SAXION HOGESCHOOL

Assignment Chatting

Internet Technology 2024-2025

Table of Contents

1 Purpose

The purpose of this individual assignment is to become familiar with the principles of network applications by building a server and client application in Java.

Building a chat server and accompanying client in Java forces you to think about:

- How do client and server applications communicate with each other?
- How does a client know how to find the server?
- How can I ensure that multiple clients can log in simultaneously and send messages?
- How can I make sure clients can log in?
- What should I do when a client logs out?
- What possibilities does the Java programming language offer in this area?

The assignment consists of three levels of increasing complexity. At level 1, you implement a chat client for an existing server (written in Node.js). For level 2, you are asked to extend the client and implement your own server (written in Java) with new basic functionality. At level 3, you will have to implement file transfer functionality.

We end with a challenge level. At this level, you do not have to change the client or server, but instead, you will connect with an external machine where you must solve a couple of encoding and encryption challenges.

2 Level 1

Build a client that can communicate with the given Node.js server. The requirements are given in chapter 2.5. The protocol for communication with the server can be found when you clone the repository via the HBO-ICT website (Saxion HBO-ICT, 2024). The implementation should adhere to the given protocol specification to ensure the client works correctly.

By **correct** we mean:

- The client can set up a connection with the server.
- A user can log in to the server using the client and send messages to other logged in clients.
- To detect connection issues quickly the server periodically sends a heartbeat (PING) to every client and expects a response (PONG) from each client. If no PONG is received, then the connection with the client is closed by the server.
- A user can log out and disconnect from the server.
- Good error handling (e.g. checking username).

2.1 Repository

It is mandatory to create a repository via the HBO-ICT website (Saxion HBO-ICT, 2024). Use this repository to commit and push your work regularly. The protocol specification can also be found in the cloned repository.

2.2 The given server

On Brightspace you will find a ZIP file containing the code for the server you can use to develop/test your client. The server is written in Javascript. Please make sure to install a recent version of NodeJS. The NodeJS installation you've used in Web Basics is sufficient.

You can run the server by executing the following command in the command-line interface:

```
$ node server.js
```

The server also has options which can be set in the source code at the start of the file, such as:

- Configure the port on which the server listens to (default is 1337).
- Disable the sending of the PING message to the client.

2.3 Client code

The client application (and later the server application) needs to be implemented in Java. To help you get started, here are some code examples for implementing the client. The client must set up a socket connection with the server. The client needs at least two threads:

- One thread **listens to the input of the user**.
- The other thread **listens to messages from the server**. Using the input stream of the socket you can receive data from the server.

```
// Set up a connection with the server.
// If you are connecting locally use SERVER_ADDRESS = "127.0.0.1".
socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
```

```
// Get the input and output stream from the socket.
InputStream inputStream = socket.getInputStream();
OutputStream outputStream = socket.getOutputStream();
```

Please read the documentation of the Java Socket class (Oracle, 2024a). There is also a somewhat old but still relevant socket tutorial (Oracle, 2024b) available which you can use.

Tips:

- Start by creating a simple client that can create a socket connection with the server. Use the code snippets provided in this document.
- Apply *separation of concerns* in your code. The way the user interacts with your application is not the same as the messages exchanged between the client and the server. Create a simple menu in the client.
- Disable the sending of the PING messages in the server during initial development, so that connections are not disrupted. You can enable this later.

2.4 Input and output

Reading strings

To read data, the **InputStream** of a socket can be used. Java has a helper class to read whole strings at once.

```
InputStream is = socket.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(is));
// Block thread until a full line is read
String line = reader.readLine();
```

Sending strings

You can use the **OutputStream** in the other direction. This only allows you to send a few bytes. Java also has a helper class for a whole string.

```
// Send message using the print writer.
OutputStream os = socket.getOutputStream();

PrintWriter writer = new PrintWriter(os);
writer.println("Hello world");


// The flush method sends the messages from the output buffer to client.

writer.flush();
```

