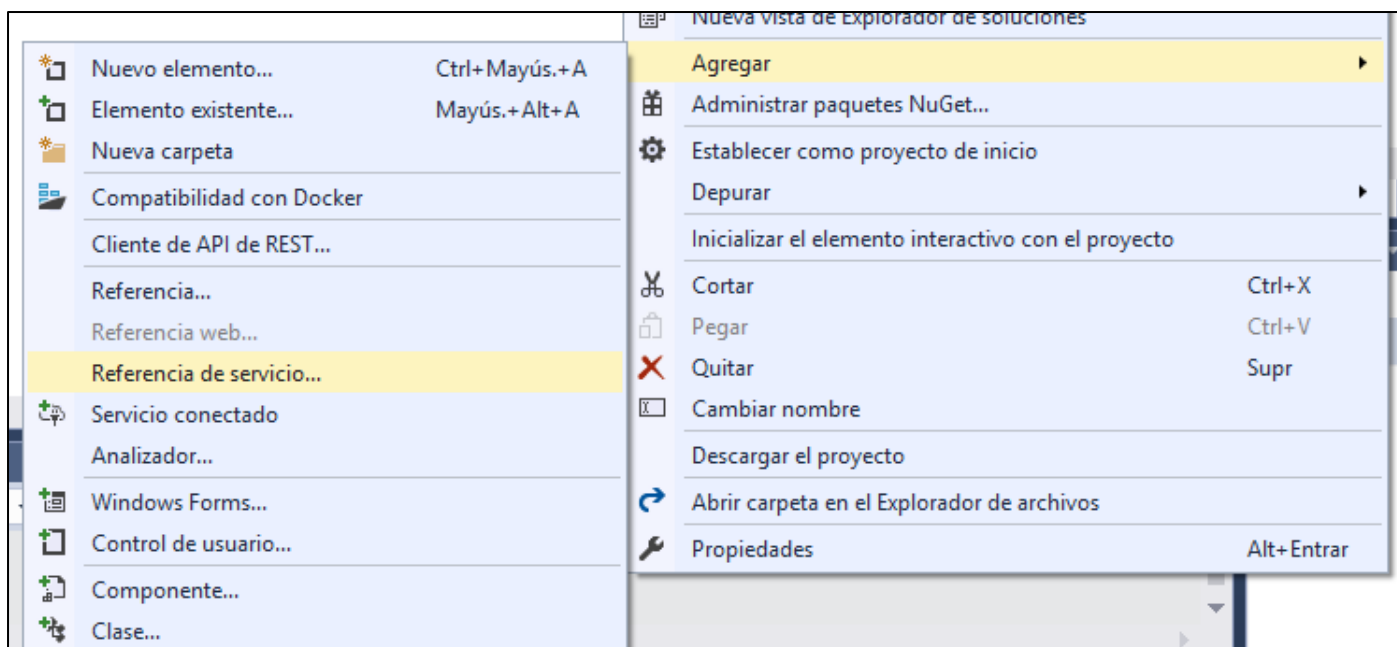
- [dividir](#)
- [multiplicar](#)
- [restar](#)
- [saludar](#)
- [sumar](#)

Habiendo realizado las pruebas y teniendo la dirección WSDL ya es posible codificar el cliente.

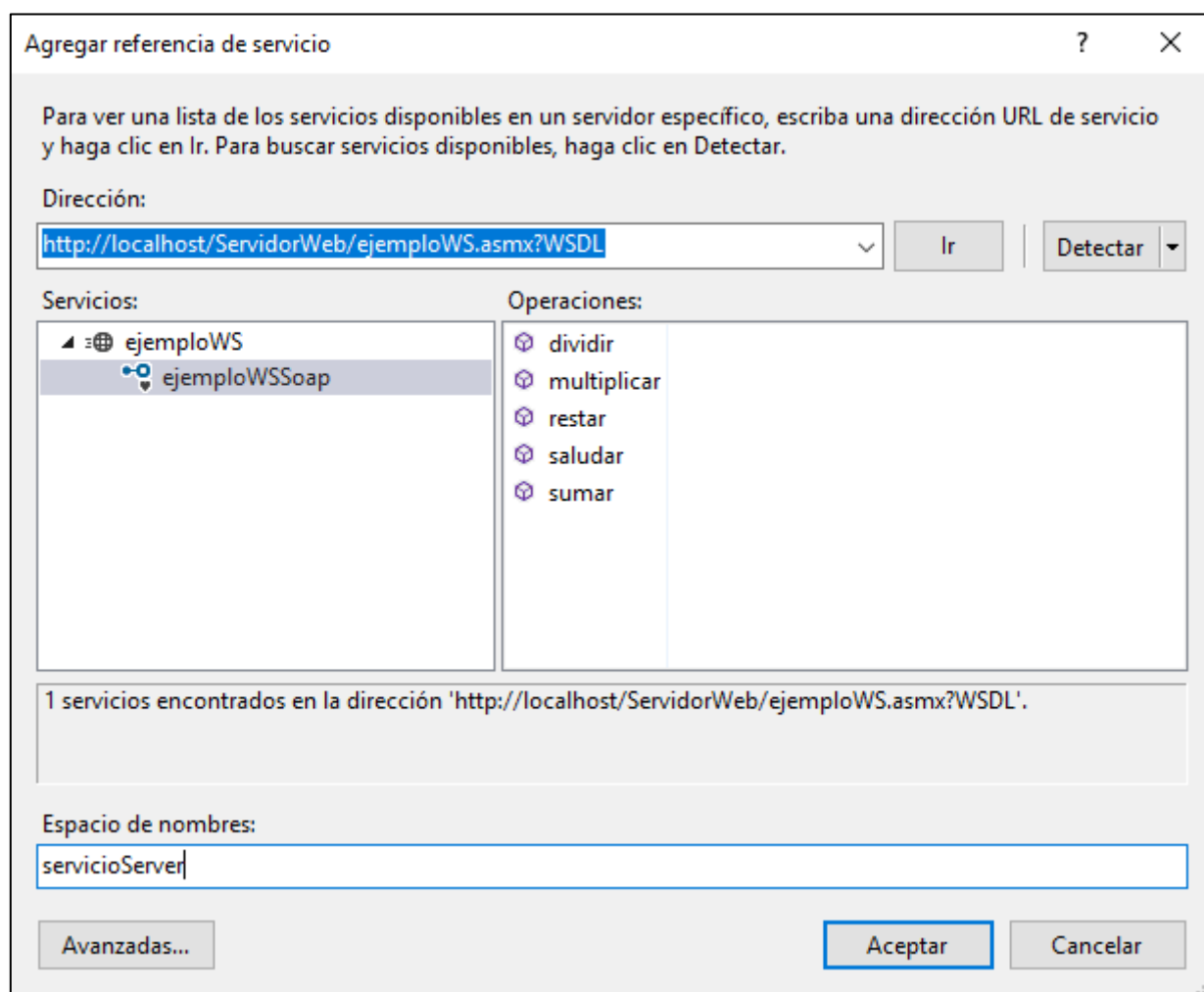
Creando el cliente:

