University of St Andrews School of Computer Science

CS2003 — Internet and the Web — 2020/21

Assignment: Net2
Deadline: 2020-11-11 Credits: 30% of coursework mark

MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Aim / Learning objectives

The aim of this assignment is to extend your experience in building networked systems, both by building a client and server and understanding how this can interact with a web page.
Your task is to complete a simple message board application, consisting of a client and a server. You are given a web-based 'front-end' *which should not be modified*. This web-based front-end reads files from your filespace. You will create a server application that can create these files. What goes into the files are messages sent from a remote client application which you will also write. When the client application sends some text, it is received by the server, and the server creates a file containing the message. The web front-end reads these files and displays them on a web page. This is shown in Figure 1 below. Your client and your server will be unaware of the web page that reads the messages.
Useful documentation for Java 11 can be found at:
https://docs.oracle.com/en/java/javase/11/
https://docs.oracle.com/en/java/javase/11/docs/api/allclasses.html

1 Requirements

You should create a client program and a server program in Java. Your overall solution should compile on the CS lab Linux machines. The server program should run on the host servers and the client program should run on the lab machines.

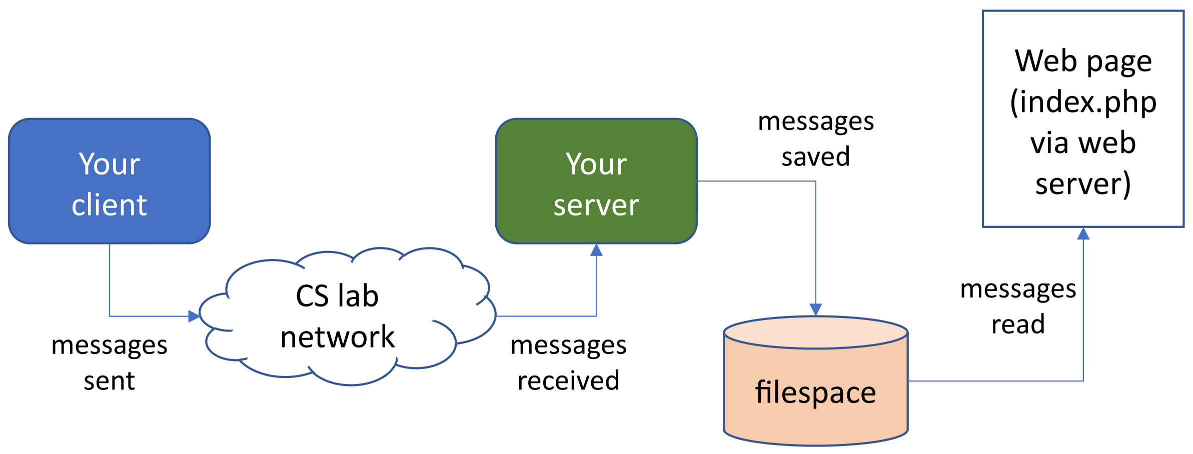


Figure 1: A logical diagram of the communication path for messages in this practical.

R1 The application should provide reliable data transfer, so it should use TCP.

R2 The application should allow the client program to connect to the server program and to deliver a printable text message of 256 characters or less. That message is saved by the server in a "well-known" place, from where it is read and displayed by a web page (the web page is provided *and should not be modified*).

R3 The client program must be invoked from the command line, and have a simple text-based interface for entering text messages. The client *MUST* be named `DMBClient` (for "Daily Message Board" Client). A simple client is provided which should work for the initial set of requirements (**R1-R5**) but will need adaptation to meet further requirements.

R4 The server program should run continuously, and must create files as expected by the web page (see below for more information about the web page). The server *MUST* be named `DMBServer`, and should:

- Make sure that the correct directory for today's messages exists.
- Receive the messages from the client program.
- Create the message files from the client text, and store them in today's message directory.

You must run your server program on your logical server with the FQDN:

`<your-username>.host.cs.st-andrews.ac.uk`

You should pick an individual port number as advised in the week 01 lab session (and as listed in *CS2003-usernames-2020.csv*). Client programs must run on the lab machines.

