PRÁCTICA 3

CONTROL PARKING SERVICIOS WEB.

- INDEX -

Introducción

Esta práctica tiene como objetivo adquirir conocimientos sobre la tecnología de Servicios Web y estudiar el paradigma de computación distribuida SOA.

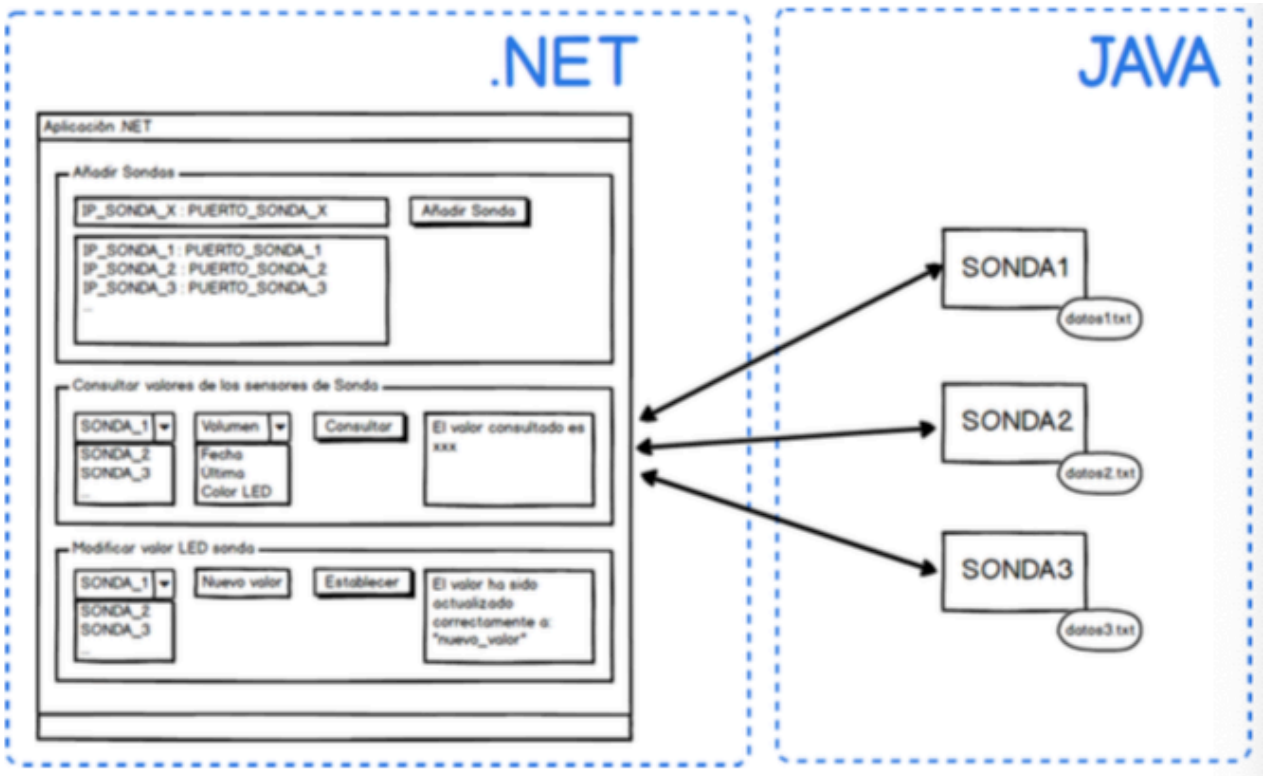
Por un lado **SOA** es un paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos, destacan su facilidad y flexibilidad de integración.

Por otro lado los **Servicios Web** utilizan protocolos y estándares que nos sirven para intercambiar datos entre aplicaciones.

En está práctica se desarrolla este sistema añadiéndole unos métodos de **seguridad** básicos. El enfoque de esta práctica es **B2B** (Bussines to Bussines), es decir, que vamos a simular ser una empresa que ofrecen sus servicios a otras empresas. Nuestra empresa va a ser un controlador de las plazas de un parking, dando información del estado que cada una de estas a los clientes que lo soliciten.

En el proyecto, un objetivo es reutilizar el documento **WSDL** que define los servicios de tiene un sensor y los expone de manera remota, para que un cliente pueda consumirlos. Los sensores van a ser expuestos como servicios web para que un cliente, mediante una aplicación **.NET**, pueda acceder a ellos y consumirlos.

