



Linnéuniversitetet

Kalmar Vaxjö

Report

Assignment 2

IDV701



Author: Yishu Yang, Noah
Carlsson

Semester: Spring 2024

Email yy222cm@student.lnu.se
nc222hx@student.lnu.se

Contents

1 Problem 1	I
1.1 Discussion	III
2 Problem 2	IV
2.1 Discussion	V
2.2 VG 2	V
2.2.1 Discussion	V
3 Problem 3	V
3.1 Discussion	VIII
4. Contribution and Workload	VIII

1 Problem 1



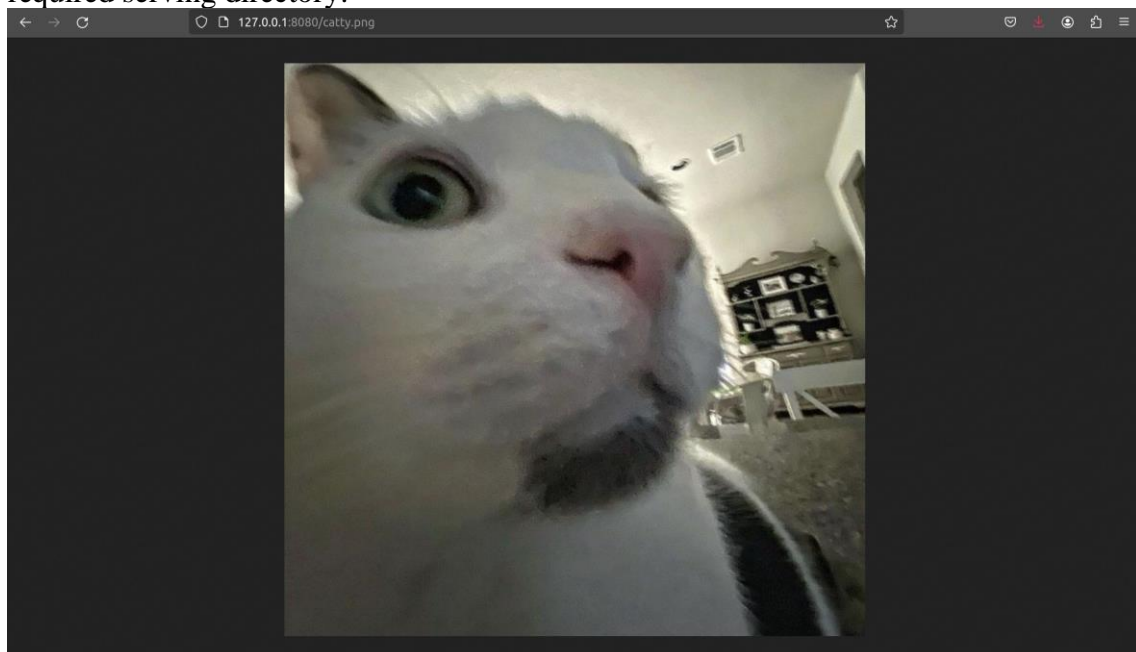
My fun page

This is a fun page, with an image!

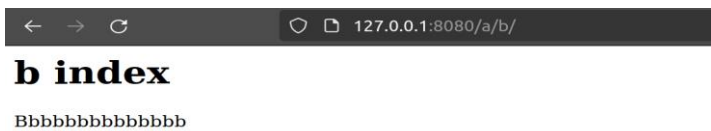


And a link to a page about [clowns](#)

This is the screenshot of the browser window when we are requesting a named HTML page on the listening port 8080 with a successful connection. It belongs to the named HTML files. It is from the /public folder, or directory, meaning it belongs to the required serving directory.



This is the screenshot of the browser window when we are requesting a PNG file image on the listening port 8080 with a successful connection. Also, it is prepared by ourselves storing in a directory hierarchy under a folder: memes. The correct output of the web browser shows that the local directory in our disk could function successfully.



