

# 1DV701 Assignment 2 – Web Server

---

The second assignment is designed to familiarise yourself with the Java networking API. Your task is to develop a web server using Java that is capable of serving static content. This assignment should improve your understanding of TCP, server sockets, and the HTTP protocol.

Note: This assignment is either performed alone or in groups of two. Groups of three or larger are not allowed

## *Reference material*

- [HTTP overview](#)
- [Java networking tutorial](#)
- [Official Java documentation](#)

## *Useful downloads*

- PuTTY client (for Windows users).
  - If you use Homebrew (on macOS), you can use: `brew install telnet` You can also use `nc` instead of telnet on either platform.
- Insomnia REST client or Postman REST client – REST clients for testing
- POST tldr.sh – Community driven cheat-sheets for unix commands

## Problem 1

Implement a simple web server that accepts multiple client connections on some port (e.g., 8888) using the `ServerSocket` class. The server should serve (at least) HTML and PNG files from a local directory, using the GET method in the HTTP protocol.

The response from the server should include *correct and relevant headers* as well as the requested file. The server should run until you terminate it (ctrl-c or ctrl-d keyboard shortcut).

*Program requirements:*

- Two [program arguments](#) in the specified order
  - The port (listening port)
  - [Relative path](#) to /public directory. (Do **NOT** use absolute paths)
- You are *not* allowed to use third-party libraries or existing Java classes (such as `com.sun.net.httpserver`) since these make the problem trivial.
- Console output from the program should provide useful debugging information and in general make sense. Eg. connections being established and content served is a good idea to display.
- Use the provided public folder.
- Use the provided python test script to check if your implementation works as expected.

*Note: What is the /public directory?*

Think of this folder as the serving directory from which our web server will serve html files and images.

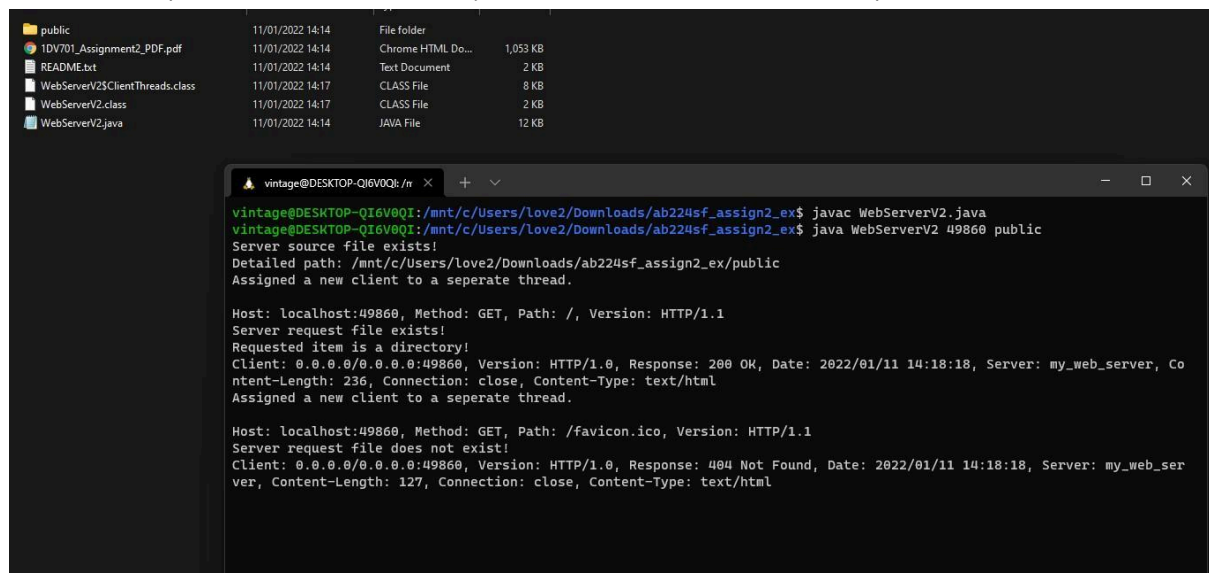
Accessing `http://yourip:port/stuff.html` should serve the file from `your_project_folder/public/stuff.html`.

*Note: Using the command line*

Your assignment is going to be tested via the command line, it will not be run via an editor/IDE. Note that this means you should verify that everything works correctly via the command line. If you are on Windows, WSL is recommended to use here as to get into a Linux environment for more convenient testing. Java also exists for powershell/cmd. Compiling is done via the command "javac" while the actual class is run using the "java" command.

*Note: An example of someone doing it right*

The following image represents someone that has fully completed this problem (and a small part of P2) and has provided all the parts necessary to get full marks. Take note of the document structure of the submission (visible in the file explorer) and the way compiling and running the solution works. Take special note of where the 'java' main file is in relation to the public folder.



The screenshot shows a file explorer on the left with the following files and folders:

File/Folder	Date/Time	Type	Size
public	11/01/2022 14:14	File folder	
1DV701_Assignment2_PDF.pdf	11/01/2022 14:14	Chrome HTML Do...	1,053 KB
README.txt	11/01/2022 14:14	Text Document	2 KB
WebServerV2ClientThreads.class	11/01/2022 14:17	CLASS File	8 KB
WebServerV2.class	11/01/2022 14:17	CLASS File	2 KB
WebServerV2.java	11/01/2022 14:14	JAVA File	12 KB

The terminal window on the right shows the following commands and output:

```
vintage@DESKTOP-QI6V0QI: /mnt/c/Users/Love2/Downloads/ab224sf_assign2_ex$ javac WebServerV2.java
vintage@DESKTOP-QI6V0QI: /mnt/c/Users/Love2/Downloads/ab224sf_assign2_ex$ java WebServerV2 49860 public
Server source file exists!
Detailed path: /mnt/c/Users/Love2/Downloads/ab224sf_assign2_ex/public
Assigned a new client to a separate thread.

Host: localhost:49860, Method: GET, Path: /, Version: HTTP/1.1
Server request file exists!
Requested item is a directory!
Client: 0.0.0.0/0.0.0.0:49860, Version: HTTP/1.0, Response: 200 OK, Date: 2022/01/11 14:18:18, Server: my_web_server, Content-Length: 236, Connection: close, Content-Type: text/html
Assigned a new client to a separate thread.

Host: localhost:49860, Method: GET, Path: /favicon.ico, Version: HTTP/1.1
Server request file does not exist!
Client: 0.0.0.0/0.0.0.0:49860, Version: HTTP/1.0, Response: 404 Not Found, Date: 2022/01/11 14:18:18, Server: my_web_server, Content-Length: 127, Connection: close, Content-Type: text/html
```

**How to start:** Start by implementing a server that accepts connections and returns a predefined (valid) HTTP/HTML response. Once you can connect to the server with a web browser and view the predefined HTML page, you should add support for reading HTML files from disk and returning these. Finally, add support for images (binary files). If a user requests a specific directory, e.g. `/` or `user1/pages` and this directory contains an `index.html` file, then this file should be returned. Note that the ending `/` is optional, so `http://myserver.com/user1` and `http://myserver.com/user1/` should both return the `index.html` file in the `user1` directory. Prepare a few HTML and PNG files in a directory hierarchy (e.g. `/memes/rarepepe.png`) and use this to test your web server. Make sure to include both `index.html` files as well as other named HTML files, e.g., `myfile.html`. Your report should include screenshots of the browser window when you request a named HTML page, an image, and a directory (with an `index.html` file).

Your web server should be robust -- you have to handle possible exceptions. **Document which exceptions you handle and why.** Do NOT allow a user to escape the public directory and access other files on the computer.

## Problem 2

The web server you implemented in Problem 1 only supports the HTTP response 200 OK . You should now add support for the following response types:

- 302
- 404
- 500

When implementing 302, hardcode a specific url that redirects to any resource of your choosing. Include a description of how you tested each of these response codes with screenshots of the result in your report.

Include in the instruction text file how to trigger each of these.

## Problem 3

After completing Problem 1 and 2 you should configure the web server to handle a simple login page. You need to create a new HTML file with a [form](#) that handles POST action. You also need to have an external file which contains the login information it can be for example a JSON or txt. If the received information is correct then the server should send 200 OK if not then 401 Unauthorised.

Add support for the HTTP POST method. Add a webpage with a form that allows a user to upload PNG images somewhere in your directory hierarchy. We recommend implementing a multipart-form parser.

# Point distribution for the programming part

---

**Prerequisite for getting points: That it compiles and runs**

**Some items are marked with a "\*", these are mandatory to pass the assignment in addition to getting enough points**

P1:

- \*Working HTTP server with HTML+HTM support - 3p
- Image support - 2p
- Safety against escaping the root - 2p
- \*Exception handling - 2p
- \*Program arguments in correct order - 1p
- Console output - 1p
- 

P2:

- 302 Implementation - 2p
- \*404 Implementation - 1p
- 500 Implementation - 2p

P3:

- \*HTTP POST login - 3p
- HTTP POST Implementation with a supporting webpage - 4p

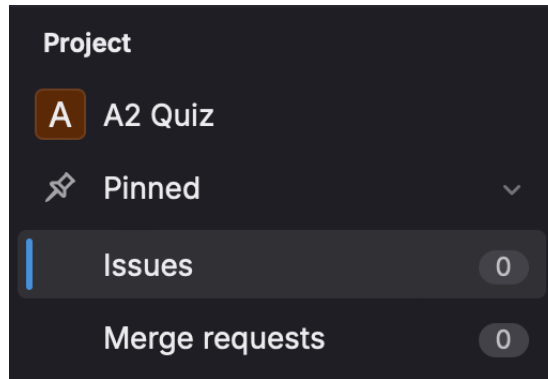
**Getting the points for the mandatory task will give you an E. Getting full points will give an A. Other grades are distributed between these.**

**Points are deducted from the total for things such as poor code quality or other strange things with your program or version control, so even if you do all the mandatory tasks you could still get a F or Fx.**

---

# Submission

All submissions are done in GitLab. To submit, please create a new **issue** in your repository and select the TA for your group as an **Assignee** when prompted:



Your repository should contain the following:

- A **PDF** report containing a brief overview of each problem, along with the things specifically outlined to write about.
- All **.java** files required to run the program.
- A public folder containing resources.
- A **README.MD** file containing instructions on how to run the program

Make sure to have a good Git **commit history** to demonstrate your collaboration. Make the commit names meaningful! Example guide:

<https://www.freecodecamp.org/news/how-to-write-better-git-commit-messages/>

Please agree on the final submission before you submit.

In the **README**, include a summary of the instructions about how to run the program. In the **report** of what each of the participants of the group contributed and how you split the workload (in percent). For example, student\_name\_1: 45%, student\_name\_2: 55%. If these differ too much, the participants might receive different grades for the assignment. If one of the members in the group does not contribute at an acceptable level, please notify the teacher as soon as possible.