|  |
| --- |
| Report |

Computer networking –

Web Server

*Laboration 2*

|  |
| --- |
| *Authors:* Christofer Nguyen & Jonathan Walkden  *Subject:* Computer network  *Corusecode:* IDV701 |



# Introduction

The report will be containing the task for laboration 2. Laboration 2 goes through the process of creating a web server. In order to do this it is nessecary to understand the HTTP protocol and how it works. There is no startup code for this laboration all code is built from scratch using TCP server code from the previous laboration . The purpose of this laboration is understand how the a web server works through HTTP in requesting and responding. In this laboration there is also a collaboration in pairs.

# Partner Participation

In this section the participated work will be shown in percentage in relation to the time spend in this laboration.

People involved in this work:

1. Christofer Nguyen – 60%
2. Jonathan Walkden – 40 %

Christofer did most of the back end Java programming in order to get the web server up and running. He referenced the work done in his first assignment as the basis for the code in assignment 2. Christofer took the lead in implementing the functionality of the client thread and the way requests are handled within the web server. This includes implementing the http responses and header functionality.

Jonathan did not do as much raw coding as Christofer, instead working on the testing and commenting of the code. His responsibilities included working on task 3 of the assignment and ensuring everything was in working order overall. Although the implementation was handled by Christofer, both members fully understand the code and work that was done on the assignment.

# Assignment Task

In this section there will be a short presentation of the task and its solution. The solutions are described and illustrate through a set of pictures from the actual compilation.

Tools used in this laboration are the following:

* Eclipse to simulate the server
* Putty to simulate a telnet client

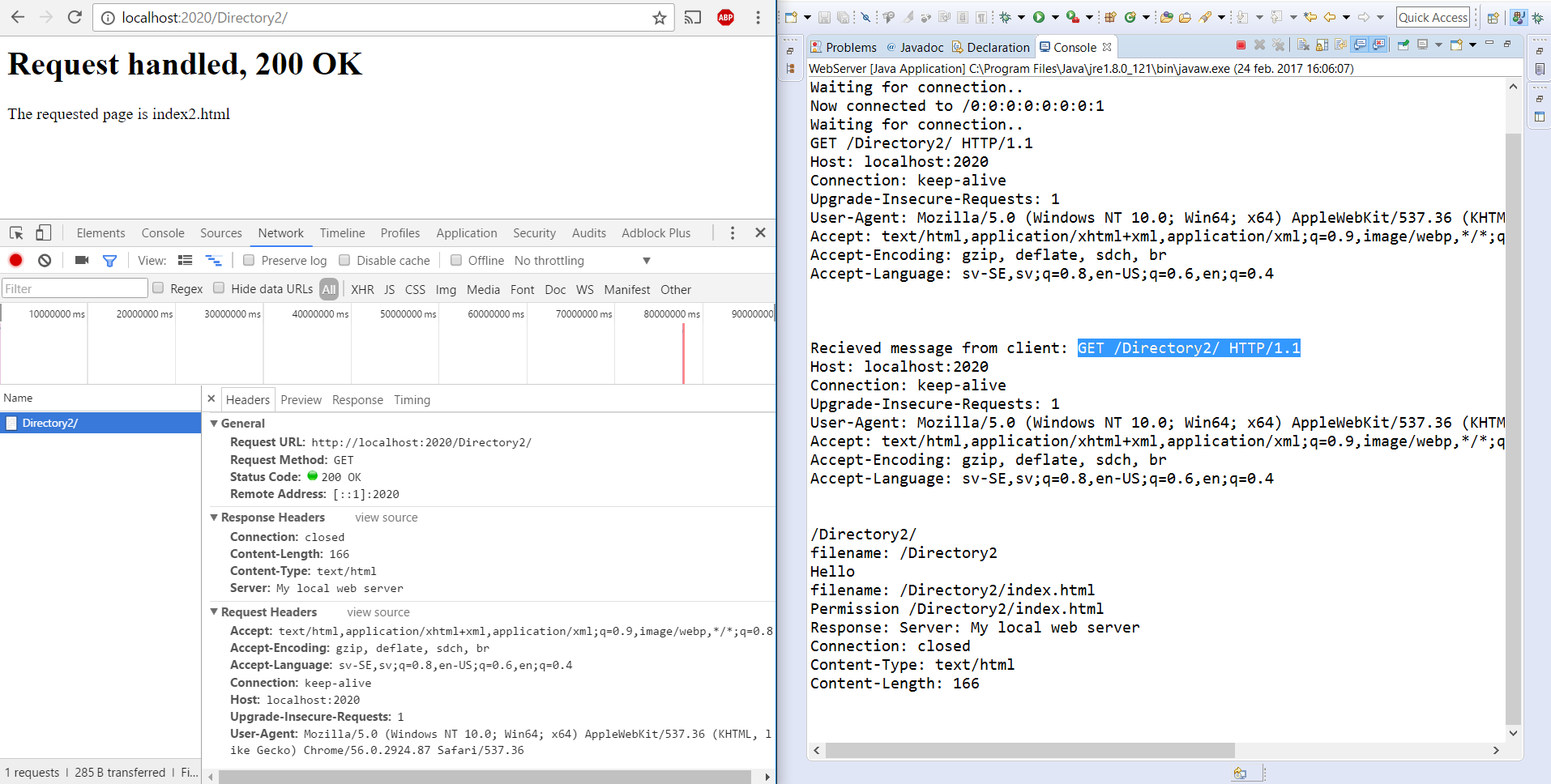
## Task 1

In this task we are to set up a web server using HTTP protocol. The server will be serving html and png files. There is no third party libraries or inbuilt java classes containing HTTP protocols. The