**Server Socket Request**

**Author: *Jean-Pierre Erasmus***

**Last Updated: *13/04/2017***

**Assignment Brief:**

*“*

*Please write a Java program that shows that you understand the following instructions:*

*There is already a server at IP address 196.37.22.179 listening on port 9011. You do not have to write the server portion of this demo. Your program must create a socket connection to the IP address and port mentioned above. Remember, Blue Label has already written the server portion of this demo.*

*Send the following XML message to the socket, Please note there is no security, no certificates, just a plan TCP/IP connection. The request message is formatted below. Please include a README.txt file where you saved what you have received from the server. Without this file, your assignment will not be evaluated.*

*<request> <EventType>Authentication</EventType> <event> <UserPin>12345</UserPin> <DeviceId>12345</DeviceId> <DeviceSer>ABCDE</DeviceSer> <DeviceVer>ABCDE</DeviceVer> <TransType>Users</TransType> </event> </request>*

*Display the message that you receive back from the server.*

*“*

**Assignment Solution:**

The basic solution to the above requested assignment and technical test is to open a simple JAVA socket to the IP and Port number. Once a socket connection is acquired, a writer may be instantiated to send the above mentioned XML string to the socket. A reader is instantiated to read the response back from the server. Once the response is received, the response message may be displayed to the user and or caller.

**Detail Solution Explained:**

Using the request details provided in the specification a XML document is build, and then transformed into a XML formatted string for transmitting.

The javax.xml library is used to transform the above data into a proper XML document object.

The javax.xml library is once again used to transform the XML object into a XML formatted string, either as a simple string or a Human Readable formatted string.

The java.net library is used to create a simple socket connection to the above mentioned IP number and port number. Once the connection is established a print writer for sending and a buffered reader for receiving the response is created over the socket connection.

We can now send the request using a “println” including end of line character and perform a “readline” to receive the response back from the server.

Once the response message is received we may format it again as Human Readable text as well as displaying it to the user and or saving it to an external file for perusal after the fact as requested.

The java.io library is used to create a file writer to output the response message to the README.txt file.

The response is sent to the console as Human Readable text using “System.out”.

**Embellished Assignment Solution:**

The embellished solution is based around a very simple Java Web project, using a simple client (web page) and servlet (backend) structure.

The servlet uses a single process request method to process get and post requests. The basics of the servlet is based around the simple solution to the assignment and I refer to the above paragraph for the basic explanation of these. The only difference and or extra processing done in the servlet is retrieving the request parameters as well as creating a file object to print the path of the readme file.

The frontend for the web project is a single index.html file accompanied by the mainCss.css style sheet and the requestJS.js JavaScript file. The web page is a simple structure, with inputs for some of the request data and server details, a big red button to submit the request to the servlet and an output box to display the response received from the server.

I used CSS3 and the web kit “utilities” to create a look of a corkboard with post it notes on it, a broken submit plate hanging on one screw with a big red pulsating rotating button on it for submission of the request. I used CSS3 and the web kit again to create the look of an old CRT green screen where the output is displayed as the request returns the response.

3D effects are created using CSS shadowing. Minimal images are used, only were it is really needed did I use images.

The user is able to hover over the post it notes, triggering effects as well as altering some of the data. Some data does not allow altering as this displays the ability to make fields read only all though this is trivial.

The button is created using two buttons layered on top of each other, so that the top butto may spin, but the shadow stay static.

**Conclusion:**

Thank you very much for the opportunity to complete the above assignment. The QR code and 64 bit encryption was a nice addition to starting the project. I enjoyed playing around with the CSS3 to create a unique interface for this solution. Please find below a copy of the simple solution as well as the output.

Jean-Pierre Erasmus

13/04/2017

**JavaXmlSocket.java**

package javaxmlsocket;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.io.StringReader;

import java.io.StringWriter;

import java.net.Socket;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.DOMException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.xml.sax.InputSource;

import org.xml.sax.SAXException;

/\*\*

\*

\* @author Jean-Pierre Erasmus

\*/

public class JavaXmlSocket {

/\*\*

\*

\* @param args

\*/

public static void main(String[] args) {

//Get Socket connection string

String xmlSocketAuthString = buildXMLAuthString("Authentication",

"12345",

"12345",

"ABCDE",

"ABCDE",

"Users",

false);

String xmlSocketAuthStringHuman = buildXMLAuthString("Authentication",

"12345",

"12345",

"ABCDE",

"ABCDE",

"Users",

true);

//Print String

System.out.println("Socket XML String :\n" + xmlSocketAuthString);

System.out.println("Socket XML String Formatted :\n" + xmlSocketAuthStringHuman);

//Create socket connection and print request response

try (

Socket sendSocket = new Socket("196.37.22.179", 9011);

PrintWriter out = new PrintWriter(sendSocket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(sendSocket.getInputStream()));) {

//Send request

out.println(xmlSocketAuthString);

//Get Response

String resultString = in.readLine();

//Display respone

System.out.println("Result String: " + resultString);

System.out.println(outputXMLFormatter(resultString, null, true));

//Write response

writeReadme(outputXMLFormatter(resultString, null, true));

} catch (Exception e) {

System.err.println("Exception " + e.getMessage());

System.exit(1);