1. Archivo -> Nuevo Proyecto
2. Es posible tener como cliente un proyecto de tipo “Application Windows Forms” (.NET Framework) o “Console Application” (.NET Framework).
3. En este caso, seleccionaré “Aplicación de consola”.
4. Colocamos un nombre, elegimos la ruta y colocamos un nombre al proyecto.

En el proyecto autogenerado, tenemos que agregar una referencia de servicio. Click derecho al proyecto, agregar, referencia de servicio.

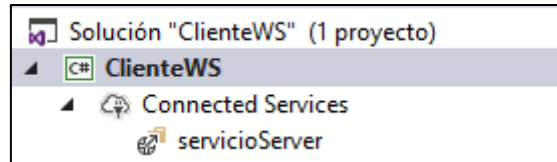


Nos solicitará ingresar la dirección WSDL donde está alojado el servicio. La ingresamos, hacemos click en IR y se mostrará el servicio y todos sus métodos asociados (que pone a disposición el servidor).



Colocamos un nombre en espacio de nombres, Aceptamos. El espacio de nombres servirá para posteriormente llamar a las clases alojadas en el servidor.

En el proyecto debe aparecer la referencia al servicio:



Para hacer uso de los métodos que nos ofrece el servidor como servicio es necesario hacer uso de la siguiente clase:

[nombre del servicio]SoapClient

1. Para mi caso específico, instanciamos un objeto de tipo ejemploWSSoapCliente haciendo referencia al espacio de nombres al cual indicamos que pertenecería. En mi caso específico:

```
servicioServer.ejemploWSSoapClient acceso = new servicioServer.ejemploWSSoapClient();
```

Con el objeto creado es posible llamar a los métodos del servicio. Veamos la siguiente implementación:

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

namespace ClienteWS
{
    class Program
    {
        static void Main(string[] args)
        {
            servicioServer.ejemploWSSoapClient acceso = new
servicioServer.ejemploWSSoapClient();
            double suma = acceso.sumar(2.0,3.0);
            double resta = acceso.restar(2.5, 2.0);
            double multiplicacion = acceso.multiplicar(2.0, 5.0);
            double division = acceso.dividir(10.0, 2.0);
            String saludo = acceso.saludar("Freddy");
            System.Console.WriteLine(suma);
            System.Console.WriteLine(resta);
            System.Console.WriteLine(multiplicacion);
            System.Console.WriteLine(division);
            System.Console.WriteLine(saludo);
            System.Console.Read();
        }
    }
}
```

Observamos el resultado:

```
c:\users\freddy\documents\visual studio 2017\Projects\ClienteWS\ClienteWS\bin\Debug\ClienteWS.exe
5
0.5
10
5
Hola Freddy
```

Ejercicio propuesto:

Desarrollar el Laboratorio 9 del semestre 2017-2 (disponible en PAIDEIA) colocando la capa de acceso a datos y modelo en JAVA y la capa de lógica de negocio y presentación en C#. La comunicación entre ambos programas debe ser a través de Web Services. Referencia: Observar el ejercicio desarrollado en la última clase.

Notas:

Si el cliente es C# y maneja distintas capas (ej. Lógica de Negocio y Presentación), el archivo app.config generado al momento de importar la referencia al servicio, debe ser copiado a todas las capas.

Si tenemos un servidor C# y un cliente JAVA, la forma de generar un objeto en el lado del cliente que permita llamar a los métodos del servicio que ofrece el servidor es la siguiente:

```
paquete.[nombre del servicio] obj = new paquete.[nombre del servicio]();  
paquete.[nombre del servicio]Soap port = obj.
```

Tomando como ejemplo el servidor desarrollado en C# en esta guía:

El cliente implementaría las siguientes líneas de código:

```
org.tempuri.EjemploWS servicio = new org.tempuri.EjemploWS();  
org.tempuri.EjemploWSSoap acceso = servicio.getEjemploWSSoap();  
double resultado = acceso.sumar(2, 2);  
System.out.println(resultado);
```