This is the screenshot of the browser window when we request a directory with an index.html file on the listening port 8080 with a successful connection. Here the index.html file is shown on the website, which belongs to the b index. It is from the given serving directory: /public and under the hierarchy: b folder.



c index

ccCCCCCCCCCCCC

This is the screenshot of the browser window when we request a direct index.html file on the listening port 8080 with a successful connection. This index.html file is contained in ./public/a/b/c/index.html.

```
Requested item is a directory! (Defaulting to index.html within folder)
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 145

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/b/b.html, Version: HTTP/1.1
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 249

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/b/c.html, Version: HTTP/1.1
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 122

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/b/c, Version: HTTP/1.1
Requested item is a directory! (Defaulting to index.html within folder)
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 119

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/b/c/c.html, Version: HTTP/1.1
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 233

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /nosuchpage.html, Version: HTTP/1.1
Server request file does not exist!
Response: 404 Not Found, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 22

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/fail.html, Version: HTTP/1.1
Server request file does not exist!
Response: 404 Not Found, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 22

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/b/c/d, Version: HTTP/1.1
Server request file does not exist!
Response: 404 Not Found, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 22

Assigned a new client to a separate thread.
Host: 127.0.0.1, Method: GET, Path: /a/b/c/noimage.png, Version: HTTP/1.1
Server request file does not exist!
Response: 404 Not Found, Version: HTTP/1.0, Date: 2024-02-19, Content-Type: text/html, Content-Length: 22
```

```
noah@noah:~/School/ComputerNetworks/A2/assignment2$ p
OK: Main index page
OK: Named page
OK: Named page
OK: Clown PNG
OK: Bee PNG
OK: World PNG
OK: Index a
OK: Page a
OK: Fake page b
OK: Index b
OK: Page b
OK: Fake Page c
OK: Index c
OK: Page c
OK: Page fail
OK: Page in dir fail
OK: Dir no index fail
OK: Image fail
noah@noah:~/School/ComputerNetworks/A2/assignment2$
```

This shows the testing result of testing the provided Python script on our web server with a detailed demonstration of command windows for both the web server and the client, where the left one is for web server connection and the right one is for client output.

Notation: we are pretty safe because it will always look for the files within the directory the WebServer is run on, which is the /public and our local directory. Hence no one can escape the public directory and access other information on the computer.

Here is detailed explanation.



When we connected to the web server on 8080 with the path to the served directory .\public, we added two dots like ".." after the localhost url: "127.0.0.1:8080/". This input of url is going to the upper folder of the public and trying to access our local computer resources. But the warning of "This is the main index page!" appears, which means we cannot go to the upper folders and access those resources. We could only go down to see the documents under .\public. This is the evidence for preventing escape from the root.

1.1 Discussion

Brief Overview:

We have constructed a simple web server allowing multiple client connections on a specifically required listening port number by majorly implementing two java files WebServer.java and SocketHandler.java, where the client could input the listening port number independently and the web server uses the ServerSocket class. WebServer.java is for building a basic web server that continuously listens on the required port and ensures a valid connection to the specific client socket. Once the connection has been successfully made, the web server will deliver the client to the socket handler, where SocketHandler.java works for serving content and offering the requested files. Server content should be at least HTML and PNG files from a local directory using the GET method in the HTTP protocol.

Apart from the requested file, the correct and relevant headers should also be included until the client terminates it manually with the keyboard shortcut ctrl-c or ctrl-d. For the local directory containing HTML and PNG files, we should use the /public directory with a relative path emphasized in our Java files, where we write the port program

arguments and then the emphasized /public path. The /public directory could be seen as the serving directory from which HTML files and images would be served in our connected web server.

Specific things to write about.

In summary, the pivotal steps could be concluded as the following:

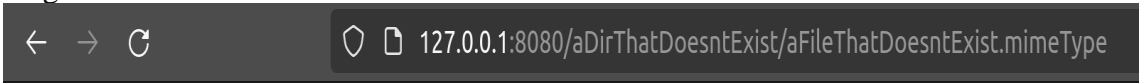
1. Construct a web server that allows continuous connections on the required port and returns a predefined HTML/HTTP response handled by the socket handler.
2. For verifying, connect to the server with a web browser and confirm the predefined HTML page.
3. Add sustenance for reading HTML files from the hierarchy disk and return these.
4. Add support for images (binary files). If the required directory includes an index/html file, this file should be outputted, where the ending “/” is also optional.
5. More HTML and PNG files should be prepared in a directory hierarchy (which may be included under /public) and index.html and named HTML files should be included.
6. Finally, use the provided Python script to confirm that the construction of the web server works.