server is instead simulating a HTTP protocol by applying request and responds. In the first task we are to implement a GET request and respond with correct HTTP format together with headers and body.

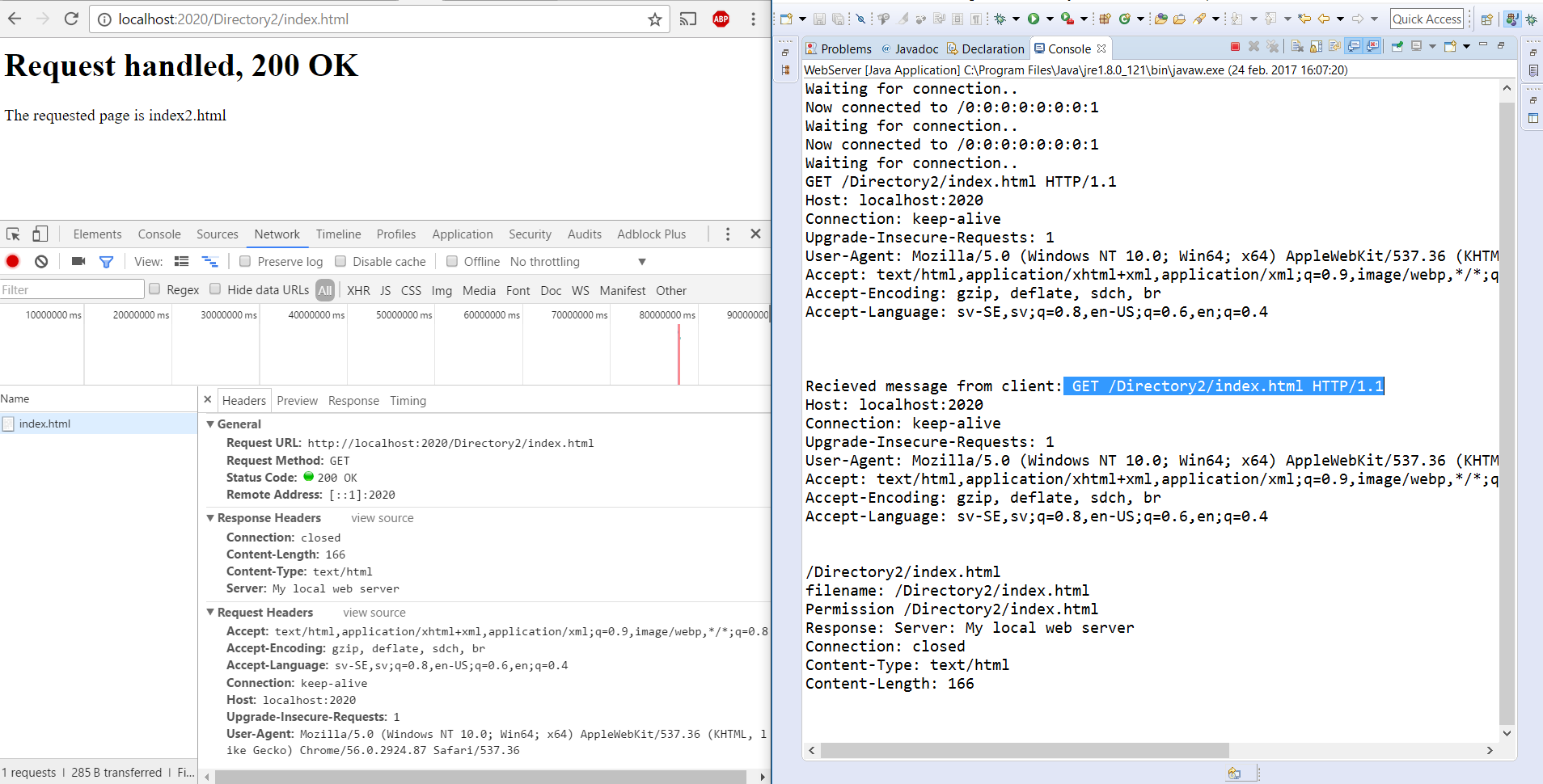
**Solution:**

The following headers are send together with the response:

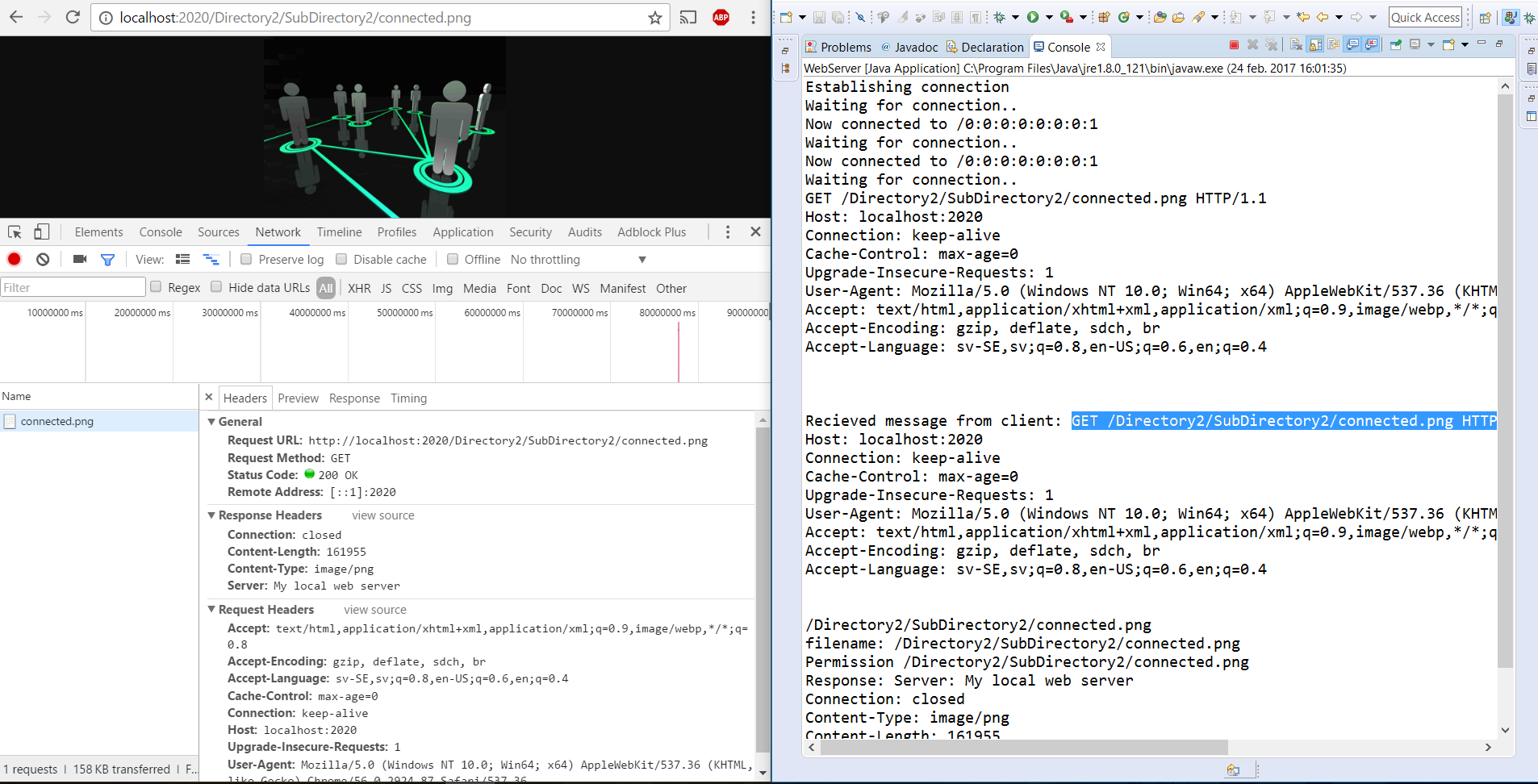
* Connection – Will indicate to the client if the connection is closed or alive
* Content – Length – Tells the browser how big the content of the file.
* Content – Type – What type of content
* Server



The above picture illustrates requesting a directory, the respond is a 200 OK because the specified directory contains the index.html file which is the root page for the directory. In cases where the specified directory does not contain a index.html file the response will be a 404 not found.



Here we are requesting a html file called index2.html, the respond is 200 OK and we are also displaying the requested file as seen above.

