

Control Parking

The background of the page features a series of parallel diagonal stripes. From top-left to bottom-right, the stripes are blue, red, and grey. The stripes are of varying widths and create a dynamic, modern aesthetic.

2019/2020

Camelia Beltrá García

Introducción

En esta práctica se ha desarrollado un sistema distribuido siguiendo un enfoque B2B que gestiona un numero de sondas situadas en plazas de aparcamiento, que son capaces de detectar el volumen que se está ocupando en dicha plaza y de emitir una luz de color en función de si están ocupadas o libres.

Para realizar esto se ha generado con Java un “Dynamic Web Project” y con la aplicación .Net que se van a explicar más específicamente en cada uno de sus puntos. Cada sonda tiene un fichero que se va leyendo y modificando si fuese necesario que define cada una de estas plazas.

Para no tener problemas de seguridad se ha generado en la aplicación .Net un login donde se hashea la contraseña para evitar ataques de seguridad que pongan en peligro la información, se ha encriptado con un algoritmo simétrico conocido como AES la comunicación entre las sondas y el cliente y por último se han generado logs.

Implementación Java

Como se ha mencionado anteriormente se ha generado un proyecto “Dynamic Web Project” que utiliza un servidor Apache Tomcat con la versión 9 junto con Axis2 (versión 1.6.3).

La estructura del proyecto y el servidor:

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the structure of the 'EjemploServiciosWeb' project, including 'Deployment Descriptor', 'JAX-WS Web Services', 'Java Resources' (with 'src' containing 'ejemplo' and 'Sonda.java'), 'Libraries', 'JavaScript Resources', 'build', and 'EjemploServiciosWeb' (with 'axis2-web', 'META-INF', and 'WEB-INF'). The main editor area shows the 'Overview' tab of the 'Server' configuration. It includes sections for 'General Information' (Server name: Tomcat v9.0 Server at localhost, Host name: localhost, Runtime Environment: Apache Tomcat v9.0), 'Server Locations' (Server path: metadata/plugins/org.eclipse.wst.se, Deploy path: wtpwebapps), and 'Ports' (Tomcat admin port: 9005, HTTP/1.1: 9080, AJP/1.3: 9009). The 'Publishing' and 'Timeouts' sections are also visible.

En este proyecto como podemos ver se ha implementado la clase “Sonda.java”, esta clase es un servicio de web a la que se puede acceder con su ruta correspondiente (si la lanzamos en la misma máquina sería la siguiente:

<http://localhost:9080/EjemploServiciosWeb/services/Sonda?wsdl> si la quisiéramos lanzar desde otra máquina se debería de cambiar la IP).

Respecto a la clase **Sonda.java**, se han implementado los getters y los setters requeridos para poder acceder a ellos por la aplicación del cliente. Por ejemplo, podemos ver el ejemplo de *getVolumen*:

```
public String getVolumen(String nombre, String usuario, String ip){
    try {
        String nombreDecrypt = decrypt(nombre,k);
        String usuarioDecrypt = decrypt(usuario,k);
        String ipDecrypt = decrypt(ip,k);
        logEscribir(nombreDecrypt, usuarioDecrypt, ipDecrypt, "Operacion: GetVolumen");
        leerFichero(nombreDecrypt);

    } catch (GeneralSecurityException | IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    try {
        return encrypt("" + Volumen, k);
    } catch (InvalidKeyException | UnsupportedEncodingException | NoSuchAlgorithmException | NoSuchPaddingException
        | InvalidAlgorithmParameterException | IllegalBlockSizeException | BadPaddingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}
```

Como se comentó en el apartado de introducción se le ha metido seguridad a la aplicación, por ello que lo primero que hacemos es desencriptar los valores que vienen por parámetro, más tarde guardamos lo que estamos realizando en el log de dicha sonda y se lee el fichero para tener todos los valores de la sonda, y ya por último devolvemos el valor del volumen encriptado.

Este proceso se ha hecho de la misma forma en los siguientes métodos/ funciones: *getVolumen(parámetros)*, *getUltimaFecha(parámetros)*, *getLed(parámetros)* y *setLed(parámetros)* → este último se diferencia más porque lo que se hace es modificar el fichero en dónde se encuentran los datos de la sonda, con el nuevo dato.