2.5 Requirements

2.5.1 Business

Req.	Description	F/NF	Priority	Source
RQ-B100	The application functions as a chat client	F	MUST	Course Director
RQ-B101	The application communicates with the given node.js server (compatibility)	NF	MUST	Course Director
RQ-B102	The application is written in the Java programming language, version 21.	NF	MUST	Course Director

2.5.2 User

Req.	Description	F/NF	Priority	Source
RQ-U100	The user can use the client to login to the server by providing a username	F	MUST	RQ-B100
RQ-U101	The user can send a (broadcast) message to other logged in users which will be shown to all other logged in users in the format "<username>: <message>"	F	MUST	RQ-B100
RQ-U102	The user can logout from the server	F	MUST	RQ-B100
RQ-U103	In the client application a help menu will be shown when typing "help" followed by the enter key and shows all available commands in the client in a list containing "<command>: <explanation including available arguments>"	F	MUST	RQ-B100
RQ-U104	The user is informed when the connection fails or when a user tries to login with a name that is invalid or already in use.	F	MUST	RQ-B100

2.5.3 System

Req.	Description	F/NF	Priority	Source
RQ-S100	The client implements the specified protocol of the given node.js server (compatibility)	NF	MUST	RQ-B101
RQ-S101	The client can connect to one server at a time	F	MUST	RQ-B100

RQ-S102	The client application responds to ping messages from the server with a pong according to the given protocol	F	MUST	RQ-B101
RQ-S103	The application uses a command-line interface (CLI) or[1] is build using JavaFX[2]. Other libraries (except for Jackson) are not allowed.	NF	MUST	Course Director

[1] Mixing CLI and JavaFX is not allowed.

[2] Swing, Springboot, etc. are not allowed. It is advised to choose the CLI interface. JavaFX can be difficult due to the threading model used. This course will not teach you on how to use this properly. Some guidance can be given during the lessons. Only use JavaFX if you are up to a challenge! Discuss with your teacher. Please read about the threading model used and understand what Platform.runLater() does. Suggested is the use of the Model-View-ViewModel (MVVM) design pattern.

3 Level 2

Now that you have a working client application, it is time to expand it to do more than broadcasting messages. In level 2 you will have to implement your own server that implements level 2 functionality in addition to the level 1 functionality. In addition to implementing the server, you should also extend the client.

Implement the extra functionality as stated in the requirements in paragraph 3.5. You are going to work on:

- Listing the currently connected users
- Sending and receiving private messages
- Rock/paper/scissors game

Extend the protocol given in the file **doc/protocol.md** file in the cloned repository with the above-mentioned extra functionality. Describe the commands that the server must process and the format of the messages. **Discuss this with your practical teacher before you start implementing.**

Then implement the protocol in both the client and the server. Note that you must implement your own server. Your source code should be set up so that it is readable and maintainable. So, think about how you can implement the extensions in a modular way.

Then create **tests** for your server. You need to automate the test using JUnit. Your test results need to be documented. **An example test application including a couple of tests can be downloaded via Brightspace**. Two tests still fail on the given NodeJS server. Make sure they pass when implementing your own server. Of course, you must write more tests when implementing your own server. If you wish you can automate the starting of your own server by integrating the test suite in your own application.

For Windows users, please read Appendix A if you have trouble starting your chat server.

3.1 Server code

The client sets up a socket connection with the server. To establish a connection, the server should listen for incoming requests. You can do this by creating a ServerSocket class with a certain port (for example port 1337) and then calling the serverSocket.accept() method. This method is *blocking* which means the server will wait until a client connects (verify this with your debugger).

```
// Create a socket to wait for clients.
serverSocket = new ServerSocket(SERVER_PORT);
while (true) {
    // Wait for an incoming client-connection request (blocking).
    Socket socket = serverSocket.accept();
    // Your code here:
    // TODO: Start a message processing thread for each connecting client.
    // TODO: Start a ping thread for each connecting client.
}
```

Note: If you test the application on a remote server via the Saxion network, make sure your server uses port 1337 or port 8080. With other ports, after a while, the Saxion firewall will block the connections.