2 Problem 2

404 Not Found:

```
} else {  
  
    // File doesn't exist - 404 Not Found  
    writer.println(x:"HTTP/1.0 404 Not Found");  
    writer.println(x:"Content-Type: text/html");  
    writer.println();  
    byte[] htmlResponse = "<h1>404 Not Found</h1>".getBytes();  
    writer.write(htmlResponse);  
    System.out.println(x:"Server request file does not exist!");  
    System.out.println("Response: 404 Not Found, Version: HTTP/1.0, Date: " + LocalDate.now()  
        + "Content-Type: text/html, Content-Length: " + htmlResponse.length);  
  
}
```

This is how we handle the HTTP response of 404 Not Found with detailed codes in the else condition apart from the HTTP 200 OK. The triggering else condition here is majorly for files not existing, where after printing out the basic information of 404 Not Found and the content type we use `getBytes()` to write out the response message specifically. This includes a response of 404, version, date, content type, and content length as well as a reminder that the file does not exist.



← → ↻ 127.0.0.1:8080/aDirThatDoesntExist/aFileThatDoesntExist.mimeType

404 Not Found

This is the evidence of the correct depiction of triggering a 404 Not Found HTTP response in a web browser. Below is a detailed explanation.

Here we type a non-existent file under the localhost 127.0.0.1 and listening port: 8080 based on our web server. Since the “404 Not Found” is depicted, we concluded that 404 Not Found has been correctly implemented.

2.1 Discussion

The basic web server in problem 1 only treats the condition of the HTTP response 200 OK. For the mandatory part, we should deal with the case of the HTTP response 404 Not Found.

2.2 VG 2

500 Internal Server Error:

```
private void internalServerErrorResponse(PrintStream writer) throws IOException {
    writer.println(x:"HTTP/1.0 Internal Server Error");
    writer.println(x:"Content-Type: text/html");
    writer.println();
    byte[] htmlResponse = "<h1>Internal Server Error</h1>".getBytes();
    writer.write(htmlResponse);
    System.out.println(x:"An Internal Server Error have occurred!");
    System.out.println("Response: Internal Server Error, Version: HTTP/1.0, Date: " + LocalDate.now()
        + ", Content-Type: text/html, Content-Length: " + htmlResponse.length);
}
```

This is how we handle the HTTP Response 500 Internal Server Error, where a function called `internalServerErrorResponse` is created and used in the else if condition apart from the HTTP Response 200 OK in the main body. In addition, the case of HTTP Response 404 Not Found is in the other else condition. Here in this function, we print out the error first, which is followed by using `“getBytes()”` to locate the 500 response successfully. Thus we could print out the whole information including Response type, Version, Date, Content-Type, and Content-Length.



This is the evidence of the correct depiction of triggering a 500 Internal Server Error HTTP response in a web browser. Below is a detailed explanation.

Here we insert a jpg file called “1.jpg” into the /public default directory for the webserver to serve. Since our web server doesn’t support jpg files it will report the 500 Internal Server Error. This is because the requested file “1.jpg” truly exists in our directory:”/public”, but our web server doesn’t support this file format, which means a 500 Internal Server Error HTTP Response happens and shows.

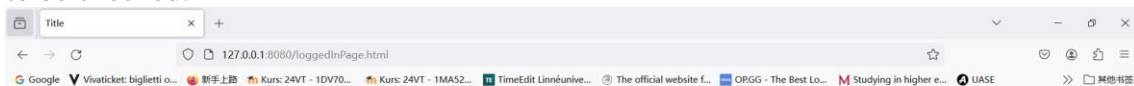
2.2.1 Discussion

For the additional part, we should deal with the case of the HTTP response 500 Internal Server Error, where the problem is the conflict between the existing jpg file and the jpg file being unsupported in our web server.

3 Problem 3



Here is the new HTML file named "login.html". We created a simple login page with a form that handles POST action. We use the GET method to create this login page with a path being /login.html. The next few screenshots are implemented by POST action to handle credential checking. The login information will be connected to the outside file to be checked.



LOGGED IN!