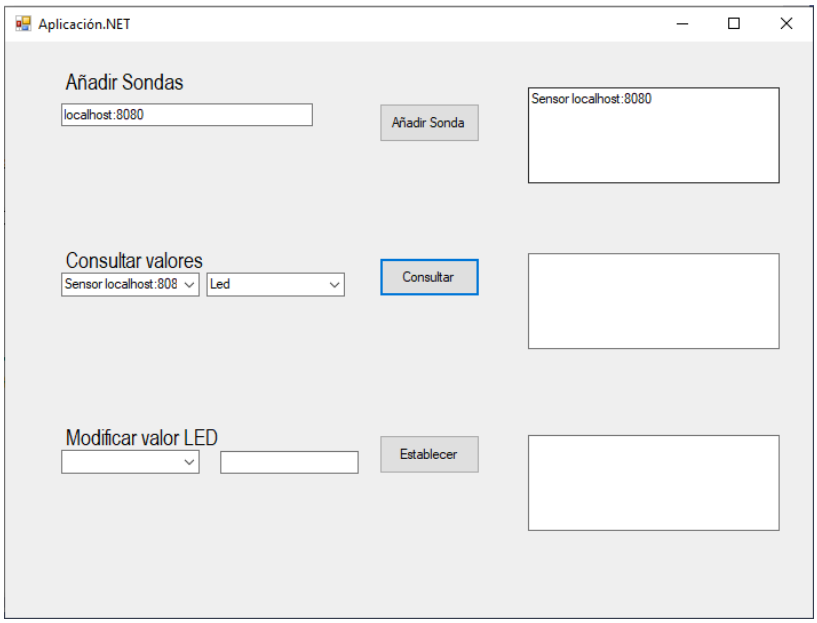
El esquema nos sirve de guía.



Aplicación cliente .NET

Implementa la lógica necesaria para acceder a un determinado servicio (sensor) y consultar o modificar sus variables.

Esta es la interfaz de la aplicación.



1. Se indica la **IP** y el **PUERTO** donde se encuentra el sensor que queremos consultar.

2. Las sondas añadidas se listan y una vez seleccionada una, se elige que variable queremos **consultar**.

3. Por último, seleccionada una sonda podemos cambiarle el valor de la variable **led**.

De la parte del código que hay que destacar es al añadir los sensores. En el código de la siguiente imagen se añade un sensor y es muy importante que la variable url se cambie y se ponga la dirección que hace referencia a **Soap11EndPoint,** pues es donde los servicios funcionan. Si el sensor ya está añadido en la lista de sensores, no se añade al sistema, en caso de añadirse el sensor, se le cambia el nombre para que en el nombre tengamos información de dónde se sitúa este servicio. Por último llamamos al método **escribir()** para que los añada al **textbox** para que el cliente pueda acceder a este sensor.

```
List<localhost.Sensor> listaSensores = new List<localhost.Sensor>();
static int cont = 0;

private void Button1_Click(object sender, EventArgs e)
{
    String ip_port = tbAddSonda.Text;
    localhost.Sensor sensor = new localhost.Sensor();

    sensor.Url = "http://" + ip_port + "/Sensor/services/Sensor.SensorHttpSoap11Endpoint/";
    sensor.readSonda();

    bool exist = false;
    for (int i = 0; i < listaSensores.Count; i++)
    {
        if (sensor.Url == listaSensores[i].Url)
        {
            exist = true;
        }
    }

    if (exist == false)
    {
        sensor.setNombre("Sensor " + ip_port);
        sensor.readSonda();
        listaSensores.Add(sensor);
    }
    escribirSondas();
}
```

Un método importante, que refleja como funciona el consumo de servicios, es el método **consultar()**. Se busca el sensor especificado en la lista de sensores añadidos y se envía la petición de consulta al sensor, encriptando el mensaje previamente. La encriptación se explicará más adelante.

```
private void BConsultar_Click(object sender, EventArgs e)
{
    string nombreSensor = cbSondasConsultar.Text;
    string valor = cbAtributos.Text;

    for (int i = 0; i < listaSensores.Count; i++)
    {
        if (nombreSensor == listaSensores[i].getNombre())
        {
            descConsultar.Text = listaSensores[i].consultar(Encryptar(valor), Login.user);
        }
    }

}
```

Por último el método **modificar()** es el que se encarga de cambiar el valor de la led, del sensor indicado. Una vez cambiado led del sensor, se muestra por pantalla y se guarda este valor en el sensor.