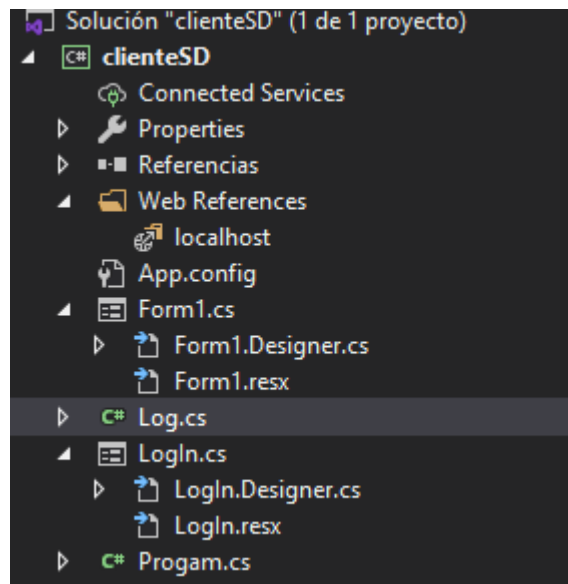
En esta clase también nos encontramos con otros métodos como los de encriptación, y los que se usan para el tratamiento de ficheros (tanto los ficheros para las sondas como para los logs).

```
public boolean nuevaSonda(String nombre, String usuario, String ip) {
    try {
        String nombreDecrypt = decrypt(nombre,k);
        String usuarioDecrypt = decrypt(usuario,k);
        String ipDecrypt = decrypt(ip,k);
        logEscribir(nombreDecrypt, usuarioDecrypt, ipDecrypt, "Operacion: Crear nueva Sonda");
        File file = new File(fichero + "sondas/" + nombreDecrypt + ".txt");
        if(file.exists()){
            leerFichero(usuarioDecrypt);
        }
        else {
            Volumen = randomVolumen();
            Led = randomLed();
            try{
                file.createNewFile();
                UltimaFecha = getFecha();
                String datos = "Volumen=" + Volumen + "\n" +
                    "Led=" + Led + "\n" +
                    "UltimaFecha=" + UltimaFecha + "\n";
                Writer writer = new BufferedWriter(new FileWriter(fichero + "sondas/" + nombreDecrypt + ".txt", true));
                writer.write(datos);
                writer.close();
            } catch (Exception e){
                e.printStackTrace();
            }
        }
        return true;
    } catch (GeneralSecurityException | IOException e1) {
        // TODO Auto-generated catch block
    }
}
```