R5 The client and server programs should both use the Java *Properties* class for all run-time configuration (e.g. port numbers). Recompilation should not be required for any such configuration. I have provided a skeleton *cs2003-net2.properties* file which is also used by the sample DMBClient — your programs *MUST* use the variable names in this file, but you are allowed to extend this file with additional compile-time variables as you see fit. You should *NOT* hardcode any parameters such as your username, port number or file-paths into your programs — you will see examples of how to do this at run-time in *cs2003-net2.properties*.

R6 Allow the server program and client program to deal gracefully with TCP connection problems, such as when the server is not running, or if an abnormal TCP connection termination occurs, e.g. client and server have an active TCP connection, and the client is terminated. (*Hint:* look at the kind of errors that could occur in each program, and the exception handling for Socket and ServerSocket.)

R7 In your report, discuss the security and privacy issues with your application. Focus on what *threats* (security and privacy problems) are present, rather than on the mechanisms / technology that might offer solutions (we will examine this later on in the course). (*Hint:* Writing 'there is no encryption' is not enough.)

(*Hint* for **R8** and **R10**: the sequence `::` at the beginning of a line of text entered by the user indicates to the client that what follows is a 'command' to the program. Your client and server code will need to be modified to generate and accept the application PDUs described below. Regular expressions (see the week 5 exercise class) may also be useful here.)

R8 Allow the client to choose from a list of classmates to send a message to, based on their individual usernames.

At the client, the user now types:

```
::to rrrr mmmmmmmm
```

where `rrrr` is a username for a classmate who will receive the message, and `mmmmmmmm` is the text of that message to be sent. Suitable checks should be made in your client code to ensure that the message can be sent to `rrrr` at that time. Once connected to the correct server, the format of the application PDU that is transmitted by the client is:

```
::from uuuu mmmmmmmm
```

where `uuuu` is the username of the user running the client (e.g. you!), and `mmmmmmmm` is the text of the message that is being sent. The information `from uuuu` should be stored at the receiving server as part of the message.

To discover the hostname and port number of a particular classmate's server, I have provided a CSV file *CS2003-usernames-2020.csv* which your client program should treat as a flat-file database and use to generate the required information. *Hint:* you may wish to load this CSV file into a HashMap (refer back to CS1003).

For initial development you can send from your own client running on a lab machine to your own server running on your host server (i.e., the protocol does not preclude *Alice* from sending to *Alice*). But to completely test that you have fulfilled this requirement, you will need to coordinate with your classmates to test sending to their servers. This should be documented in your report.

R9 *Note: Completing these final two Requirements may take a considerable amount of time compared to **R1**-**R8**. Also, it could require a lot of additional coding, depending on how you have designed and implemented your program so far. Before you attempt this, please make sure you keep a working copy of your code and report which has **R1**-**R8** implemented and documented.*

As well as sending messages, allow the Java client to retrieve from the server all messages from today, or all messages from a previous day, by specifiying a date.

Use a simple *session protocol* as defined below. The user can enter the following message format at the client:

```
::fetch [date]
```

The string `[date]` indicates that the date is optional (hence the square brackets – you would not type those as a user). If a date is not provided by the user, then the client should use today's date. The application PDU sent to the server is:

```
::fetch date
```

If the date is determined by the server to be valid (i.e., in %Y-%m-%d form such as `2020-12-25`), then all messages from that date should be returned to the client using the following application PDU format:

```
::messages date
  tttttt mmmmmmmm
  tttttt mmmmmmmm
  tttttt mmmmmmmm
```

```
      [ ... continuing list of messages .. ]
  ::end
```

where `date` is the date of the message list, `tttttt` is the specific timestamp for each message, and `mmmmmmmm` is the body of the text sent by the user, which includes the `from uuuu` (as in **R8**) for each message. There is one message per line.

If the date is valid, but there are no messages for that date, the server's response should be the application PDU:

```
::none
```

If no valid date is determined by the server, or there is some other error, then the server's response should be the application PDU:

```
::error
```

R10 Extend your server so that it can use one or more queues to deal with requests from multiple users. You can adapt the code given in exercise classes as required (but remember to attribute).

The Java client and server must be completed as described in *Requirements* **R1**-**R6**, as an absolute minimum.