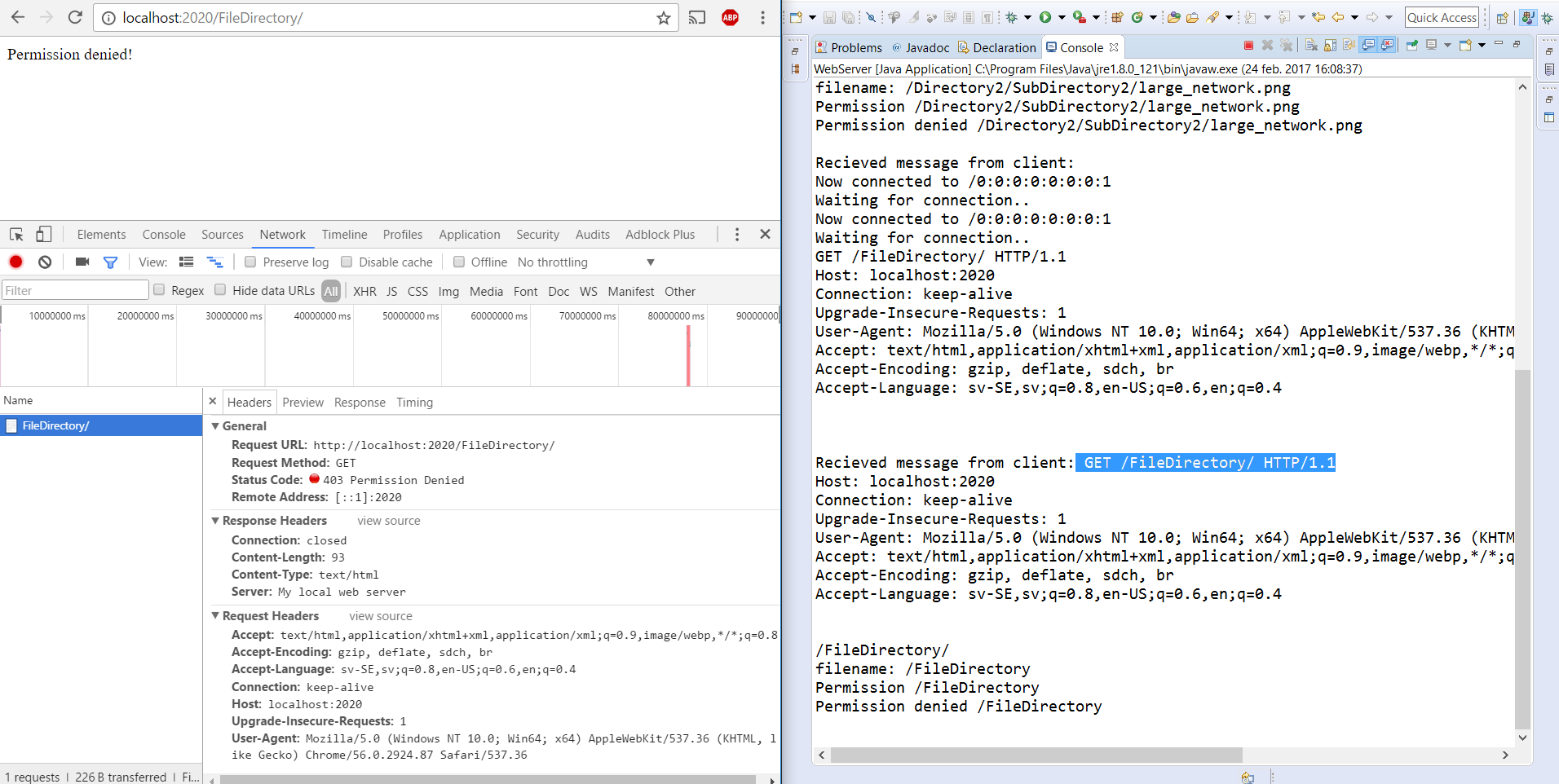


Finally we are also supposed to handle png files, as shown in the picture above the server is responding with a png file together with the corresponding headers returning a 200 OK to the client. The specified png file is not translated into a string instead it is send through the outputstream as raw bytes.