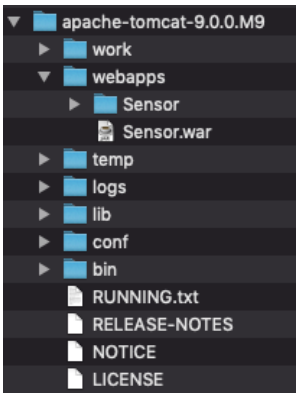
```
private void Button3_Click(object sender, EventArgs e)
{
    string nombreSensor = cbSondasModificar.Text;
    string newValue = tbValor.Text;

    for (int i = 0; i < listaSensores.Count; i++)
    {
        if (nombreSensor == listaSensores[i].getNombre())
        {
            listaSensores[i].setLed(int.Parse(newValue), Login.user);
            listaSensores[i].saveSensor();
            descModificar.Text = listaSensores[i].getNombre() + " modificando.";
            listaSensores[i].readSonda();
        }
    }

}
```

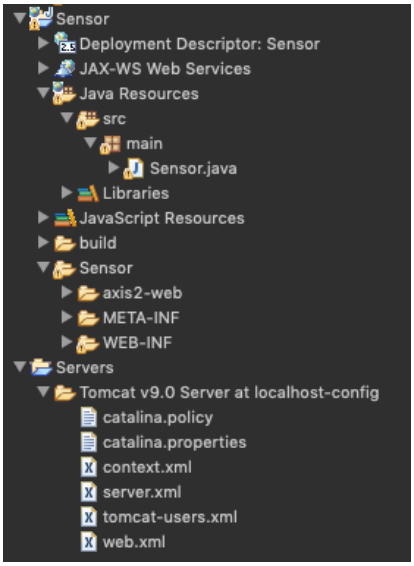
Servicios Sonda

Para que los servicios de una sonda se publiquen utilizamos el servidor **apache tomcat v9**, este nos permitirá lanzar los servicios y que sean accedido a ellos de forma remota, también nos proporciona el archivo **WSDL** para que sea más fácil utilizar este servicio.



Aquí se puede ver el directorio dentro de la carpeta apache tomas versión 9. Creamos un .war del servicio web de la clase sensor y lo metemos dentro de la carpeta **webapps**. Una vez lanzado el servidor tomcat, los servicios del sensor ya están publicados.

En la carpeta **bin** es donde guardaremos los ficheros de texto que vayamos a utilizar.



La clase Sensor.java es la misma que la de la práctica anterior con algunas pequeñas modificaciones, para que se adapte a esta práctica.

Este es el directorio del proyecto para el **Sensor**. La carpeta sensor, es la que tiene la información necesaria para que sea un **Servicio Web.**

Una modificación que he hecho es añadir el campo **nombre** a la clase de esta manera cuando se lanza el sensor, también se guarda donde está lanzado.

También se le añade una lógica para que guarde información de cuándo se ha lanzado y quien accede a los servicios. Se explicará más adelante, pues es la parte que atiende a **seguridad.**

```java
public class Sensor {

    private String nombre = "Sensor1";
    public String ultimaFecha = "11/12/20";
    public int volumen = 10;
    public int led = 0;


    public Sensor()
    {
        readSonda();

        String str = nombre + " dando servicios " + LocalDateTime.now() + "\n";
        BufferedWriter writer;
        try {
            writer = new BufferedWriter(new FileWriter("Log.txt", true));
            writer.write(str);
            writer.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }


    }
```

Seguridad

Para la parte de seguridad se pedía que tuviera un sistema de login, para el cliente, que se guardara información de quién y cuando se pide un servicio y un sistema de encriptado en el paso de mensajes entre servidor y cliente.

Empezamos por el sistema de login. Un archivo .txt guarda la información de los usuarios y su contraseña, como vemos en el ejemplo hay un usuario y una contraseña que está encriptada.

```
saav1
K058aodI9hM=
```

Al introducir la contraseña, la aplicación encripta la contraseña y la compara con la contraseña guardada en la base de datos, que en este caso es un archivo .txt.

Para la encriptación de la contraseña se ha utilizado la librería de c# **Cryptography.** Dentro de una clase estática con dos métodos uno para encriptar y otro para desencriptar.

