A
PROJECT REPORT
ON

<u>RELIABLE PACKET TRANSFER USING UDP IN JAVA</u>

FOR
ENTS 640 NETWORKS AND PROTOCOLS I

SUBMITTED TO
UNIVERSITY OF MARYLAND, COLLEGE PARK

SUBMITTED BY
ASHUTOSH VERULE (UID: 114880994)
AJIT YADAV (UID: 115090590)

(BATCH 2016-18)
FALL 2016

CONTENTS

PROBLEM STATEMENT

Write a Java application consisting of a client and a server using Java's UDP sockets. The reliability of the communication has to be ensured by the application using UDP's unreliable data transfer services. The protocol will use integrity check, timeout and retransmission as needed for this purpose. The client will send a measurement ID to the server, and the server will respond by sending the temperature measurement values (in degrees Fahrenheit) corresponding to the measurement ID back to the client.

The messages sent between the client and the server will be text-oriented and consist of human-readable character sequences. Thus, they will need to be assembled as String objects and then converted to byte sequences before sending them out through the UDP sockets.

Data File:

The measurement IDs and the corresponding measurement values can be found in the data file. The data file contains 100 lines, and each line represents a separate measurement ID/temperature value pair. The server should use the content of this file when serving the client's request.

Client's Message:

The client's request message (except for the checksum) is enclosed within the <request> opening and </request> closing tag pairs. Inside, there should be two more message elements: the request ID, enclosed within the <id> and </id> tag pairs, and the measurement ID, enclosed within the <measurement> and </measurement> tags.

Server's Message:

The server's response message (except for the checksum) is enclosed within the <response> opening and </response> closing tag pairs. If there are no errors (normal response), there should be four more message elements inside (in this order): the request ID, enclosed within the <id> and </id> tag pairs, the response code, enclosed within the <code> and </code> tags, the measurement ID, enclosed within the <measurement> and </measurement> tags, and finally, the temperature value, enclosed within the <value> and </value> tags. The request ID should be the same as the one received from the client with the request message. The response code should indicate the outcome of the received request, taking up values according to the table below:

Response Code	Meaning
0	OK. The response has been created according to the request.
1	Error: integrity check failure. The request has one or more bit errors.
2	Error: malformed request. The syntax of the request message is not correct.
3	Error: non-existent measurement. The measurement with the requested measurement ID does not exist.

If some error has occurred, the response message should only contain the request ID and the response code elements, and the response code should indicate the problem encountered based on the response code table above. The checksum should also be included at the end.

Checksum:

The last message element is the checksum, which is a 16-bit unsigned integer, and its role is to provide bit error detection capability. The checksum should be calculated over all characters in the request message (except for the checksum), starting with the "<" character of the opening <request> tag, and ending with the ">" character of the closing </request> tag.

Protocol Operation:

Set up the client and the server UDP socket parameters (IP addresses, port numbers etc.) so that the two sides could communicate with each other.

Client application:

1. It should select one of the available measurement IDs to receive the corresponding temperature measurement value from the server.
2. It should generate a random request ID, assemble the request message as a sequence of characters (as a String object), including the integrity check field, convert it into a byte array and send it to the server. It should also start a timer at this point with initial timeout value of 1 second.
3. It should wait for the response from the server. If no response arrives and the timer expires, it should resend the request, and the timeout interval should be doubled at each timeout event. After the 4th timeout event, the transmitter should declare communication failure and print an error message on the screen. If the response arrives after a number of timeout events and retransmissions, the timeout value and the timeout counter should be reset to their initial values.
4. It should calculate the integrity check value for the response message and compare it with the integrity check value supplied in the response message. If they are not equal, it should go back to step 2 and repeat sending the request message.
5. It should check the value of the response code. If no errors occurred (i.e. if it is 0), then it should display the measurement value received from the server on the screen. If some errors occurred, it should display an error message on the screen indicating the problem that has occurred (e.g. malformed request message). If the error was that the request message's integrity check failed (response code 1), it should ask the user if he/she would like to resend the request and go back to step 2 and resend the request if needed.
6. It should go back to step 1 and do the whole request/response sequence in a loop until the application is terminated.