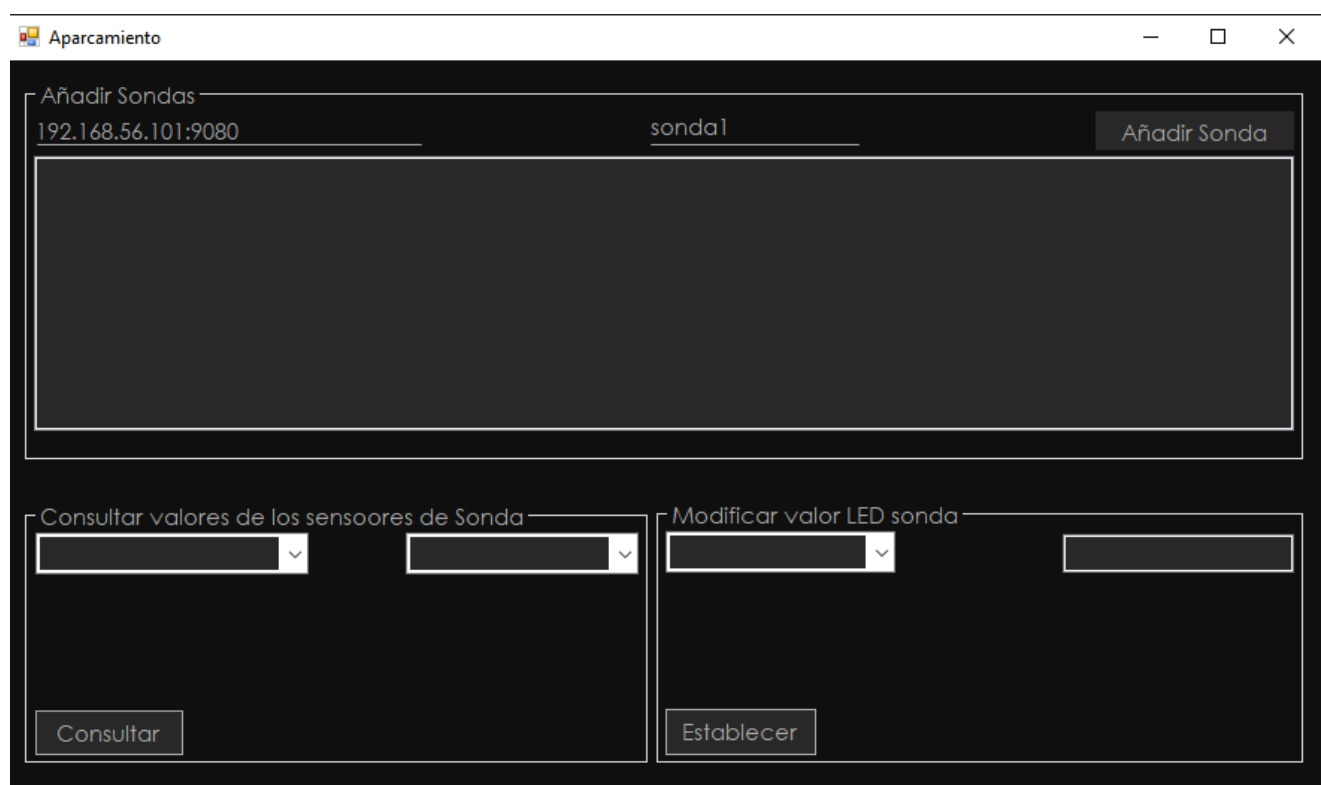
La imagen anterior muestra el como se añade la sonda, en el caso de que no existiera la sonda, se crearía una nueva con valores aleatorios.

Implementación .Net

La aplicación .Net, es la parte del cliente de todo este sistema, para acceder a el se ha generado un proyecto con la siguiente estructura:

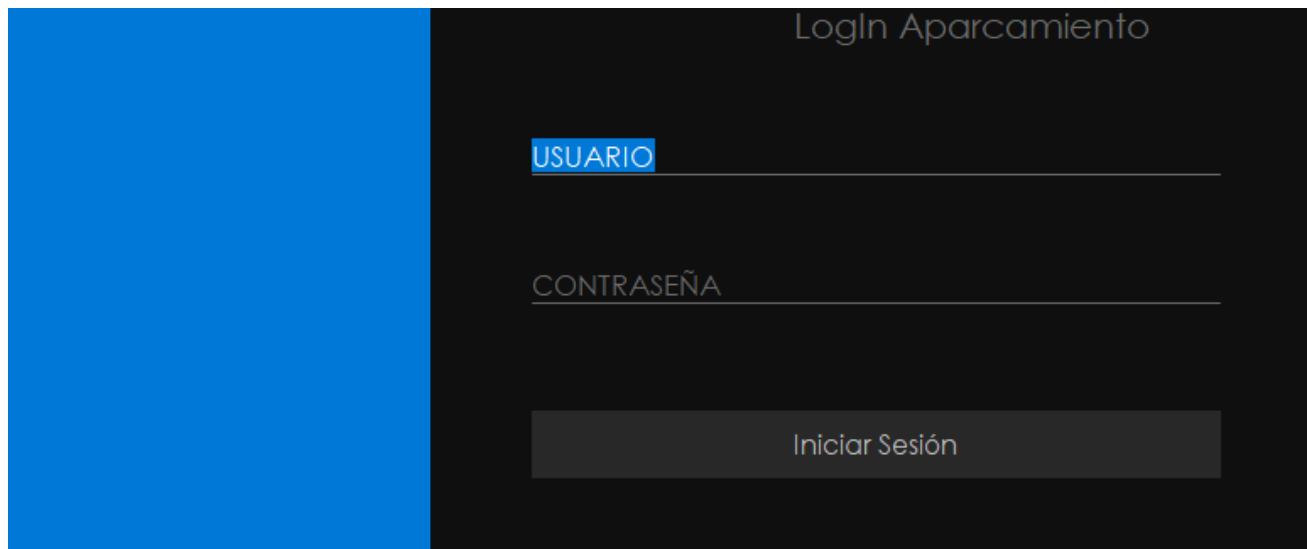


Como podemos ver, en *Referencias* nos encontramos con que tenemos una a referencia a web, esta referencia es para poder a acceder a los servicios que se han generado en la implementación de java (el servicio marcado por Sonda?wsdl). Esta referencia se encuentra en el “Form1.cs”, en este formulario nos encontramos con lo siguiente:



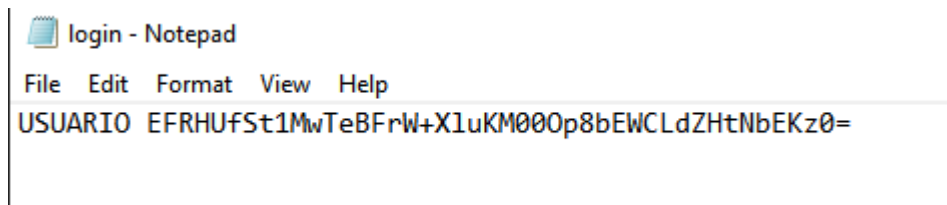
Como podemos ver, es una aplicación que nos permite añadir sondas nuevas, que se verían en el cuadrado siguiente, y se puede consultar a los valores de los sensores (Volumen, Led y la UltimaFecha) como vimos anteriormente, aparte de poder mostrarnos la fecha actual, y por último nos permite elegir una sonda y modificarle el valor de la Led de dicha sonda.

Pero para poder ver dicha pantalla lo primero que tendremos que hacer es logearnos en el sistema, por lo que la primera ventana que se verá será la siguiente:

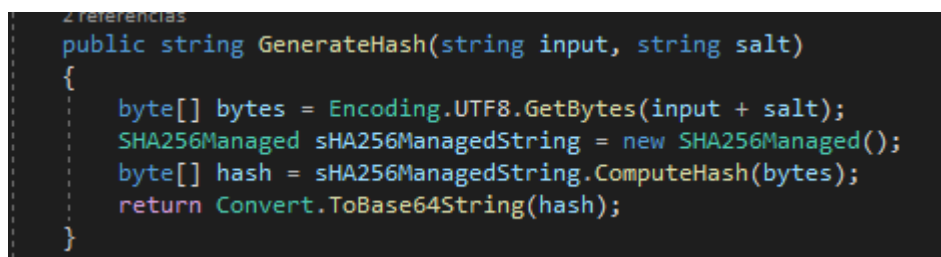


Para acceder deberemos de poner un usuario y una contraseña ya existentes.

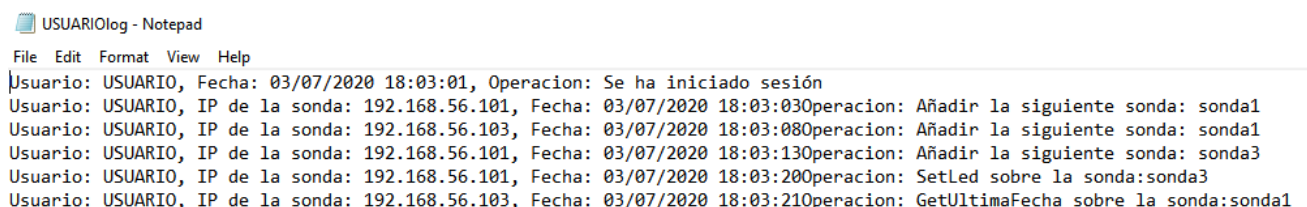
Se creó un usuario con los valores que se ven en pantalla, y se ha guardado en un fichero, que es el siguiente:



Como podemos ver, es un fichero de texto con la password hasheada de esta forma:



Una vez logeados, cada operación que realicemos se escribirá en un fichero de texto (tanto en la aplicación java como en la de .Net) que es el Log, y esto se guardará de esta forma:



Seguridad

Como se ha ido comentando en los anteriores puntos, se ha hecho la opción de la seguridad, para ello hemos utilizado el **algoritmo simétrico AES** para el paso de mensajes (es decir, la comunicación entre java y .Net). Podemos ver en las siguientes imágenes las funciones utilizadas:

Este algoritmo es un esquema de cifrado por bloques adoptado como un estándar de cifrado que tiene un tamaño de bloque fijo de 128 bits.