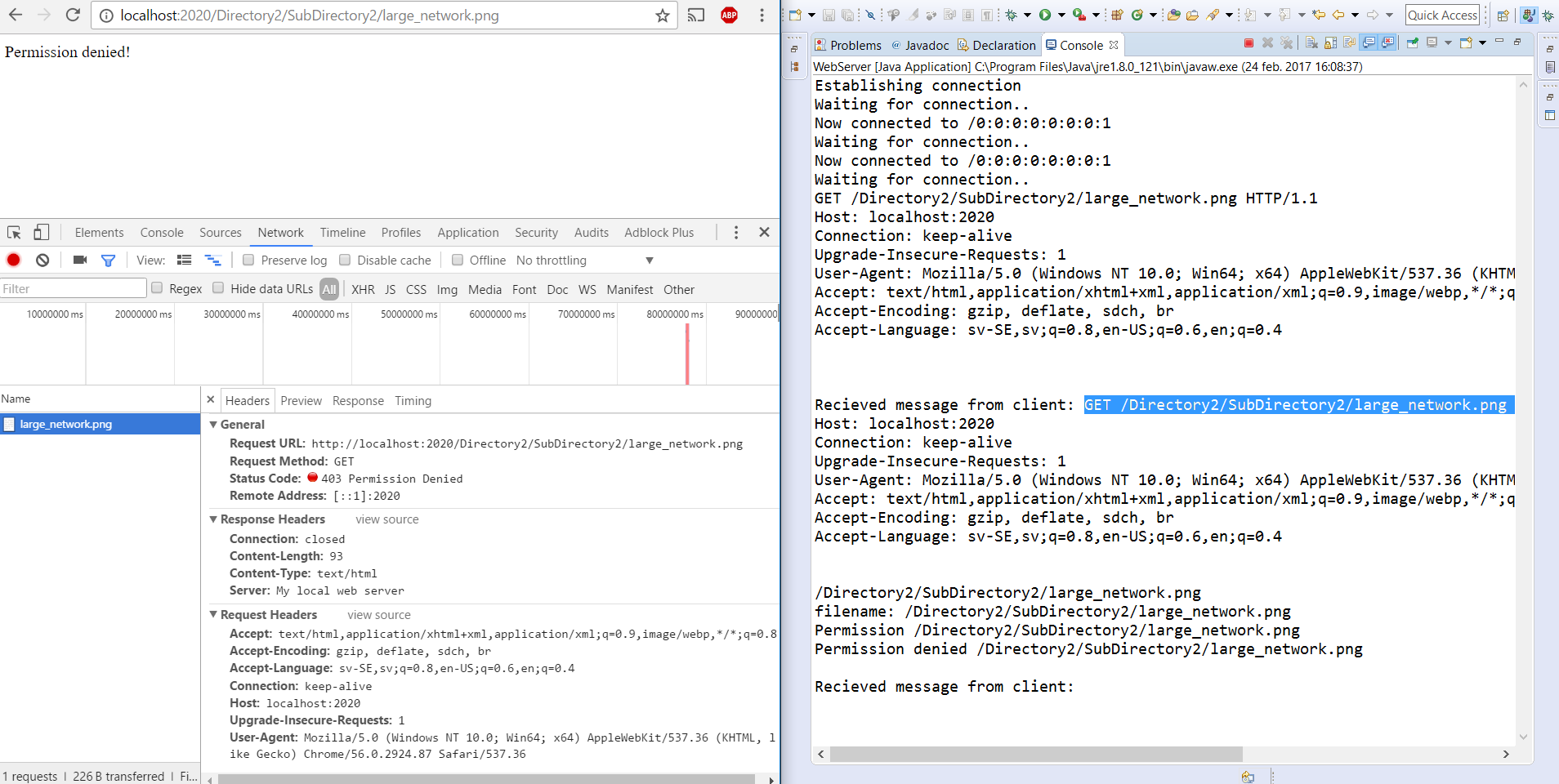
## Task 2

In the second task we are to implement responses outside of the 200 OK. The following responses where implemented:

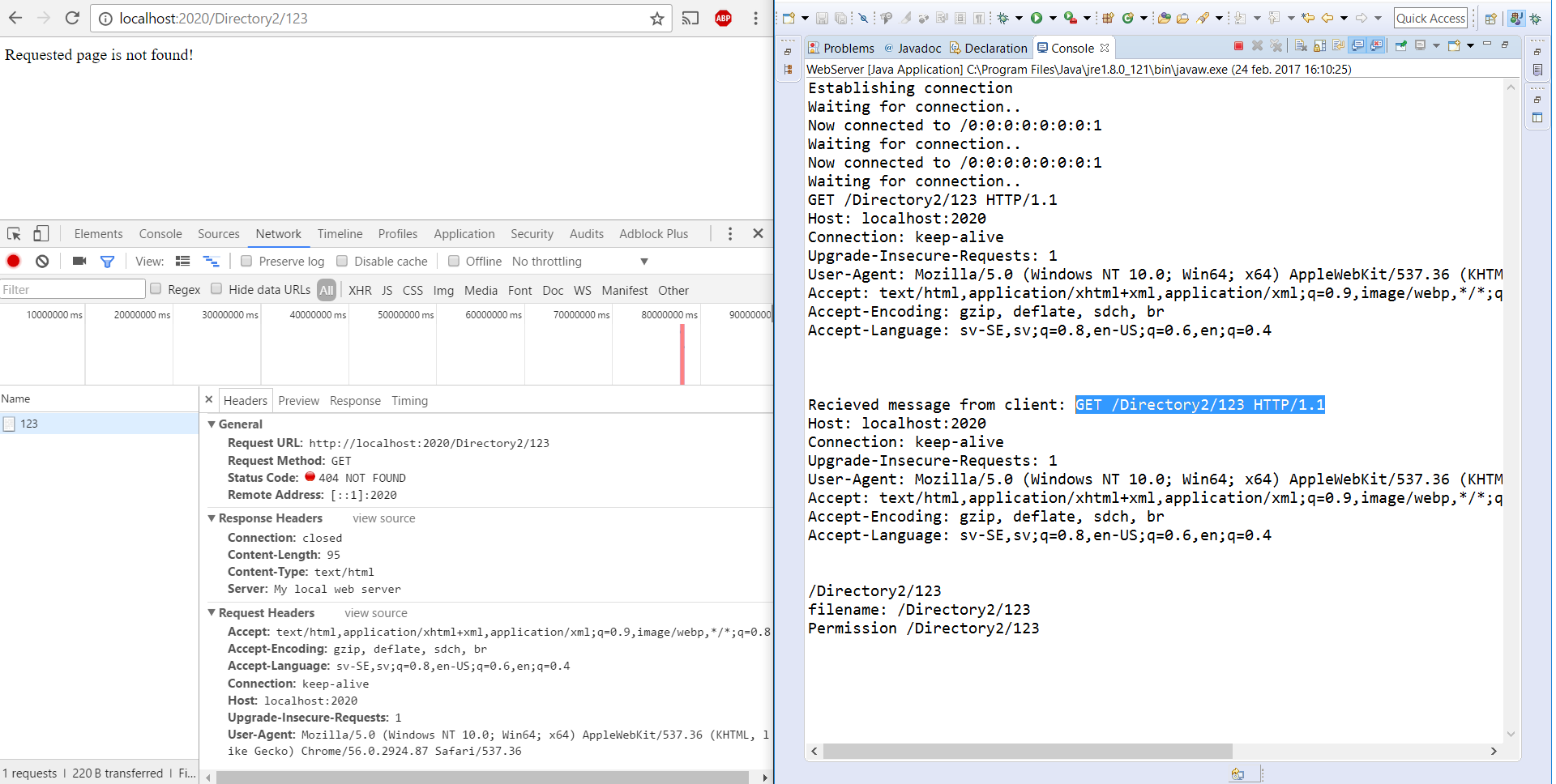
* 403 permission denied
* 404 not found
* 500 internal server error (default)

**Solution:** The picture below illustrates the 403 permission denied when trying to access root directory and any files inside of it. 

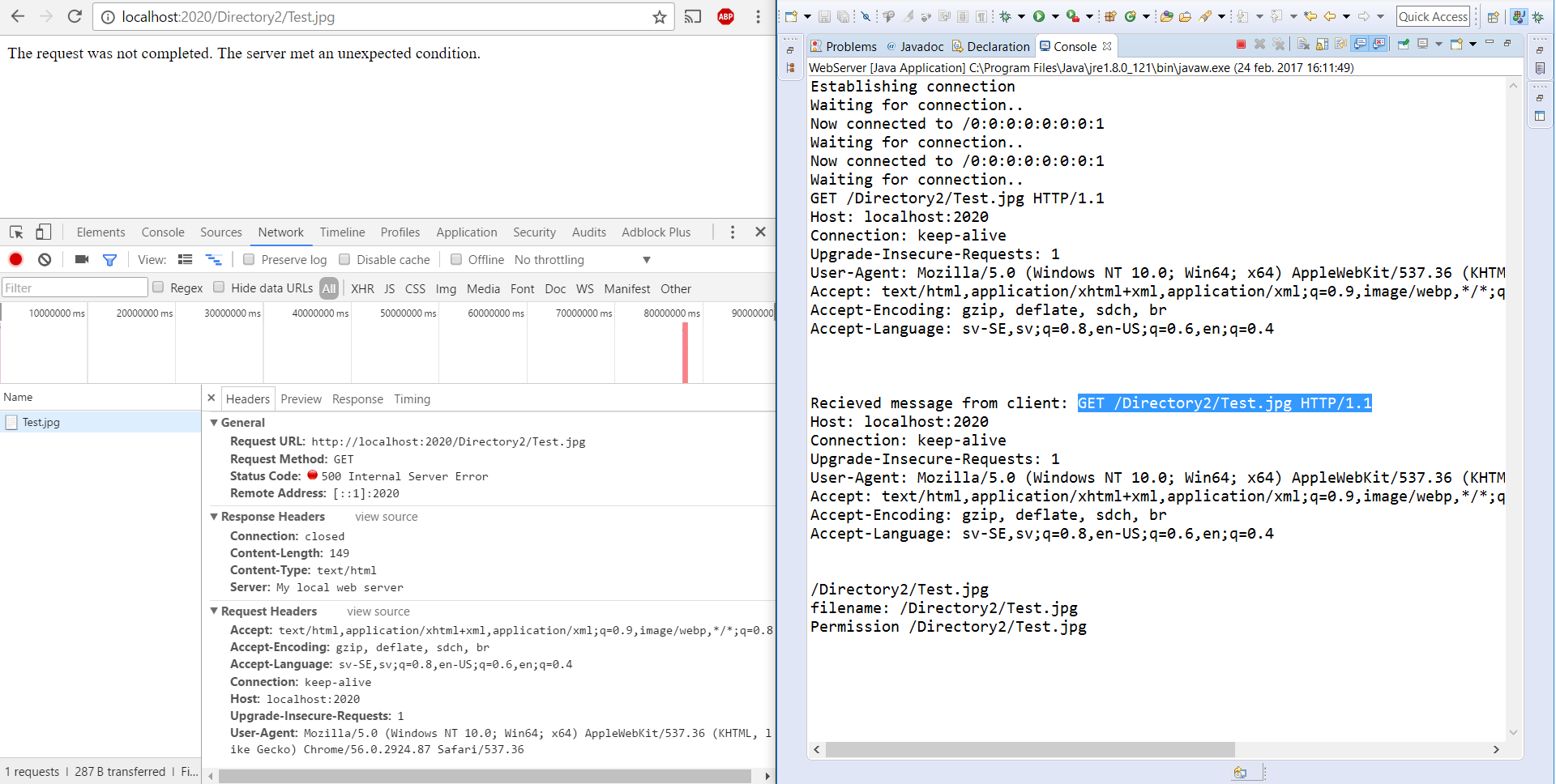
The response as seen in the status code above is a 403 permission denied. The permission denied will also send a entity body which is a html response showcasing the “Permission denied” when trying to access the given URL.



