

# CS5001 – Practical 3 Report

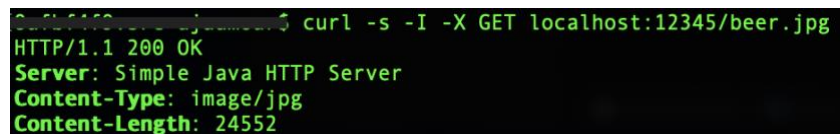
Matriculation id: 150014151

Advanced Requirements Attempted:

- Returning of binary images (GIF, JPEG and PNG)
- Multithreading – support multiple concurrent client connection requests up to a specified limit
- Logging – each time requests are made, log them to a file, indicating date/time request type, response code etc.
- Supporting other methods in addition to GET and HEAD

## Binary Images Support

Support for binary images was achieved by specifying the content type in the response header. This is done in the *getContentType* method in the *ConnectionHandler* class. After analysing the file extension of the requested file, an appropriate header is returned by the method. The header for text files such as html files is “text/html”, whereas the header for binary image files such as JPG is “image/jpeg”, PNG is “image/png” and GIF is “image/gif”. A screenshot of the header of a GET request for the “beer.jpg” file can be found below:



```
root@kali:~/java-journal$ curl -s -I -X GET localhost:12345/beer.jpg
HTTP/1.1 200 OK
Server: Simple Java HTTP Server
Content-Type: image/jpeg
Content-Length: 24552
```

The binary image itself is sent over to the client by first converting it to an array of bytes before sending it through a *BufferedOutputStream*. The method performing the conversion is *fileDataToBytes* in the *WebUtil* class, while the method sending the data over to the client is *sendHttpResponseContent* in the *HttpResponse* class. Both of these methods are used to send the body of the HTTP response to the client for text files (e.g. html pages) and binary image files (e.g. jpg images), thus reducing code duplication.

This implementation can be tested by viewing the image in a web browser by requesting a GET method on an image file using the following:

- Visit the following URL: <http://localhost:12345/beer.jpg>
- Right-click on one of the images in “page2.html” and select “Open Image in New Tab”
- Run the following command in the terminal: “open [http://127.0.0.1:12345/tp\\_it.jpg](http://127.0.0.1:12345/tp_it.jpg)”

## Multithreading

Multithreading is implemented by extending the *Thread* class in the *ConnectionHandler* class. A main *run* method starts being executed whenever a new thread is created by calling the *start* method on an instance of the *ConnectionHandler* class.

The number of active threads is limited by a number specified by the user when creating the server. This limit is set by a third optional argument in the form of an integer and is enforced by using the *Thread.activeCount* method to ensure that the number of threads does not exceed the limit.

## Logging

The server logs each HTTP request received from a client in a log file called "*http\_requests\_history.log*", which can be found in the root directory of the project. The date and time of the request, its method type, the file requested, the response code and the connected client's IP address are all logged in the file every time a client connects, just before the server sends back the HTTP response. This is achieved through the use of the *FileWriter* and the *PrintWriter*, which are encapsulated into a separate *WebLogger* class. An example of the logged requests can be found below:

```
08/11/2019 20:30:57
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'GET' requesting file '/another/page3.html'. Server responded with code '200'

08/11/2019 20:31:00
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'HEAD' requesting file '/index.html'. Server responded with code '200'

08/11/2019 20:31:02
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'HEAD' requesting file '/index.html'. Server responded with code '200'

08/11/2019 20:31:05
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'HEAD' requesting file '/index.html'. Server responded with code '200'

08/11/2019 20:31:10
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'HEAD' requesting file '/index.html'. Server responded with code '200'

08/11/2019 20:31:20
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'GET' requesting file '/beer.jpg'. Server responded with code '200'

08/11/2019 20:31:23
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'GET' requesting file '/beer.jpg'. Server responded with code '200'

08/11/2019 20:31:32
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'DELETE' requesting file '/delete_me.txt'. Server responded with code '204'

08/11/2019 20:31:35
Client IP address: /0:0:0:0:0:0:1
HTTP method: 'DELETE' requesting file '/delete_me1.txt'. Server responded with code '204'
```

## Additional HTTP Method

An additional HTTP method was introduced: the DELETE method. This method allows the client to request a file to be deleted. It was implemented as per the W3C Protocol Standard for HTTP/1.1 [1]. To do so, the server attempts to delete the file using the *File.delete* method. If it succeeds, then it responds with a “204 No Content”, which is meant to occur when “The server has successfully fulfilled the request and that there is no additional content to send in the response payload body” [2]. If the server does not succeed with the file deletion, then it responds with a 404 error, assuming that the requested file could not be found. This is done in the *deleteAndGetResponseCode* method in the *HttpResponse* class.

A few plain text files have been added in the “www” directory to test the DELETE request, which can be tested by using the following command:

- To delete the “delete\_me.txt” file: `curl -s -X DELETE localhost:12345/delete_me.txt`
- To view the header of the response: `curl -s -I -X DELETE localhost:12345/delete_me1.txt`

## Other Improvements

- Default URL routing: when no specific file is requested by the client e.g. “localhost:12345”, then the server automatically responds with the default “index.html” page.
- Additional tests were added in the Tests/advanced directory to test the binary images support and the DELETE request support.
  - Test01 to Test05 test the headers of GET requests for binary images.
  - Test06 and Test07 test the header and the content of DELETE requests.
  - Note: The ROOT in the “progRun.sh” file for each of the new advanced tests was changed from “ROOT=\$TESTDIR/../../Resources/www” to “ROOT=\$TESTDIR/../../www” in order to run them locally.

```
Testing CS5001 Practical 3 (Web Server)
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - advanced/build : pass
* COMPARISON TEST - advanced/Test01_GET_Correct_Header_Status_Line_for_Existing_Image/progRun-expected.out : pass
* COMPARISON TEST - advanced/Test02_GET_Correct_Header_Content-Type_Line_for_Existing_JPG_Image/progRun-expected.out : pass
* COMPARISON TEST - advanced/Test03_GET_Correct_Header_Content-Type_Line_for_Existing_PNG_Image/progRun-expected.out : pass
* COMPARISON TEST - advanced/Test04_GET_Correct_Header_Content-Type_Line_for_Existing_GIF_Image/progRun-expected.out : pass
* COMPARISON TEST - advanced/Test05_GET_Correct_Header_Content-Length_Line_for_Existing_Image/progRun-expected.out : pass
* COMPARISON TEST - advanced/Test06_DELETE_Correct_Header_Status_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - advanced/Test07_DELETE_Request_Should_Not_Return_Response_Body/progRun-expected.out : pass
```

## References

- [1] W3C, “HTTP/1.1: Method Definitions,” 2004. [Online]. Available: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. [Accessed: 08-Nov-2019].
- [2] “204 No Content,” *Httpstatuses*. [Online]. Available: <https://httpstatuses.com/204>. [Accessed: 08-Nov-2019].