*Please make sure that you have a working submission fulfilling the Requirements **R1**-**R6** before you attempt any of the other Requirements.*

There is example code on studres for you to use as you wish. You can also use any code examples distributed for the lab sessions. If you prefer, you can simply write all your own code. *You should not need to use any third party code.*

Once you have completed **R1**-**R6**, you should attempt as many of the remaining requirements as you are able to within the time you have.

Hint: The server-side program does most of the work, saving the text to be displayed in a well-known place. The main role of the client program is to connect to the server program, using a TCP connection, and send a line of printable text and/or commands. This is why you have been provided with a simple client - the bulk of your time should be spent on developing the server.

2 Example code and messages

In *CS2003/Practicals/net2/* you will find:

- The file *cs2003-net2.php*, which needs to be served by the School web servers. You should copy this file to a directory called:

  ~/nginx_default/cs2003-net2/

  and rename it *index.php*.

- Two directories, *2020-09-20/* and *2020-09-24/*, which can also be copied to the same directory: these contain examples of the messages, including filenames, that conform to the description given below in Section 3, *How the web page works*.

- The configuration file *cs2003-net2.properties* which you should use and extend as needed (see *Requirement* **R5**). Your programs *MUST* use the variable names listed in this file. The Java file *Configuration.java* can be used to process this configuration file, but you are also welcome to write your own code to do this.

- The files *DirAndFile.java* and *TimeStamp.java*, which you can use as you wish.

- The file *CS2003-usernames-2020.csv*, which you will need for attempting *Requirement* **R8**.

3 How the web page works

A very simple web page has been written for you to use directly: *you do not need to change anything in that web page – it will just work*.
You should copy the file *cs2003-net2.php* to the directory *~/nginx_default/cs2003-net2/*, and rename it to *index.php*, so that it looks similar to Figure 2 and be visible via:
https://<your-username>.host.cs.st-andrews.ac.uk/cs2003-net2/
The web page is not required for your code to work, but it can be used as a simple testing tool. It is designed to read files stored as follows:

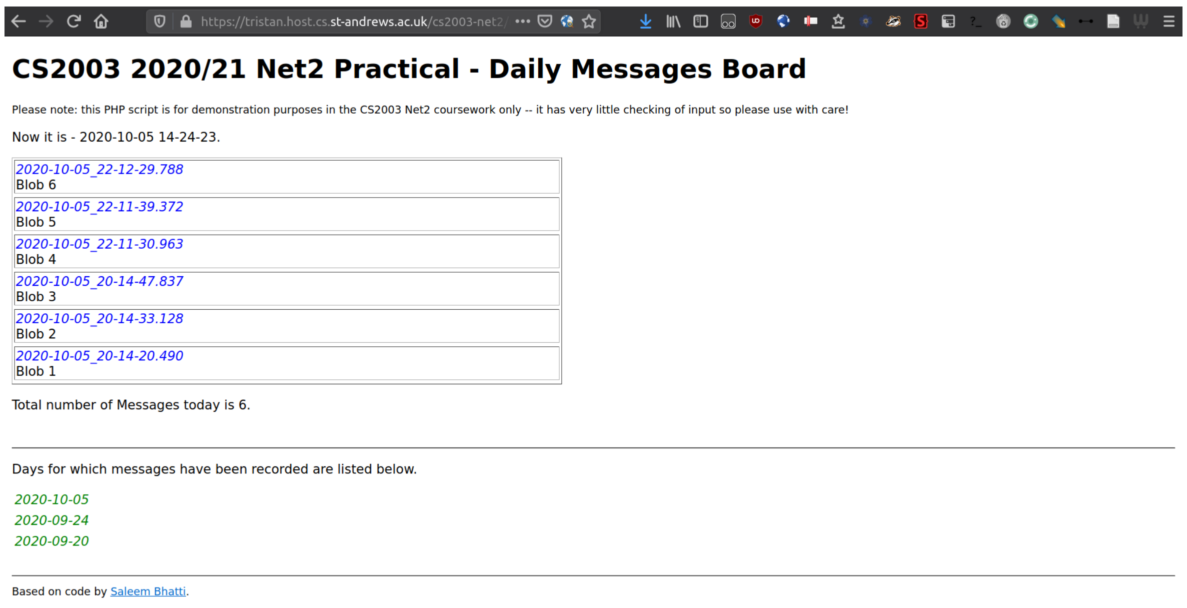
- In the same directory as the *index.php* file, it looks for a directory with the name:

  ```
  YYYY-MM-DD
  ```

  representing `YYYY` as the year, `MM` as the month, and `DD` as the day, e.g.

  ```
  2020-09-24
  ```