En la imagen se puede ver un ejemplo de la utilización de la librería encriptando una cadena de texto (la **contraseña**).

```
class EncryptAlgorithm
{

    public static String Encrypt(string source)
    {
        DESCryptoServiceProvider des = new DESCryptoServiceProvider();
        byte[] Key = { 12, 13, 14, 15, 16, 17, 18, 19 };
        byte[] IV = { 12, 13, 14, 15, 16, 17, 18, 19 };

        ICryptoTransform encryptor = des.CreateEncryptor(Key, IV);

        try
        {
            byte[] IDToBytes = ASCIIEncoding.ASCII.GetBytes(source);
            byte[] encryptedID = encryptor.TransformFinalBlock(IDToBytes, 0, IDToBytes.Length);
            return Convert.ToBase64String(encryptedID);
        }
        catch (FormatException)
        {
            return null;
        }
        catch (Exception)
        {
            throw;
        }
    }
```

Una captura de ejemplo de funcionamiento.

```
'clienteNet.exe' (CLR v4.0.30319: cliente
Cadena encriptada >> K058aodI9hM= <<
Cadena desencriptada >> a123 <<
'clienteNet.exe' (CLR v4.0.30319: cliente
```

Para guardar información sobre las conexiones en la parte del cliente tenemos el fichero de texto **log.txt,** donde se almacena el nombre del usuario y tanto la fecha como la hora a la que se ha conectado.

```
Usuario saav1 conectado el 13/12/2019 11:33:14
Usuario saav1 conectado el 13/12/2019 14:24:47
Usuario saav1 conectado el 16/12/2019 14:32:38
Usuario saav1 conectado el 16/12/2019 14:32:38
Usuario saav1 conectado el 16/12/2019 14:32:38
```

El siguiente método es el encargado de aplicar la lógica para el login. Se encarga de encriptar la contraseña, de comparar y verificar que es la correcta, y de escribir en el fichero de texto los datos correspondientes, para que quede constancia de su inicio de sesión.

```
private void Button1_Click(object sender, EventArgs e)
{
    string password = tbPassword.Text;
    string scr = EncryptAlgorithm.Encrypt(password);
    Console.WriteLine("Cadena encriptada >> " + scr + " <<");
    string dscr = EncryptAlgorithm.Decrypt(scr);
    Console.WriteLine("Cadena desencriptada >> " + dscr + " <<");

    if (user == tbUser.Text && scr == pass)
    {
        this.Hide();
        Form1 f1 = new Form1();
        f1.Show();

        using (System.IO.StreamWriter file =
        new System.IO.StreamWriter(@"C:\Users\EPS\Desktop\SD\clienteNet\log.txt", true))
        {
            file.WriteLine("Usuario " + user + " conectado el " + DateTime.Now);
        }
    }
    else
    {
        Console.WriteLine(user + "/" + tbUser.Text);
        Console.WriteLine("Pas: " + pass);
        Console.WriteLine("SCR: " + scr);
    }
}
```

Esto es lo que tiene que ver en la parte del cliente con seguridad. Ahora vamos a ver lo que tiene que ver en la parte del servidor.

Para encriptar los mensajes entre el servidor y el cliente, he hecho mis propios métodos de encriptación directa. Uno en java para el servidor y un en c# para el cliente.

En el cliente antes de hacer una petición se encripta con el siguiente método.

```
private String Encryptar(string valor) {

    char[] ctr = valor.ToCharArray();

    StringBuilder s = new StringBuilder("");

    for (int i = 0; i < ctr.Length; i++) {
        s.Append(Char.ToString((char)(ctr[i] + (char)23)));
    }
    Console.WriteLine("Petición encriptada : " + s);
    return s.ToString();
}
```

En la parte de java, se desencripta la petición que ha hecho el cliente. Y luego se envía la información deseada por el cliente.

```
public String consultar(String str, String usuario) {

    String str2 = Desencriptar(str);
    System.out.println("Petición desencriptada: s " + str2);
    String res = "";
    switch(str2)
    {
    case "Volumen":
        res = String.valueOf(getVolumen(usuario));
        break;
    case "Led":
        res = String.valueOf(getLed(usuario));
        break;
    case "Fecha":
        res = getFecha(usuario);
        break;

    }
    return res;
}

public String Desencriptar(String str) {
    char[] ctr = str.toCharArray();
    StringBuilder s = new StringBuilder("");
    for(int i = 0; i < ctr.length; i++) {
        s.append(((char)(ctr[i] - (char)23)));
    }
    return s.toString();
}

public int getVolumen(String usuario)  {
    saveLog(usuario, "GET VOLUMEN");
    return volumen;
}
```