}

}

/\*\*

\*

\* @param responseToWrite

\*/

private static void writeReadme(String responseToWrite) {

//Create new Readme file with response

try (FileWriter filewriter = new FileWriter("README.txt",false);

BufferedWriter buffwriter = new BufferedWriter(filewriter); ) {

//Output Response String to output

buffwriter.write(responseToWrite);

} catch (IOException e) {

System.out.println("Readme.txt Write Exception " + e.getMessage());

}

}

/\*\*

\*

\* @return

\*/

private static String buildXMLAuthString(String eventType,

String UserPin,

String deviceID,

String deviceSerial,

String version,

String transactionType,

boolean indentOutput) {

String xmlOutput = "";

DocumentBuilderFactory xmlFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder xmlBuilder;

try {

//Create XML Document

xmlBuilder = xmlFactory.newDocumentBuilder();

Document xmlDoc = xmlBuilder.newDocument();

//Create request node

Element mainRootElement = xmlDoc.createElement("request");

xmlDoc.appendChild(mainRootElement);

//Create event type node

Element eventTypeElement = xmlDoc.createElement("EventType");

eventTypeElement.appendChild(xmlDoc.createTextNode(eventType));

mainRootElement.appendChild(eventTypeElement);

//Create event Node

Element eventNode = xmlDoc.createElement("event");

//Event Detail nodes

//User Pin

Element userPinNode = xmlDoc.createElement("UserPin");

userPinNode.appendChild(xmlDoc.createTextNode(UserPin));

eventNode.appendChild(userPinNode);

//Device ID

Element deviceIDNode = xmlDoc.createElement("DeviceId");

deviceIDNode.appendChild(xmlDoc.createTextNode(deviceID));

eventNode.appendChild(deviceIDNode);

//Device Serial

Element deviceSerialNode = xmlDoc.createElement("DeviceSer");

deviceSerialNode.appendChild(xmlDoc.createTextNode(deviceSerial));

eventNode.appendChild(deviceSerialNode);

//Device Version

Element deviceVersionNode = xmlDoc.createElement("DeviceVer");

deviceVersionNode.appendChild(xmlDoc.createTextNode(version));

eventNode.appendChild(deviceVersionNode);

//Transaction Type

Element tranTypeNode = xmlDoc.createElement("TransType");

tranTypeNode.appendChild(xmlDoc.createTextNode(transactionType));

eventNode.appendChild(tranTypeNode);

//Add event to request

mainRootElement.appendChild(eventNode);

// output XML to String

return outputXMLFormatter(null, xmlDoc, indentOutput);

} catch (ParserConfigurationException | DOMException | SAXException | IOException | TransformerException e) {

System.out.println("buildXMLAuthString() Exception : 1001 : " + e.getMessage());

}

return xmlOutput;

}

/\*\*

\*

\* @param xmlString

\* @param xmlInDocument

\* @param indentXML

\* @return

\* @throws SAXException

\* @throws IOException

\* @throws ParserConfigurationException

\* @throws TransformerConfigurationException

\* @throws TransformerException

\*/

private static String outputXMLFormatter(String xmlString,

Document xmlInDocument,

boolean indentXML) throws SAXException, IOException, ParserConfigurationException, TransformerConfigurationException, TransformerException {

String formattedXMLOutput = "";

//Cater for Strings or Documents

if (xmlString != null) {

//Create XML Document from string

DocumentBuilderFactory docfactory = DocumentBuilderFactory.newInstance();

DocumentBuilder docBuiler = docfactory.newDocumentBuilder();

InputSource inSource = new InputSource(new StringReader(xmlString));

xmlInDocument = docBuiler.parse(inSource);

} else if (xmlInDocument != null) {

//Process the input document

} else {

//No XML to formated

return "";

}

//Use Transformer to format the XML Document.

//output XML to String

Transformer transformer = TransformerFactory.newInstance().newTransformer();

//Set indentation

if (indentXML) { //dependant on human readability requirements

transformer.setOutputProperty(OutputKeys.INDENT, "yes");

} else {

transformer.setOutputProperty(OutputKeys.INDENT, "no");

}

//Create formatted String as output

StreamResult result = new StreamResult(new StringWriter());

DOMSource source = new DOMSource(xmlInDocument);

transformer.transform(source, result);

//Return formatted XML String

return result.getWriter().toString();

}

}

**Socket XML String :**

<?xml version="1.0" encoding="UTF-8" standalone="no"?><request><EventType>Authentication</EventType><event><UserPin>12345</UserPin><DeviceId>12345</DeviceId><DeviceSer>ABCDE</DeviceSer><DeviceVer>ABCDE</DeviceVer><TransType>Users</TransType></event></request>

Socket XML String Formatted :

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<request>

<EventType>Authentication</EventType>

<event>

<UserPin>12345</UserPin>

<DeviceId>12345</DeviceId>

<DeviceSer>ABCDE</DeviceSer>

<DeviceVer>ABCDE</DeviceVer>

<TransType>Users</TransType>

</event>

</request>

**Result String:**

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<response>

<SessionId/>

<EventType>Authentication</EventType>

<event>

<EventCode>1</EventCode>

</event>

<data>

<ErrorCode>3</ErrorCode>

<ErrorText>Invalid user</ErrorText>

<AeonErrorCode>103</AeonErrorCode>

<AeonErrorText>Invalid User</AeonErrorText>

</data>

</response>