Como hemos mencionado es un algoritmo simétrico que se basa en cifrar con una clave secreta, la clave secreta que he usado ha sido la siguiente:

```
static string KEY = "abewqlo1469kstq1";
```

Que es compartida tanto por el emisor como por el receptor para poder cifrar y descifrar los mensajes.

Por ejemplo, en java para descifrar hemos hecho uso de las siguientes funciones:

```
public byte[] decrypt(byte[] cipherText, byte[] key, byte [] initialVector) throws NoSuchAlgorithmException, NoSuchPaddingException {
    Cipher cipher = Cipher.getInstance(cipherTransformation);
    SecretKeySpec secretKeySpec = new SecretKeySpec(key, aesEncryptionAlgorithm);
    IvParameterSpec ivParameterSpec = new IvParameterSpec(initialVector);
    cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivParameterSpec);
    cipherText = cipher.doFinal(cipherText);
    return cipherText;
}

private byte[] getKeyBytes(String key) throws UnsupportedEncodingException{
    byte[] keyBytes= new byte[16];
    byte[] parameterKeyBytes= key.getBytes(characterEncoding);
    System.arraycopy(parameterKeyBytes, 0, keyBytes, 0, Math.min(parameterKeyBytes.length, keyBytes.length));
    return keyBytes;
}

public String decrypt(String encryptedText, String key) throws KeyException, GeneralSecurityException, GeneralSecurityException {
    byte[] cipheredBytes = Base64.getDecoder().decode(encryptedText);
    byte[] keyBytes = getKeyBytes(key);
    return new String(decrypt(cipheredBytes, keyBytes, keyBytes), characterEncoding);
}
```

Y en c# por ejemplo, para encriptar sería lo siguiente:

```
16referencias
public static String doEncryptAES(String plainText, String key)
{
    var plainBytes = Encoding.UTF8.GetBytes(plainText);
    return Convert.ToBase64String(Encrypt(plainBytes, getRijndaelManaged(key)));
}

private static RijndaelManaged getRijndaelManaged(String secretKey)
{
    var keyBytes = new byte[16];
    var secretKeyBytes = Encoding.UTF8.GetBytes(secretKey);
    Array.Copy(secretKeyBytes, keyBytes, Math.Min(keyBytes.Length, secretKeyBytes.Length));
    return new RijndaelManaged
    {
        Mode = CipherMode.CBC,
        Padding = PaddingMode.PKCS7,
        KeySize = 128,
        BlockSize = 128,
        Key = keyBytes,
        IV = keyBytes
    };
}
```

```
private static byte[] Encrypt(byte[] plainBytes, RijndaelManaged rijndaelManaged)
{
    return rijndaelManaged.CreateEncryptor()
        .TransformFinalBlock(plainBytes, 0, plainBytes.Length);
}
```

Se ha realizado de forma que son compatibles, y se puede pasar de una buena forma la información.

Otro punto a destacar de la seguridad de la aplicación también ha sido comentado, se han generado **Logs**, como ya se ha visto el que se vería en el cliente, mostramos el que se genera para cada sonda:

```
sonda1log.txt
~/eclipse-workspace/.metadata/.plugins/org...ore/tmp0/wtpwebapps/EjemploServiciosWeb/log
Save
Usuario: USUARIO, IP: 192.168.56.103, Tiempo UTC de la operacion: 2020-07-03 16:53:32, Operacion:
Crear nueva Sonda
Usuario: USUARIO, IP: 192.168.56.103, Tiempo UTC de la operacion: 2020-07-03 16:53:40, Operacion:
GetVolumen
Usuario: USUARIO, IP: 192.168.56.103, Tiempo UTC de la operacion: 2020-07-03 18:03:09, Operacion:
Crear nueva Sonda
Usuario: USUARIO, IP: 192.168.56.103, Tiempo UTC de la operacion: 2020-07-03 18:03:22, Operacion:
GetUltimaFecha
```

Estos logs, se nos facilitarían el registro de auditoría.

Y, por último, para el LogIn se ha **hasheado con “sal”**, en criptografía una sal son datos aleatorios que se usan como entrada adicional a una función unidireccional que codifica datos, como en este caso lo que se codifica es la contraseña.