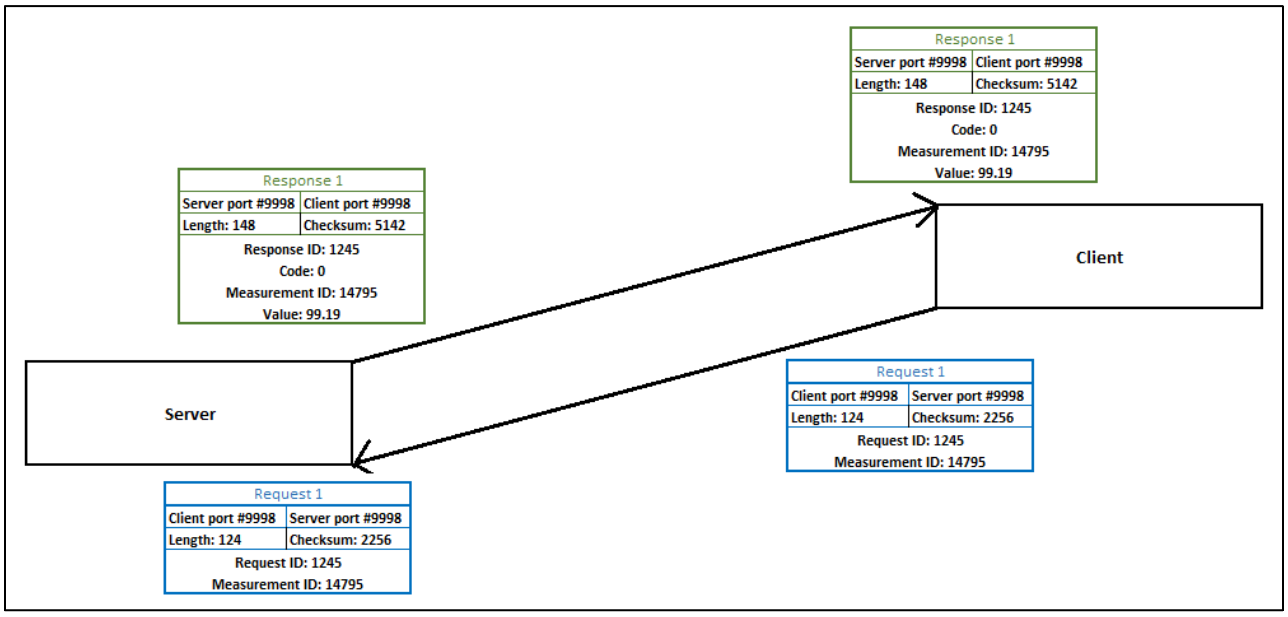
Server application:

1. It should wait for requests on the specified port.
2. Upon reception of a request message, it should verify the integrity of the message by calculating the integrity check value and comparing it with the integrity check value provided in the request message. If they are different, it should send back an error response with response code 1.
3. If the integrity check passes, it should convert the received message into a character sequence (a String object) and check the syntax of the request message. This should include checking that all opening and closing tags are present (no misspelling or invalid characters) and they are in the right order. The syntax of the element values should also be checked, e.g. that the number between the <measurement> and the </measurement> tags is a 16-bit unsigned integer. White space in the message should be ignored during this step. If the syntax check fails, it should send back an error response with response code 2.

4. It should look up the measurement value that corresponds to the requested measurement ID, as defined in the data.txt file. If there is no measurement value for the requested measurement ID, it should send back an error response with response code 3.
5. It should assemble the response message with response code 0 including the requested measurement value and the integrity check value, convert it into a byte array and send the response back to the client.
6. It should go back to step 1 and wait for another request message