状态	方法	域名	文件	发起者	类型	传输	大小	耗时	接收时间	接收大小
200	POST	127.0.0.1:8080	loggedInPage.html	document	html	224 字节	159 字节	19 毫秒	0 毫秒	160 字节
404	GET	127.0.0.1:8080	favicon.ico	FaviconLoader.aspx.mjs:175 (img)	html	已缓存	22 字节	0 毫秒	0 毫秒	0 字节



We could see that when we log in with the correct credentials, the webpage will pop out the information "LOGGED IN!". Also from the network of the web browser, we could see that this involves a 200 OK POST action, where the 404 Not Found is meaningless

because it is for the icon of the web browser. Here the path is /loggedInPage.html.



401 Unauthorised

A screenshot of a network tool interface showing a list of HTTP requests. The table has columns for status, method, host, path, file, status code, type, and size. Two requests are visible: a POST request to /loggedInPage.html with status 401, and a GET request to /favicon.ico with status 404.

状态	方法	域名	文件	发起者	类型	传输	大小
401	POST	127.0.0.1:8080	loggedInPage.html	document	html	79 字节	25 字节
404	GET	127.0.0.1:8080	favicon.ico	FaviconLoader.js.mjs.125 (img)	html	已缓存	22 字节

We could see that when we log in with the wrong credentials, the webpage will pop out the information "401 Unauthorised". Also from the network of the web browser, we could see that this involves a 401 Unauthorised POST action, where the 404 Not Found is meaningless because it is for the icon of the web browser. Here the path is /loggedInPage.html.

Notation: If someone tries to access the loggedInPage.html directly, he/she will also get a 401 Unauthorised response. This is implemented to prevent someone from skipping the login process.

A screenshot of a Windows PowerShell window displaying server logs. The logs show four separate client connections. The first connection is a GET request to /login.html resulting in a 200 OK response. The second connection is a GET request to /favicon.ico resulting in a 404 Not Found response. The third connection is a POST request to /loggedInPage.html resulting in a 200 OK response, indicating a successful login. The fourth connection is a POST request to /loggedInPage.html resulting in a 401 Unauthorised response, indicating invalid login credentials.

```
024-2-Linnaeus University Sweden\1DV701\Assignment2\assignment2\memes

Assigned a new client to a separate thread.
Exception in thread "Thread-0" java.lang.NullPointerException: Cannot invoke "String.indexOf(String)" because "<local3>" is null
    at SocketHandler.run(SocketHandler.java:43)

Assigned a new client to a separate thread.
Host: 127.0.0.1:8080, Method: GET, Path: /login.html, Version: HTTP/1.1
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-20, Content-Type: text/html, Content-Length: 550

Assigned a new client to a separate thread.
Host: 127.0.0.1:8080, Method: GET, Path: /favicon.ico, Version: HTTP/1.1
Server request file does not exist!
Response: 404 Not Found, Version: HTTP/1.0, Date: 2024-02-20, Content-Type: text/html, Content-Length: 22

Assigned a new client to a separate thread.
Host: 127.0.0.1:8080, Method: POST, Path: /loggedInPage.html, Version: HTTP/1.1
Login Successful!
Server request file exists!
Response: 200 OK, Version: HTTP/1.0, Date: 2024-02-20, Content-Type: text/html, Content-Length: 159

Assigned a new client to a separate thread.
Host: 127.0.0.1:8080, Method: POST, Path: /loggedInPage.html, Version: HTTP/1.1
Login Credentials are Invalid!
Response: 401 Unauthorised, Version: HTTP/1.0, Date: 2024-02-20, Content-Type: text/html, Content-Length: 25
Invalid Login Details!
```

Here is the entire information in the command window when we connect to the listening port and do the login page. The 200 OK Response in the first paragraph is for opening the login page. The 404 Not Found in the second paragraph is the meaningless icon request. Then the 200 OK Response and 401 Unauthorised Response in the 3rd and 4th paragraph are respectively for the successful and failed credential checking. We print out the Host, Method, Path, Version, existing notation, Response, Response Version, Date, Content-Type, and the Content-Length for all paragraphs.

3.1 Discussion

For the mandatory part of the P3, we deal with the issue of *HTTP POST login, which means we should configure the webserver to handle a simple login page. We first created a new HTML file and then inserted a form that handles POST action into the file.

In addition, an external file is needed with the login information like a JSON or Txt file, where correct input returns 200 OK and wrong input returns 401 Unauthorized.

4. Contribution and Workload

Yishu Yang: 50%

Noah Carlsson: 50%