Este es un resultado de la operación en las dos partes, al hacer una consulta de volumen y de led.

```
13-Dec-2019 17:30:10.929 INFO [main] org.apache.ca
Petición desencriptada: Volumen
Petición desencriptada: Led

Cadena encriptada >> k058aod19nM= <<
Cadena desencriptada >> a123 <<
'clienteNet.exe' (CLR v4.0.30319: clienteNet.exe):
'clienteNet.exe' (CLR v4.0.30319: clienteNet.exe):
Petición encriptada : m????|?
Petición encriptada : c|{
```

Por último explicaré los logs en el servidor. Cada sensor cuando un usuario utiliza uno de sus servicios queda registrado en el fichero de texto **Log.txt.**

Se le pasa como parámetros el usuario que ha hecho la petición y el método que ha accedido. Y se escribe esta información para que quede guardada.

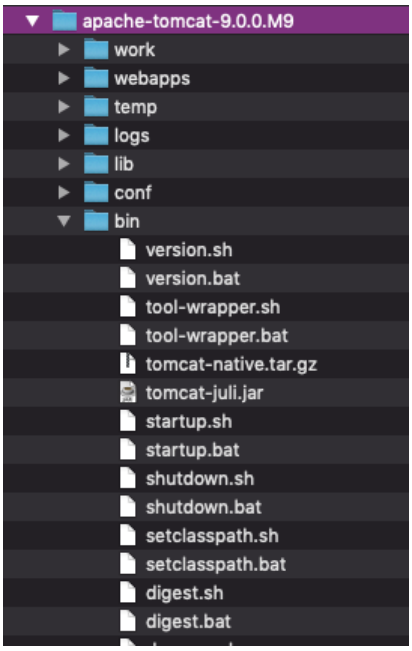
```
public void saveLog(String usuario, String metodo) {
    String str = "Sensor " + nombre + " dando servicio " + metodo +  " a "+ usuario + " el " + LocalDateTime.now() + "\n";
    BufferedWriter writer;
    try {
        writer = new BufferedWriter(new FileWriter("Log.txt", true));
        writer.write(str);
        writer.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Un ejemplo de la información que se almacena es la siguiente:

```
Sensor localhost:8080 dando servicios 2019-12-13T15:49:46.476
Sensor Sensor localhost:8080 dando servicio GET FECHA a saav1 el 2019-12-13T15:49:46.491
Sensor localhost:8080 dando servicios 2019-12-13T15:57:04.148
Sensor localhost:8080 dando servicios 2019-12-13T16:03:08.413
Sensor Sensor localhost:8080 dando servicio GET FECHA a saav1 el 2019-12-13T16:03:08.456
Sensor localhost:8080 dando servicios 2019-12-13T16:03:46.736
Sensor localhost:8080 dando servicios 2019-12-13T16:03:46.749
Sensor Sensor localhost:8080 dando servicio GET FECHA a saav1 el 2019-12-13T16:03:46.772
Sensor localhost:8080 dando servicios 2019-12-13T16:03:48.806
Sensor localhost:8080 dando servicios 2019-12-13T16:03:48.819
```

Guía de despliegue.

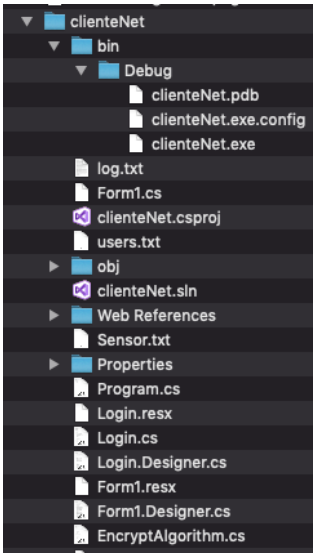
1. Una vez creamos el **Servicio Web** del Sensor y extraemos este en un archivo war y lo ponemos en la carpeta webapps del servidor apache **tomcat .v9.** Solo tenemos que inicializar el servidor ejecutando el archivo:



startup.bat > Windows.

startup.sh > Linux y MacOs (basados en Unix).

2. Para el cliente .net solo hay que ejecutar el documento .exe que encontramos en la carpeta Debug del proyecto.



clienteNet.exe