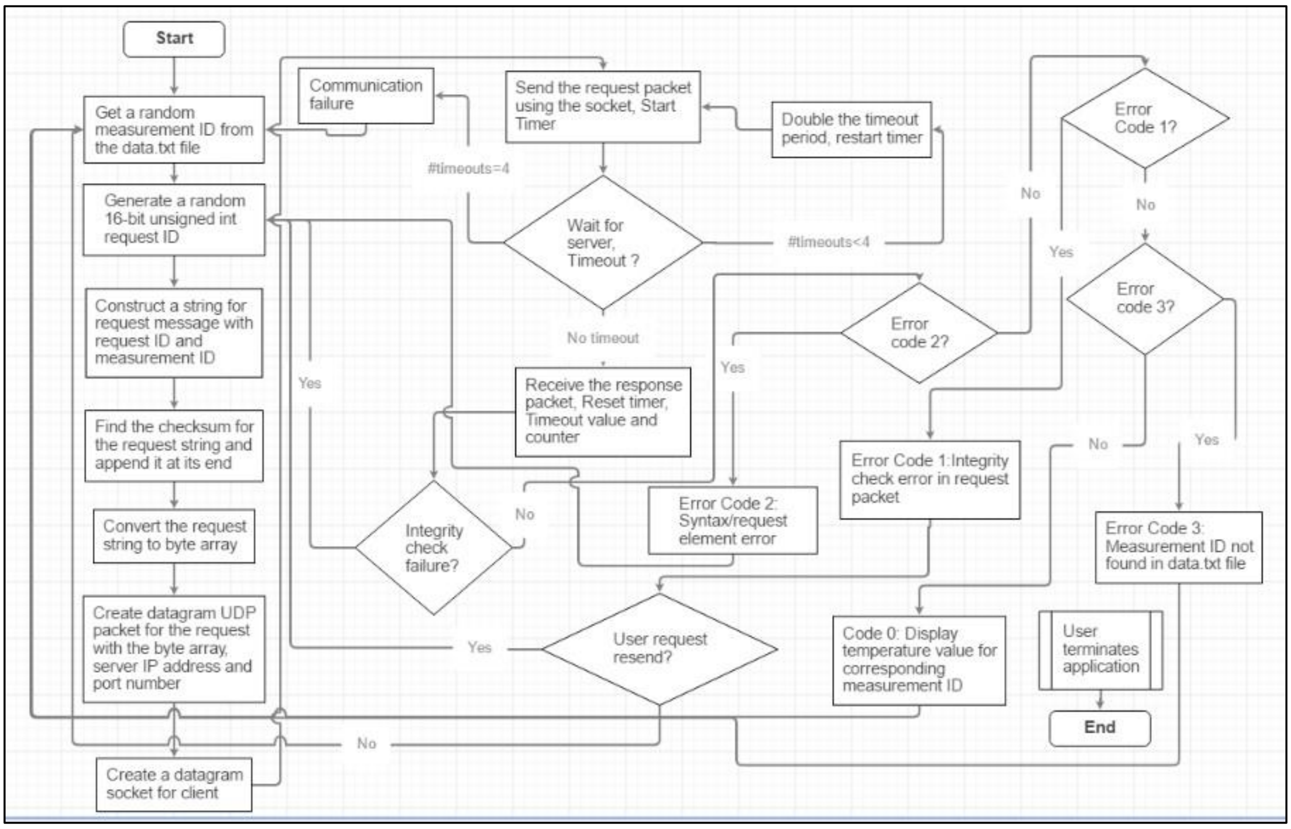
BLOCK DIAGRAM

The application can be depicted as shown below:

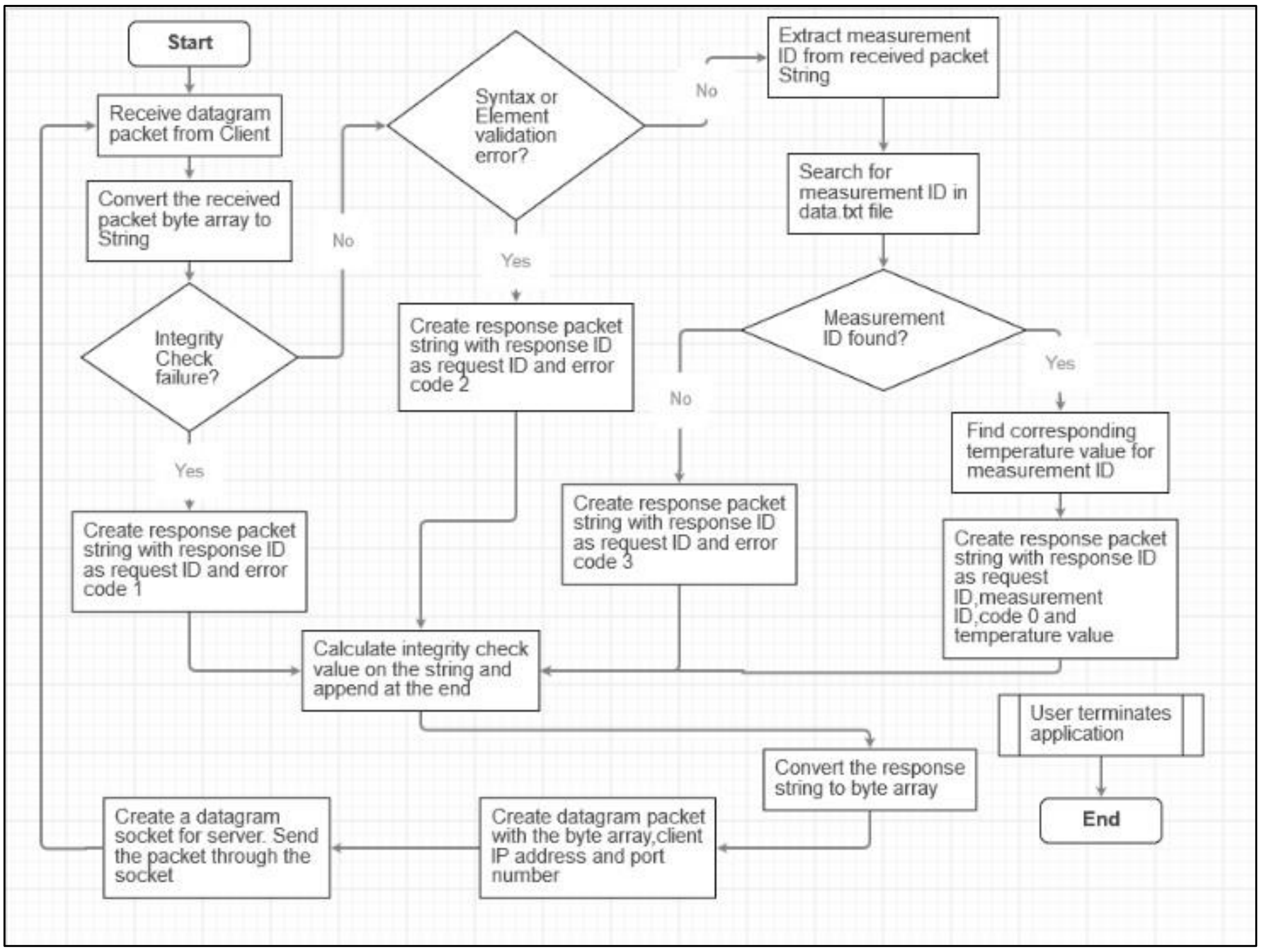


FLOW CHARTS

Client-side Flowchart:


Start
Get a random measurement ID from the data.txt file
Generate a random 16-bit unsigned int request ID
Construct a string for request message with request ID and measurement ID
Find the checksum for the request string and append it at its end
Convert the request string to byte array
Create datagram UDP packet for the request with the byte array, server IP address and port number
Create a datagram socket for client
Communication failure
Send the request packet using the socket, Start Timer
Double the timeout period, restart timer
#timeouts=4
Wait for server, Timeout ?
#timeouts<4
No timeout
Receive the response packet, Reset timer, Timeout value and counter
Yes
Integrity check failure?
No
Error code 2?
Yes
Error Code 2: Syntax/request element error
Error Code 1:Integrity check error in request packet
Error Code 1?
No
Yes
No
Error code 3?
No
Yes
Error Code 3: Measurement ID not found in data.txt file
User request resend?
Yes
No
Code 0: Display temperature value for corresponding measurement ID
User terminates application
End

Server-side Flowchart:


Start
Receive datagram packet from Client
Convert the received packet byte array to String
Integrity Check failure?
Yes
No
Create response packet string with response ID as request ID and error code 1
Syntax or Element validation error?
Yes
No
Create response packet string with response ID as request ID and error code 2
Extract measurement ID from received packet String
Search for measurement ID in data.txt file
Measurement ID found?
No
Yes
Create response packet string with response ID as request ID and error code 3
Find corresponding temperature value for measurement ID
Create response packet string with response ID as request ID,measurement ID,code 0 and temperature value
Calculate integrity check value on the string and append at the end
Convert the response string to byte array
Create datagram packet with the byte array,client IP address and port number
Create a datagram socket for server. Send the packet through the socket
User terminates application
End

UML CLASS DIAGRAMS

Client:

ProjectClient
+ SERVER_PORT_NUM : int-final
+ TIME_OUT_VALUE : int
+ serverIP : InetAddress
+ requestID : int
+ randomMeasurementID : int
+ timeOutIteration : int
+ packetToSend : String
+ sentMessage : byte[]
+ sentPacket : DatagramPacket
+ clientSocket : DatagramSocket
+ receivedMessage : byte[]
+ receivedPacket : DatagramPacket
+ receivedMsg : String
+ tempSplit : String[]
+ myScanner : Scanner
+ userInput : String
+ getMeasurementID() : int-static
+ integrityCheck(checkStr : String) : String-static

Server:

ProjectServer
+ SERVER_PORT_NUM : int-final
+ receivedMessage : byte[]
+ receivePacket : DatagramPacket
+ serverSocket : DatagramSocket
+ tempSplit : String[]
+ sentMessage : String
+ clientAddress : InetAddress
+ clientPort : int
+ receivedMsg : String
+ requestID : String
+ measurementValue : int
+ measurementIndex : int
+ temperatureFile : ArrayList<Double>
+ sentMessageByte : byte[]
+ sentPacket : DatagramPacket
+ integrityCheck(checkStr : String) : String-static
+ findMeasurementID(measurementValue : int) : int
+ getTemperature() : ArrayList<Double>

OUTPUT

Following snippets of few examples illustrate the performance of the JAVA code in various scenarios:

Example 1:

Successful transmission of request from Client and successful reception of response from the Server.

Client-side Output:

```
The data sent to server is:
<request>
	<id>3137</id>
	<measurement>20172</measurement>
</request>13572

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>3137</id>
	<code>0</code>
	<measurement>20172</measurement>
	<value>98.07</value>
</response>9562
Temperature value: 98.07 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
	<id>3137</id>
	<measurement>20172</measurement>
</request>13572
Message to send:
<response>
	<id>3137</id>
	<code>0</code>
	<measurement>20172</measurement>
	<value>98.07</value>
</response>9562

Sending the response to the client...
```

Example 2:

Integrity Check failure on the request packet from Client. User chooses to resend the packet.

Client-side Output:

```
The data sent to server is:
<request>
	<id>51851</id>
	<measurement>17765</measurement>
</request>3704
Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>51851</id>
	<code>1</code>
</response>60424
Error: integrity check failure. The request has one or more bit errors.
Do you want to resend the packet(Y/N):
y
The data sent to server is:
<request>
	<id>59806</id>
	<measurement>17765</measurement>
</request>15907

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>59806</id>
	<code>0</code>
	<measurement>17765</measurement>
	<value>92.08</value>
</response>6958
Temperature value: 92.08 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
	<id>51851</id>
	<measurement>17765</measurement>
</request>37046
Message to send:
<response>
	<id>51851</id>
	<code>1</code>
</response>60424

Sending the response to the client...
Receiving the request from the client...Received packet is:
<request>
	<id>59806</id>
	<measurement>17765</measurement>
</request>15907
Message to send:
<response>
```

```
        <id>59806</id>
        <code>0</code>
        <measurement>17765</measurement>
        <value>92.08</value>
</response>6958

Sending the response to the client...
```

Example 3:

Integrity Check failure on the request packet from Client. User chooses not to resend the packet.

Client-side Output:

```
The data sent to server is:
<request>
        <id>8861</id>
        <measurement>16047</measurement>
</request>28045

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
        <id>8861</id>
        <code>1</code>
</response>27266
Error: integrity check failure. The request has one or more bit errors.
Do you want to resend the packet(Y/N):
n
The data sent to server is:
<request>
        <id>60181</id>
        <measurement>29850</measurement>
</request>36681

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
        <id>60181</id>
        <code>0</code>
        <measurement>29850</measurement>
        <value>68.27</value>
</response>32720
Temperature value: 68.27 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
        <id>8861</id>
        <measurement>16047</measurement>
</request>280458
Message to send:
<response>
        <id>8861</id>
        <code>1</code>
</response>27266
```

```
Sending the response to the client...
Receiving the request from the client...Received packet is:
<request>
	<id>60181</id>
	<measurement>29850</measurement>
</request>36681
Message to send:
<response>
	<id>60181</id>
	<code>0</code>
	<measurement>29850</measurement>
	<value>68.27</value>
</response>32720

Sending the response to the client...
```

Example 4:

Client experiences few timeouts (less than four) before successful reception of packet from Server.

Client-side Output:

```
The data sent to server is:
<request>
	<id>17078</id>
	<measurement>18400</measurement>
</request>8166
Sending the request to the server...
Receiving the response from the server...
1. Time-out experienced

Sending the request to the server...The data sent to server is:
<request>
	<id>17078</id>
	<measurement>18400</measurement>
</request>8166

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>17078</id>
	<code>0</code>
	<measurement>18400</measurement>
	<value>65.16</value>
</response>40344
Temperature value: 65.16 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
	<id>17078</id>
	<measurement>18400</measurement>
</request>8166
Message to send:
<response>
```

```
        <id>17078</id>
        <code>0</code>
        <measurement>18400</measurement>
        <value>65.16</value>
</response>40344

Sending the response to the client...
```

Example 5:

Client experiences four timeouts.

<u>Client-side Output</u>:

```
The data sent to server is:
<request>
        <id>5158</id>
        <measurement>65523</measurement>
</request>40877

Sending the request to the server...
Receiving the response from the server...
1. Time-out experienced

Sending the request to the server...The data sent to server is:
<request>
        <id>5158</id>
        <measurement>65523</measurement>
</request>40877
Sending the request to the server...
Receiving the response from the server...
2. Time-out experienced

Sending the request to the server...The data sent to server is:
<request>
        <id>5158</id>
        <measurement>65523</measurement>
</request>40877

Sending the request to the server...
Receiving the response from the server...
3. Time-out experienced

Sending the request to the server...The data sent to server is:
<request>
        <id>5158</id>
        <measurement>65523</measurement>
</request>40877

Sending the request to the server...
Receiving the response from the server...
4. Time-out experienced
Communication Failure
Exception message: Receive timed out

The data sent to server is:
```

```
<request>
	<id>52121</id>
	<measurement>54798</measurement>
</request>38953

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>52121</id>
	<code>0</code>
	<measurement>54798</measurement>
	<value>89.7</value>
</response>50217
Temperature value: 89.7 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
	<id>52121</id>
	<measurement>54798</measurement>
</request>38953
Message to send:
<response>
	<id>52121</id>
	<code>0</code>
	<measurement>54798</measurement>
	<value>89.7</value>
</response>50217

Sending the response to the client...
```

Example 6:

Measurement ID sent in the request is absent in data.txt file, error code 3

Client-side Output:

```
The data sent to server is:
<request>
	<id>4108</id>
	<measurement>1111</measurement>
</request>40884


Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>4108</id>
	<code>3</code>
</response>58026

Error: non-existent measurement. The measurement with the requested measurement ID
does not exist.
The data sent to server is:
<request>
```

```
	<id>10552</id>
	<measurement>19746</measurement>
</request>37664


Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
	<id>10552</id>
	<code>0</code>
	<measurement>19746</measurement>
	<value>73.89</value>
</response>36912

Temperature value: 73.89 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
	<id>4108</id>
	<measurement>1111</measurement>
</request>40884

Message to send:
<response>
	<id>4108</id>
	<code>3</code>
</response>58026

Sending the response to the client...
Receiving the request from the client...Received packet is:
<request>
	<id>10552</id>
	<measurement>19746</measurement>
</request>37664

Message to send:
<response>
	<id>10552</id>
	<code>0</code>
	<measurement>19746</measurement>
	<value>73.89</value>
</response>36912

Sending the response to the client...
```

Example 7:

Server should ignore white spaces in syntax validation

Client-side Output:

```
The data sent to server is:
<request>

             <id>1138</id><measurement>20330</measurement>
</request>26139


Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
       <id>1138</id>
       <code>0</code>
       <measurement>20330</measurement>
       <value>82.82</value>
</response>40115

Temperature value: 82.82 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>

             <id>1138</id><measurement>20330</measurement>
</request>26139

Message to send:
<response>
       <id>1138</id>
       <code>0</code>
       <measurement>20330</measurement>
       <value>82.82</value>
</response>40115

Sending the response to the client...
```

Example 8:

Syntax of the request message is not correct, error code 2

Client-side Output:

```
The data sent to server is:
<request>
       <measurement>11749</measurement>
       <id>21208</id>
</request>47626

Sending the request to the server...
Receiving the response from the server...Received packet is:
<response>
       <id>21208</id>
```

```
        <code>2</code>
</response>55305

Error: malformed request. The syntax of the request message is not correct
The data sent to server is:
<request>
        <id>16289</id>
        <measurement>63353</measurement>
</request>64517

Sending the request to the server...

Receiving the response from the server...Received packet is:
<response>
        <id>16289</id>
        <code>0</code>
        <measurement>63353</measurement>
        <value>65.42</value>
</response>27563

Temperature value: 65.42 F
```

Server-side Output:

```
Receiving the request from the client...Received packet is:
<request>
        <measurement>11749</measurement>
        <id>21208</id>
</request>47626

Message to send:
<response>
        <id>21208</id>
        <code>2</code>
</response>55305

Sending the response to the client...
Receiving the request from the client...Received packet is:
<request>
        <id>16289</id>
        <measurement>63353</measurement>
</request>64517

Message to send:
<response>
        <id>16289</id>
        <code>0</code>
        <measurement>63353</measurement>
        <value>65.42</value>
</response>27563

Sending the response to the client...
```

****