Assignment 1

JSON-based Hypertext Transfer Protocol (JHTTP)

1 Synopsis

This assignment is to create a system to transfer web pages over the network using client/server architecture. The system consists of two main distributed components: *server* and *client*, which may run on different hosts in the network. A server is a program that accepts multiple incoming TCP connections from clients and responds to requests sent by clients. To build this system, you need to create a new protocol called *Json-based Hypertext Transfer Protocol (JHTTP)* which is used by server and clients to communicate. JHTTP has a set of request methods to indicate the desired action to be performed for a given web page. Please note that web pages are text files only. This assignment has been designed to use TCP Sockets. Server creates multiple threads and uses them to process clients' requests. All the communicated messages between the server and clients must be encapsulated in the form of JSON messages (more details to come).

2 Server

A server maintains a root folder (directory) which contains all the web pages named as **www**. When the server program is executed, the program checks that if **www** folder exists on the same folder in which the server executable file resides. If this folder does not exists, server program should create it automatically.

Server accepts TCP connection requests from clients and responds to the communicated JHTTP requests. The server must be able to serve multiple clients concurrently (Threads should be used to achieve this functionality). As soon as a client is connected to the server, the server is ready to accept request messages including GET, PUT, DELETE, DISCONNECT requests. The list of requests and their meaning is given in Table 1.

Table 1: List of Requests

Request Type	**Meaning**
GET	A GET request is used to retrieve a web page content from the server.
PUT	A PUT request replaces or adds the target web page content on the server with the request payload.
DELETE	A DELETE request deletes the specified web page.
DISCONNECT	Terminates the connection with the server

- Path to one file should follow this rule: Starts with slash '/' + between 0 to 10 nested folders (the length of folder name ranges from 1 to 20 ACCEPTED CHARACTERS) + End with slash '/' + file name (the length of file name ranges from 1 to 10 ACCEPTED CHARACTERS) + Dot '.' + file extension (the length of file extension ranges from 1 to 5 ACCEPTED CHARACTERS). In what follows, you can find some examples:

  * /index.html (ACCEPTED)
  * /A1/B/C532vaAZu67/6D/7/A2/1rule.ex3 (ACCEPTED)
  * /index123450.html543 (NOT ACCEPTED)

- Path to one folder should follow this rule: Starts with slash '/' + between 1 to 10 nested folders (the length of folder name ranges from 1 to 20 ACCEPTED CHARACTERS) + ends with slash '/'. In what follows, you can find some examples:
  * /A/B/C/ (ACCEPTED)
  * /A1/B/C532vaAZu67/6D/7/A2/ (ACCEPTED)
  * / (NOT ACCEPTED)

- The List of ACCEPTED CHARACTERS: [a-z, A-Z, _, 0-9]

3 Client

A client is a command-line interface (CLI) that accepts commands from the end user, sends JSON request messages to the server accordingly, and shows server responses on the CLI in the desired format. Table 2 shows lists of commands and their arguments that the end-user can type. The end-user types a command and its required arguments on the CLI. Commands are executed by *Enter (Return)* key. According to each command, the client program creates a corresponding request message from Table 1 and sends it to the server. Upon on receiving a request message by the server, a response message is generated by the server and is sent back to the client. Please note that messages sent between the client and the server are all in JSON format (more details in Section 4). The response message can be either *success* or *error*. Client shows appropriate texts on the CLI based on the message received.

Please note following conditions:

- If an end-user enters a command which is not defined, has missing or extra arguments, or it has error in any forms, client program **prints nothing on the CLI** and goes to the next line waiting for a new command.

- Commands are case-sensitive.

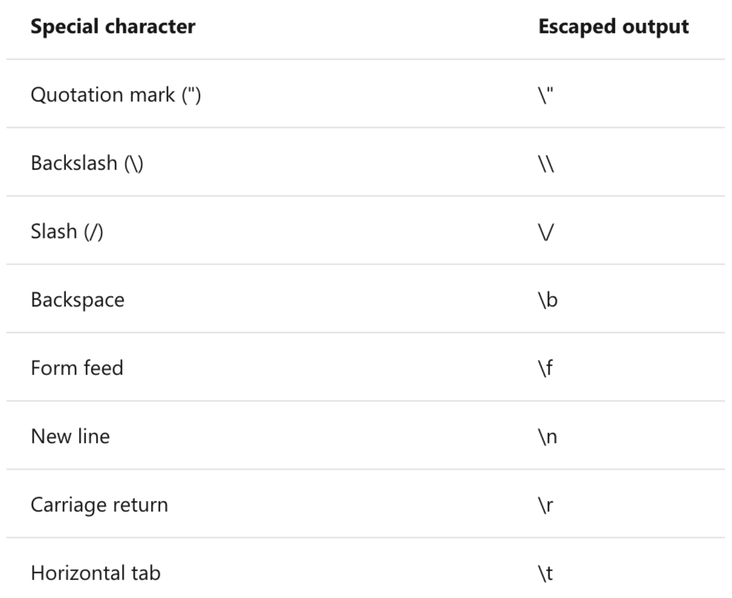
- If the source file in the put command cannot be found on the client system, the CLI shows **Source file not found**.

- Your client application is supposed to work with '/'. But Microsoft Windows uses '\' for paths. Your program should be platform independent and be able to work with '/' irrespective of OS. Python programmers might use **pathlib** library to solve this issue. Java seamlessly converts '/' to '\' on Windows. You can also further look into **File.separator** property in Java.

- Your program is supposed to only accept relative paths, for example, */index.html* or */cmd/index.html*. A relative path refers to a location that is relative to a root folder i.e. **www** folder. Please note that the root folder is not part of the target path. If *www* appears as part of the path, for example, */www/index.html*, it means that there is a subfolder called *www* within the root folder.

- If the source or target web pages on the client or server, respectively, contains special characters as shown in Fig. 1, you should replace them with the escaped output in the generated JSON messages or revert them when you are showing them to the end user.

Here is an example of how a client CLI session may look:



Special character	Escaped output
Quotation mark (")	\"
Backslash (\)	\\
Slash (/)	\/
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t

Figure 1: JSON special characters

Table 2: List of Client Commands

Request Type	**Arguments**	**Meaning**	**Example**
connect	<ip><port>	connect to a server with the IP address <ip> and port address <port>	connect 192.168.0.10 2000
get	<target>	retrieve the content of the specified web page from the server	get /index.html
put	<source><target>	upload the content of the specified <source> web page on the client file system to the specified <target> web page on the server	put /test.html /finance/index.html
delete	<target>	delete the specified web page from the server. <target> can be an empty folder.	delete /finance/test.html
disconnect		Disconnect from the server	disconnect

```
E:\>java -jar client.jar
connect 192.168.0.10 2000
Successfully connected
xyz illigal command
get /src/index.html
Not Found
put index.html /src/index.html
Ok
get /src/index.html
<html><body><h1>helloword</h1></body>m</html>
delete /src/index.html
Ok
disconnect
Successfully disconnected
```

4 JHTTP Protocol Specification

All messages exchanged between client and server must be encapsulated in the form of **UTF-8** encoded JSON messages that ends with the newline character ("\n") . Thus, you can use the following Java commands to send and receive JSON messages:

```
String line = in.readLine();
out.write((String.valueOf(obj)+"\n").getBytes("UTF-8"));
```

Python user might use following commands.

```
first_line = stream.readline()
clientsocket.send(bytes(msg+"\n","utf-8")
```

4.1 JSON messages

Messages are either "request" or "response". Thus, each JSON message has either of the following key-value pair:

```
"message":"request",
```

or

```
"message":"response",
```

4.2 Request messages

All request messages have a key/value pair that shows the "type" of the request message. "type" can either of the following: GET, PUT, DELETE, DISCONNECT.
If "type" is "GET" the format is:

```
{
    "message":"request",
    "type":"GET" ,
    "target":"relative path to the web page"
}
```

The value for"target" is the a relative path to the web page, for example "/cmd/index.html". You should validate the target to be a standard Linux path.
If "type" is "PUT" the format is:

```
{
    "message":"request",
    "type":"PUT" ,
    "target":"relative path to the web page",
    "content":"web page content in utf-8 format"
}
```

The value for "target" is the same as the GET message. "content" contains the content of the source web page (a text file) on the client encoded in utf-8 format. If the webpage file in the target does not exist, the file and all its required folder will be created. If the webpage file in the target path exists, then its content is replaced with the value of the "content" key.
If "type" is "DELETE" the format is the same as "GET":

```
{
    "message":"request",
    "type":"DELETE" ,
    "target":"/index.hmtl"
}
```

Finally, If "type" is "DISCONNECT" the format is:

```
{
    "message":"request",
    "type":"DISCONNECT"
}
```

4.3 Response messages

JSON format for *response* messages is as follows:

```
{
    "message":"response",
    "statuscode":"a valid JSON number data type" ,
    "content":"a valid JSON string data type"
}
```

The list of "statuscode" values and content values are shown in Table 3.

Please note that CLI shows the value of "statuscode" followed by the value of the "content" key for all"statuscode" values, for example *201 OK*. There is an exceptional case for 200 that only content value is shown for example `<html>test</html>`.
A few examples:

Table 3: Error codes and their description

Status Code	**Meaning**	The value of “content”
200	GET request was successful	The content of the target web page in utf-8 format
201	PUT request was successful, new file is created.	Ok
202	PUT request was successful, but file is overwritten.	Modified
203	DELETE request was successful	Ok
400	Target Web page does not exist (GET or DELETE)	Not Found
401	Request format is incorrect, missing or extra arguments, and wrong message types, and invalid targets	Bad Request
402	Any other errors	Unknown Error

```
{
"message":"response",
"code":"200",
"content":"<html><h1>Hello World!</h1></html>"
}
```

```
{
"message":"response",
"code":"400",
"content":"Not Found"
}
```

4.4 Connect and Disconnect

The end-user enters **connect** command immediately after running the client program.This command results in a TCP connection establishment with a server, for example:

```
connect 192.168.0.10 2000.
```

The end-user enters **disconnect** command to close down connection and quits program. This command results in a TCP disconnection with a server, for example:

```
disconnect
```

Please note following conditions:

- If connection is established, “Successfully connected” message is printed on the CLI otherwise client handles the exception and prints “No server” and waits for the next command. Any commands before establishment of connections simply goes to the next line and waits for the next command.

- When **disconnect** command is entered, client closes its socket, prints "Successfully disconnected" and quits program. Server clears it socket as well.
- Pressing Ctrl-C should terminate either client or server. Your program should be cleanly finished by terminating all running threads.
- If server gets disconnected abruptly (e.g., CTRL +C), the client clearly closes the socket and quits the program.
- If client gets disconnected abruptly (e.g., CTRL +C), the server handles the situation as **disconnect** command.

5 Technical aspects

For **Java** developers:

- Use Java 1.8 or later.
- All message formats should be UTF-8 JSON encoded. Use a JSON library for this (e.g., jsonsimple library).
- Package your programs into runnable jar files, one for server, one for client. Make sure all required external libraries are embedded into your .jar files.
- Your server and client should be executable exactly as follows:
```
java -jar server.jar -p port
java -jar client.jar
```
- Use command line option parsing (e.g. using the args4j library or your choice).

For **Python** developers:

- Use Python 3.
- All message formats should be UTF-8 JSON encoded.
- Your server and client should be executable exactly as follows:
```
python3 server.py -p port
python3 client.py
```

6 Report

Write 200 words at most for each of the following questions:

- In this project, we used TCP protocol for the communication between the client and server. Give two reasons **for** and two reasons **against** using UDP instead.
- Explain why we only need to set server's port? Explain how server knows the client's port to send a response?
- Suggest a mechanism that we can retrieve other file types from the server, for example, .jpg.

Use 12pt times font, single column, 1 inch margin all around. Put your name, login name (username), and student number at the top of your report.

7 Submission

You need to submit the following via Moodle:

- Your report in PDF format only.
- Your `server` and `client` executable files (Jar file for Java developers and .py files for python developer).
- For Java developers, your source files in a `.zip` archive only (not rar or other formats).
- For python developers, a readme file with a list all packages (pip install commands) that your code requires.