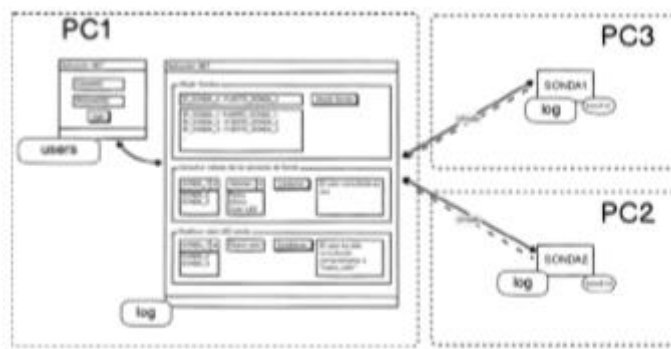
En mi caso la sal toma el siguiente valor:

```
string salt = "emkF9f1uQoQ22BqtMA0tbBHqib56mqkDlIxqYcKIwZa3AAlL646MZWw7taWgEV27k3gP1NO";
```

Gracias a estos tres puntos se consigue una mejoría en la seguridad de la aplicación.

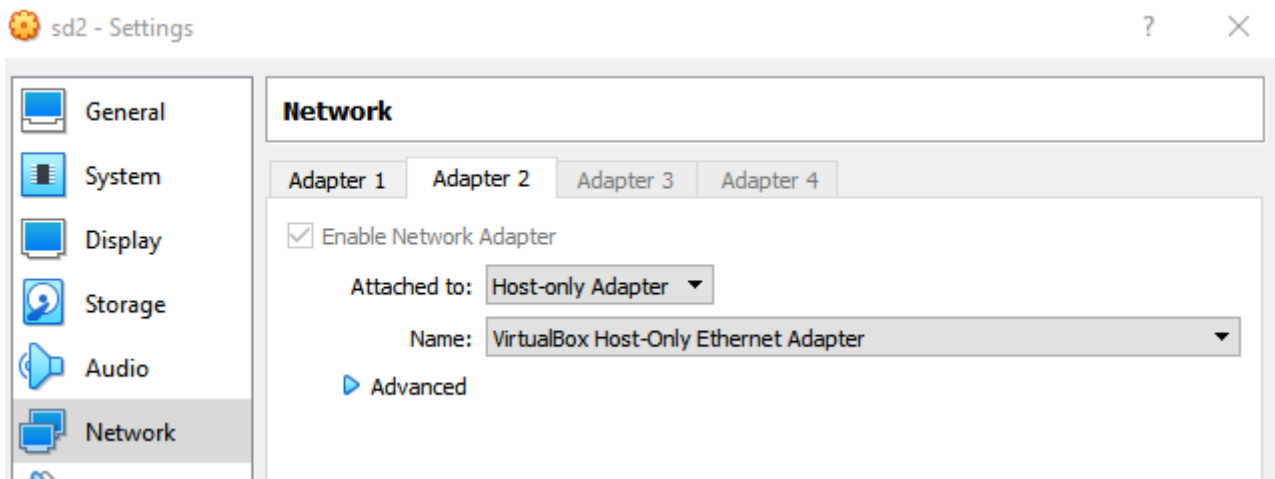
Despliegue

Se va a mostrar cómo se ha conseguido el siguiente despliegue y como realizarlo:



Lo primero se han usado para representar el PC2 y el PC3, dos máquinas virtuales con el sistema operativo ubuntu.

Para recrear la LAN que existe en las clases de la politécnica hemos modificado los ajustes de estas maquinas virtuales de la siguiente forma:



Una vez cambiados los ajustes de conexión podemos abrir las máquinas virtuales, cada una con el proyecto “EjemploServiciosWeb”, con la máquina virtual de Java, Apache Tomcat V9.0.27, Axis2 1.6.3 y Eclipse instalados.

Una vez tengamos todo instalado, se le damos a “Start” al servidor apache de cada máquina, una vez realizado esto, podemos acceder al servicio Sonda?wsdl.

En el PC1 va a ser el ordenador que tiene las máquinas virtuales y dicho ordenador ha de tener instalado como sistema operativo Windows 10. Podemos ejecutar la aplicación desde el ejecutable “clienteSD”, nos logeamos y añadimos la sonda que queremos añadir.