In the example above, the server has two threads per client. One thread **listens** to messages from a client and the other thread periodically **sends** a ping message to connected clients.

You can receive data from the client using an input steam. An output stream can be used to send data to the client.

```
// Get the input and output streams for the socket.
InputStream inputStream = socket.getInputStream();
OutputStream outputStream = socket.getOutputStream();
```

It is advised to read the Java documentation about the ServerSocket class (Oracle, 2024c).

3.2 Connected clients

Show a list of connected clients in your client application. Make sure the list contains all connected clients.

Make sure this function is well-tested by creating automated tests.

3.3 Private messages

It should be possible to send private messages from one client to another client. The client who wants to send the message must input the name of the receiving client. Only this client will receive the message. Private messages are always sent via the server, not directly between clients.

Make sure this function is well-tested by creating automated tests.

3.4 Rock, paper, scissors

Your task is to design and implement a rock, paper, scissors game (Wikipedia, 2024).

The game consists of the following steps:

1. User A starts the game by selecting a menu item (“Start rock paper scissors game”).
2. User A must select another player to play with (we will call the other user “user B”).
3. User B receives a message that the game is started.
4. Both users make a choice (rock, paper or scissors). They are not able to see what the other user selected.
5. When the server receives both choices, it selects the winner according to the game rules.
6. Both users receive the outcome of the game (the winner of the game and what the other user had selected).

Start by designing the protocol. Make sure to discuss the protocol with your teacher. After that, make sure to implement the functionality in the server and client.

Make sure this function is well tested by creating automated tests.

3.5 Requirements

3.5.1 Business

Req	Description	F/NF	Priority	Source
RQ-B200	The application consists of a chat client and chat server both written in the Java language	F	MUST	Course Director
RQ-B201	The application uses Java packages to separate the client and the server	NF	SHOULD	Course Director
RQ-B202	The chat client and chat server communicate using a well-defined and documented protocol	NF	MUST	Course Director
RQ-B203	The chat server can handle multiple connected chat clients simultaneously (only limited by memory available on the chat server)	NF	MUST	Course Director

3.5.2 User

Req	Description	F/NF	Priority	Source
RQ-U200	The user can view a list of all connected clients on the server	F	MUST	Course Director
RQ-U201	The user can send a direct (private) message to another user	F	MUST	Course Director
RQ-U202	A user should be able to start a rock paper scissors game with another user.	F	MUST	Course Director
RQ-U203	A user should be notified when a rock, paper, scissors game is started in which the user participates	F	MUST	Course Director
RQ-U204	A user should be able to select an option (rock, paper or scissors) when a game is started	F	MUST	Course Director

RQ-U205	When a rock paper scissors game is started with an invalid user (e.g. user does not exist) the game will not be started, and the user will be notified.	F	MUST	Course Director
RQ-U206	When a rock, paper, scissors game ends, both participating users will receive the result which consist of the winner (according to the game rules) and the choice made by the other user.	F	MUST	Course Director
RQ-U207	Only one number rock, paper, scissors game runs at any given moment.	F	MUST	Course Director
RQ-U208	When a user starts a rock, paper, scissors game, while two other users on the server are already playing, the game is not started and the user is informed with the message “Another game already running between <user a> and <user b>”, were user a and user b are replaced with the corresponding usernames.	F	MUST	Course Director
RQ-U212	Users are notified when another user connects to the server	F	MUST	Course Director

3.5.3 System

Req	Description	F/NF	Priority	Source
RQ-S200	The system should be designed in such a way that it is easy to replace the user interface. E.g.: move from the command line interface to a graphical user interface	NF	MUST	Course Director
RQ-S201	The client should be designed in such a way that it is able to support different natural languages at a later moment in time without changing the protocol (e.g., English, Dutch for the menu's).	NF	MUST	Course Director
RQ-S202	The system is well tested, with at least all functionality (users, listings, private messages, rock paper scissors game) covered by automatic protocol testcases	NF	MUST	Course Director
RQ-S203	A username may only consist of characters, numbers, and underscores ('_') and has a length between 3 to 14 characters	F	MUST	Course Director
RQ-S204	The server should only depend on the default Java packages and downloaded Jackson libraries. Other libraries are not allowed.	NF	MUST	Course Director

4 Level 3

Please implement the following functionality:

- File transmission between two clients, with a checksum. The checksum is to make sure the received data is the same as the original data (File integrity).

A file transfer consists of two parts:

1. Sending the file from the sending client to the server (upload)
2. Sending the file from the server to the receiving party (download).

Make sure that the receiving party gives permission to receive a file first. The implementation of the file transfer sends the file over a separate socket so that the client can continue chatting with others. Also make sure that the integrity of the file is safeguarded (using SHA-256, for example).

Extend the protocol from level 2 with the extra functionality. Describe the commands that the server must process and the format of the messages. **Discuss this with your teacher before you start implementing.**

Then implement the protocol extension in both the client and the server.

You do not need to write (automated) testcases for level 3. See Chapter 6 for what you have to show in the video.

4.1 Design

Figure 1 gives a general overview on how the file transfer should work. You must design the protocol messages including their content yourself. Feel free to make any adaptations when necessary.

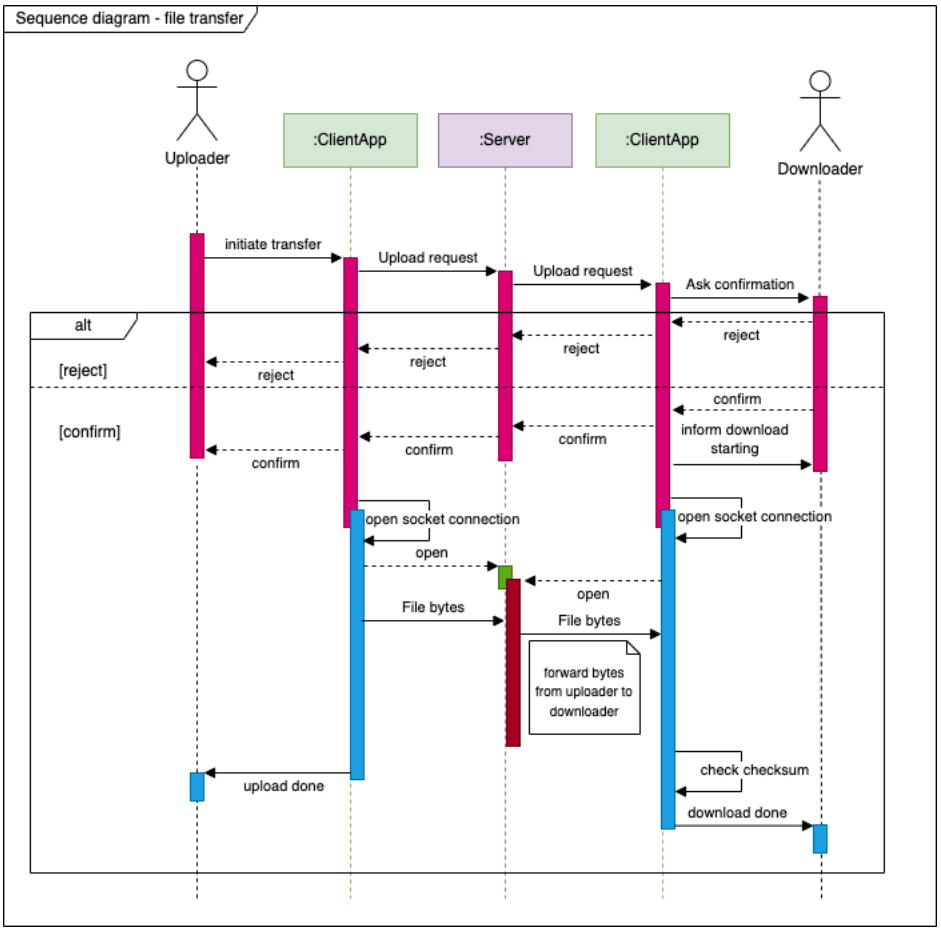

Figure 1 General design file-transfer

4.2 Level 3: Requirements

4.2.1 Business

Req	Description	F/NF	Priority	Source
RQ-B300	The application adheres to the requirements of Level 2	NF	MUST	Course Director

4.2.2 User

Req	Description	F/NF	Priority	Source
RQ-U300	A logged in user can send a file to another logged in user	F	MUST	Course Director

RQ-U301	A user who receives a file (the receiver) first needs to acknowledge the transmission before the file is stored on the receiver's computer	F	MUST	Course Director
RQ-U302	It is possible to transfer files larger than the amount of memory available on the client computer or server.	F	MUST	Course Director
RQ-U303	It is possible to transfer binary files	F	MUST	Course Director

4.2.3 System

Req	Description	F/NF	Priority	Source
RQ-S300	The protocol uses a checksum to identify when a file gets corrupted during transmission	NF	MUST	Course Director
RQ-S301	File transfer is implemented over a separate socket	NF	MUST	Course Director

5 Challenge Level

At this level, you will solve a challenge. The URL and port number of the challenge will be given on Brightspace, in the submenu “Practical Assignment”.

5.1 Documentation

A write-up is expected. A write-up is a PDF file containing a short and structured explanation of your steps to address and solve this specific practical challenge and the issues you encountered during the process.

6 Deliverables

The following deliverables must be submitted to Brightspace at the end of the module. An instructional video with explanations on how to do the submission is given in Brightspace.

Video clip

A screen recording with the following requirements:

- Recorded with Kaltura *
- At most 15 minutes *
- Including screen and camera recording *
- The video clip **must** include: *
    - 2 minutes – Show your self-made class diagram and explain how the classes are related.
    - 1 minute – Run your test cases and show that they succeed.
    - 4 minutes – Show your server code and explain how the server code handles incoming connections, messages and file transfer.
    - 4 minutes – Run the server and two clients, showing that you can log in, send broadcast messages, play the rock-paper-scissors game and logout.
    - 4 minutes – Run the server and two clients and show that you can transfer a file between two clients and that the accept/reject functionality works. Show the server code and explain how the server handles incoming file transfers.
- Explanations must be spoken by you. Subtitles or AI voices are not permitted. *

Code and files (ZIP file)

- The source code of the server and client *
- The source code of the challenge level
- An updated protocol specification (doc/protocol.md) *
- A Wireshark capture (pcapng) which contains: logging in, playing a rock/paper/scissors game, a file transfer, and logging out *
- Challenge level writeup (PDF)
    - Title page with your name and student number
    - Page numbers
    - Output when running your script/code
    - A short description of the issues you encountered and how you solved them
    - Maximum of 3 pages, excluding the title page

The assignment deadline can be found on Brightspace.

*** These marked requirements are mandatory for getting a grade.**

7 Bibliography

Saxion HBO-ICT. (2024, July 22). *Repo HBO-ICT*. Retrieved from Repo HBO-ICT: https://repo.hboictlab.nl/

Oracle. (2024b, July 22). *What is a Socket?* Retrieved from Oracle Java Documentation: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

Oracle. (2024a, July 22). *Class Socket*. Retrieved from Java 21 API documentation: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/net/Socket.html

Node.js. (2024, July 22). *Download Node.js*. Retrieved from Node.js: https://nodejs.org/en/download/package-manager

Oracle. (2024c, July 22). *ServerSocket class*. Retrieved from Java 21 API Documentation: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/net/ServerSocket.html

Wikipedia. (2024, July 22). *Wikipedia*. Retrieved from Rock paper scissors: https://en.wikipedia.org/wiki/Rock_paper_scissors

Appendix A Socket listening issues Windows

A.1 Excluded port ranges Windows

Windows programs have the possibility to reserve port ranges during start-up. Windows Hyper-V does this for example.

You can view the reserved port ranges by executing this command[3]:

```
netsh int ip show excludedportrange protocol=tcp
```

It could be that during certain system boots port 1337 is reserved. This can differ every boot. When port 1337 is included in the range, you will not be able to start your Chat Server on port 1337.