- Within that directory, the web-page code assumes there are text files containing the messages, with each file named with a timestamp, like this:


https://tristan.host.cs.st-andrews.ac.uk/cs2003-net2/

CS2003 2020/21 Net2 Practical - Daily Messages Board

Please note: this PHP script is for demonstration purposes in the CS2003 Net2 coursework only -- it has very little checking of input so please use with care!

Now it is - 2020-10-05 14-24-23.

2020-10-05_22-12-29.788
Blob 6

2020-10-05_22-11-39.372
Blob 5

2020-10-05_22-11-30.963
Blob 4

2020-10-05_20-14-47.837
Blob 3

2020-10-05_20-14-33.128
Blob 2

2020-10-05_20-14-20.490
Blob 1

Total number of Messages today is 6.

Days for which messages have been recorded are listed below.

2020-10-05
2020-09-24
2020-09-20

Based on code by Saleem Bhatti.

Figure 2: Example: The web page accessed in Firefox (note that your URL will be different).

`YYYY-MM-DD_HH-mm-ss.SSS`

with year, month and day as above, followed by `HH` hours, `mm` minutes, `ss` seconds and `SSS` milliseconds, e.g.:

`2020-09-19_07-32-27.209`

The WWW page uses the filename as a timestamp, and this is how it 'knows' which messages are most recent. *So, your Java server program must put the messages in the correct directory, with the correct filename.* (The code in *cs2003-net2.php* is very simple, and does very little checking of it its input.)

Submission

A single file containing your code and report in ZIP format must be submitted electronically via MMS by the deadline. Submissions in any other format will be rejected.
Specifically, your submission should contain:

Java Code. A single directory with all the source code (.java files) and configuration files (i.e., cs2003-net2.properties) needed to compile and run your application. This directory should include a server program that compiles and executes from the School Linux host servers, and a client program that compiles and executes from the command line on the School Linux lab machines. It should be possible for a marker to run *javac *.java* in that directory and compile your programs, ready to run with *java*. You should not use any third party code, libraries, or environments. *If the marker cannot compile or run the code you have submitted on the School's Linux machines, then you will not be given full credit for your solution.*

Report. A single PDF file which is a report with the usual School structure. In your report, where appropriate, try to concentrate on why you did something rather than just explaining what the code does:

- A simple guide to running your programs.
- A summary of the functioning of your programs, indicating the level of completeness with respect to the *Requirements* listed above.
- Use diagrams showing to show the design of your programs, especially if you have attempted *Requirements* **R8**–**R10**.
- An indication of how you tested your application, including appropriate screenshots of the WWW page showing your application working. If you have designed and implemented *Requirements* **R8**–**R10**, then testing and evidence of operation of those extensions should also be provided.
- An accurate summary (e.g. a table) stating which files or code fragments you have written yourself, any code you have modified from that which was given out as examples, or any code which you have sourced from elsewhere.
- The names of anyone you collaborated with during the assignment (please read Section 4, *A note on collaboration*, below).

Please first build a working solution meeting the *Requirements* **R1**–**R5**, before trying anything more complex. Attempts at *Requirements* **R6**–**R10** will stand you a better chance of getting a higher mark for your work.

Your final mark will depend on the overall ***quality*** *of your submission, both the code and the report. Attempting more of the Requirements will only get you a better mark if the quality of your submission is also sufficient.*

4 A note on collaboration

I encourage you to discuss this assignment with your classmates, though, of course, the code and the report that you submit should be your own work. For *Requirement* **R8**, you may want to test your implementation with a classmate. You should indicate in your report if you did this and who with.

Policies and Guidelines

Marking
See the standard mark descriptors in the School Student Handbook:
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors
Lateness penalty
The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties
Good academic practice
The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice/