

Assignment 2: Web Server

Web Server

The second assignment is designed to familiarize yourself with a Java networking API. Your task is to develop a small web server with Java that is capable of serving static content. By carrying out this assignment, you will learn how to use server sockets with HTTP protocol; later you will learn means to work with underlying protocols (on both server and client side).

Reference material

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

Required downloads

- [PuTTY client](#) (for Windows users)

Note: This assignment is performed in groups of 2 people, see submission instructions at the bottom of this page for details. The groups can be found on a [separate page](#).

Hint (optional): you can use this assignment to familiarize with version control systems, such as Git and SVN. These systems allow editing the same code from multiple computers; they are used by most of the existing software companies. For simplicity, I recommend using [SourceTree](#) GUI for Windows.

Problem 1

You can use your code from the first assignment. Implement a simple web server using `ServerSocket` class that is able to accept multiple client connections on some predefined port (e.g., 8888). Your server is supposed to work in a loop until terminated manually. The server should serve `.html` pages and `.png` images from a local directory for corresponding GET requests. The server response should contain the contents of the requested file as well as corresponding headers, etc. *Hint:* read on how web server forms a response.

Note: You are not allowed to use third party libraries or in-build Java classes (such as `com.sun.net.httpserver`) that simplify work with http servers. Of course, feel free to use the standard libraries for sockets, input/output, etc.

A possible solution algorithm for this task may look as follows: implement a program that accept connections and return some string, then change this string into a valid HTTP response, then implement `.html` file reading / transfer support, then add support for image files.

If user requests a directory (e.g. `dir1/subdir1`) and this directory contains a file called `"index.htm"` or `"index.html"` (check both), its contents should be returned in response. Mind that a directory in user request may or may not have an ending slash. You need to handle both cases.

To test your server prepare a few (2-3) small `.html` files and a few (2-3) `.png` images. Then organize these files in a hierarchy of directories with some files. For example: `"dir1"`, `"dir1/subdir1/image1.png"` and `"dir2/subdir2/index.htm"`.

Try to access your running server program from web browser. E.g., if you specified port 8888, try `http://localhost:8888` in the browser address bar. Include browser **screenshots** with server responses to a request with an `.html` page, an image, and a directory in your report.

Problem 2

By now your server should produce at least one response, namely "200 OK". The new task is to add code support for the following 3 server responses: 403, 404, 500. You may make up your own access policy for the web server to demonstrate the use of 403. *Hint:* in any case you may want to restrict user access to a root directory. Test the behaviour of your server and make it produce the mentioned 3 responses. Include corresponding web browser **screenshots** in your report.

VG-task 1: add code support for at least 8 more server responses. Feel free to choose any responses other than those specified in problem 2. List the implemented server responses in the report and add screenshots demonstrating their usage. *Hint:* think carefully how you are going to test these server responses; some of them are not easy to reproduce.

VG-task 2: implement a support for POST and PUT requests. To test this scenario add a simple web page allowing to upload .png files to your server. Explain the difference between POST and PUT requests in your report.

Problem 3

Instead of using web browser, repeat your requests with a telnet client (use PuTTY in Windows). You will have to type some HTTP related commands by hand this time. Include **screenshots** of few testing cases made with the terminal emulator in your report. Explain the results of requesting an image through a telnet client.

Submission

The second assignment is done **in groups** of two people. The groups can be found on a [separate page](#). Name your submission accordingly, e.g. aa222bb_cc333dd_assign2.zip.

In the beginning of your report, write a short summary of work performed by each group participant (one paragraph per person). Also evaluate your work compared to the work of your partner in percentage. For example: student_name_1: 45%, student_name_2: 55%. Mind that if these percentages differ by a big amount, the group members will receive different grades.


Note: if your group member is not able to provide any input for the assignment, you should notify the teacher in advance.

Both people from the group should make a Moodle submission and **it should be identical!** In other words, you need to reach a consensus in terms of your work evaluation. If two group members submit reports with two different solutions, their grade will be drastically decreased.

For other information read the general [submission instructions](#).

[optional] If you like to give an opinion about an assignment (e.g., what could be improved), you can use an [anonymous form](#).

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Tuesday, 28 February 2017, 11:55 PM
Time remaining	21 days 10 hours
Last modified	-
Submission comments	 Comments (0)

Add submission