To resolve this issue, add port 1337 to the *excludedportrange* so that it can be used by your application. You can do this by executing the following command:

```
netsh int ipv4 add excludedportrange protocol=tcp startport=1337
numberofports=1
```

It may also be that port 1337 is claimed by an internal Windows process, in which case excluding the port fails. You can check this using the Windows resource monitor: Task Manager --> Performance --> Open Resource Monitor --> Network --> TCP Connections (this may take a few seconds). If not shown, it should be visible using this process monitor tool that also takes suspended processes into account: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon In any case, the resolution of this problem is the same, which is to reboot so that Windows chooses a different port than 1337.

A.2 Firewall

Another common issue is a misconfiguration of the windows (or other) firewall. Please make sure port 1337 is allowed in your firewall.

A.3 More info

- https://www.faqcode4u.com/faq/263071/how-to-see-what-is-reserving-ephemeral-port-ranges-on-windows
- https://stackoverflow.com/questions/54010365/how-to-see-what-is-reserving-ephemeral-port-ranges-on-windows
- https://github.com/docker/for-win/issues/3171

[3] You will have to start a new command prompt/ Powershell as administrator to execute this.

Appendix B Coding Style Guide

B.1 Filename

The source file name consists of the case-sensitive name of the top-level class it contains (of which there is exactly one), plus the .java extension.

B.2 Whitespace characters

Aside from the line terminator sequence, the ASCII horizontal space character (0x20) is the only whitespace character that appears anywhere in a source file. So, tab characters are not used for indentation.

B.3 Source file structure

A source file consists of, in order:

1. Package statement
2. Import statements
3. Exactly one top-level class

B.4 Use of optional braces

Braces are used with if, else, for, do and while statements, even when the body is empty or contains only a single statement.

Other optional braces, such as those in a lambda expression, remain optional.

B.5 Braces

Braces follow the Kernighan and Ritchie style ("Egyptian brackets") for nonempty blocks and block-like constructs:

- No line break before the opening brace, except as detailed below.
- Line break after the opening brace.
- Line break before the closing brace.
- Line break after the closing brace, only if that brace terminates a statement or terminates the body of a method, constructor, or named class. For example, there is no line break after the brace if it is followed by else or a comma.

Example:

```
if (condition()) {
    try {
        something();
    } catch (ProblemException e) {
        recover();
    }
} else if (otherCondition()) {
    somethingElse();
} else {
    lastThing();
}
```

B.6 Block indentation

Each time a new block or block-like construct is opened, the indent increases by four spaces. When the block ends, the indent returns to the previous indent level. The indent level applies to both code and comments throughout the block.

B.7 One statement per line

Each statement is followed by a line break.

B.8 Where to break

- A method or constructor name stays attached to the open parenthesis (() that follows it.
- A comma (,) stays attached to the token that precedes it.

B.9 Vertical Whitespace

- A single blank line always appears between consecutive members or initializers of a class: constructors, methods.
- A single blank line may also appear anywhere it improves readability, for example between statements to organize the code into logical subsections.

B.10 Horizontal whitespace

Beyond where required by the language or other style rules, and apart from literals, comments and Javadoc, a single ASCII space also appears in the following places only:

- Separating any reserved word, such as if, for or catch, from an open parenthesis (() that follows it on that line
- Separating any reserved word, such as else or catch, from a closing curly brace (}) that precedes it on that line

- Before any open curly brace ({)
- Between the type and variable of a declaration: List<String> list

B.11 One variable per declaration

Every variable declaration (field or local) declares only one variable: declarations such as “int a, b;” are not used.

B.12 Naming

- Class names are written in UpperCamelCase.
- Method names are written in lowerCamelCase.
- Constant names (final static) use UPPER_SNAKE_CASE
- Non-constant field names (static or otherwise) are written in lowerCamelCase.
- Parameter names are written in lowerCamelCase.
- Local variable names are written in lowerCamelCase.

Based on: https://google.github.io/styleguide/javaguide.html