In second restriction the specified png file is none accesable therefor also returning a 403 permission denied.



The 404 not found occurs whenever the client tries to access a content that does not exist. In above scenario the client is trying to access “123” from directory2 which does not exist.

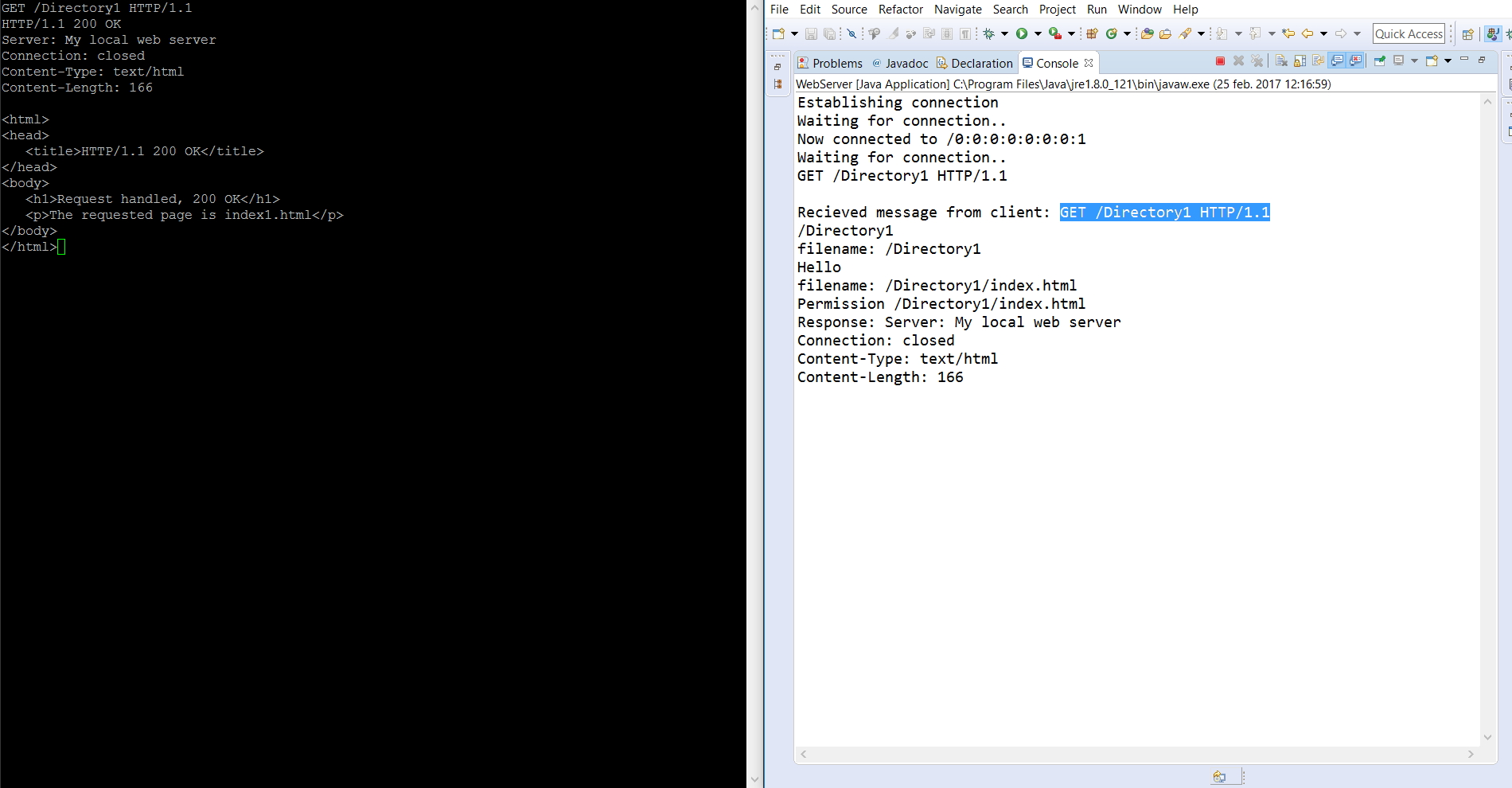


The final error response occurs as default whenever anything else in the web server can not handle from the request. The following scenario shown above is when the client wants to display a .jpg file which is not recognized by the server.

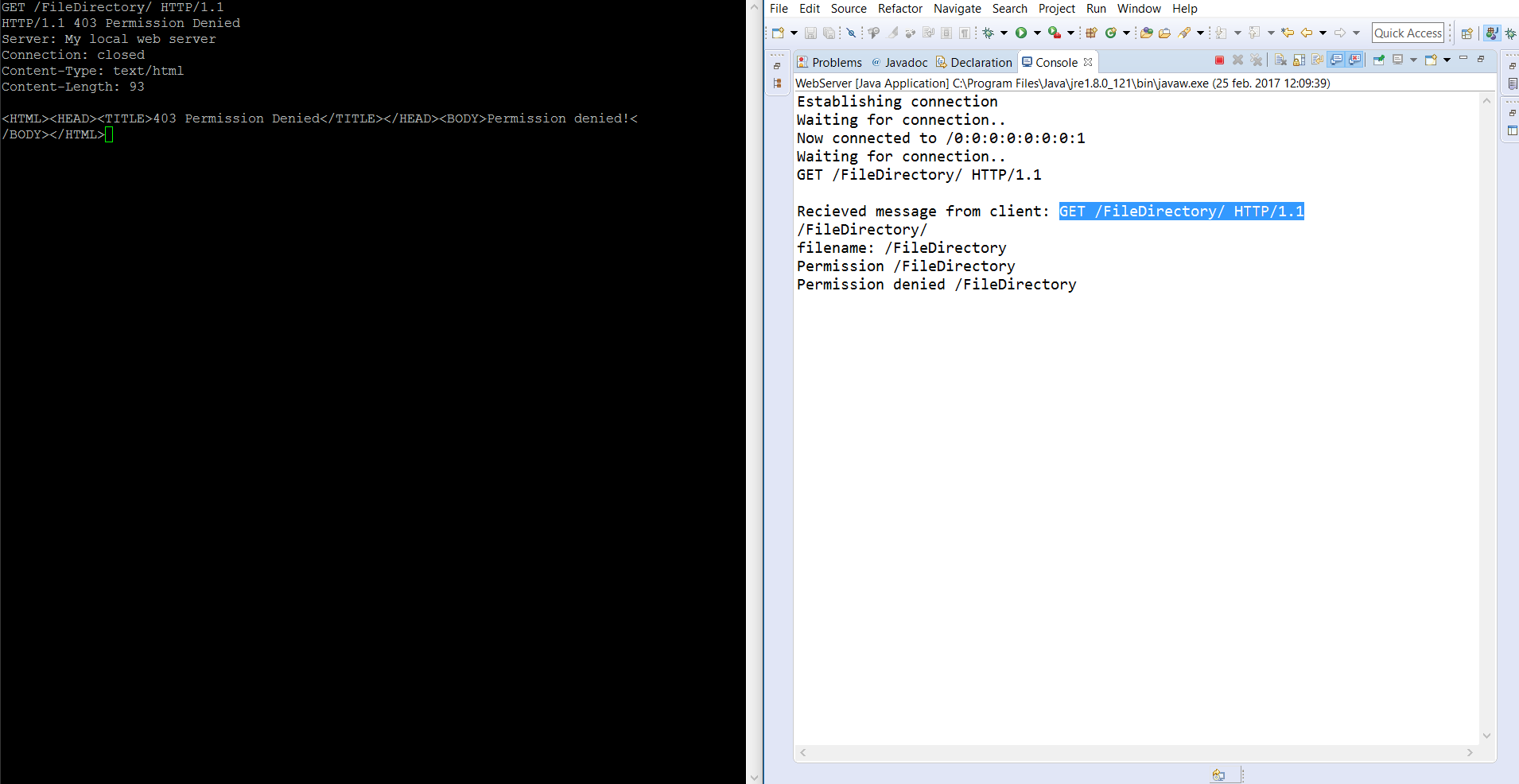
## Task 3

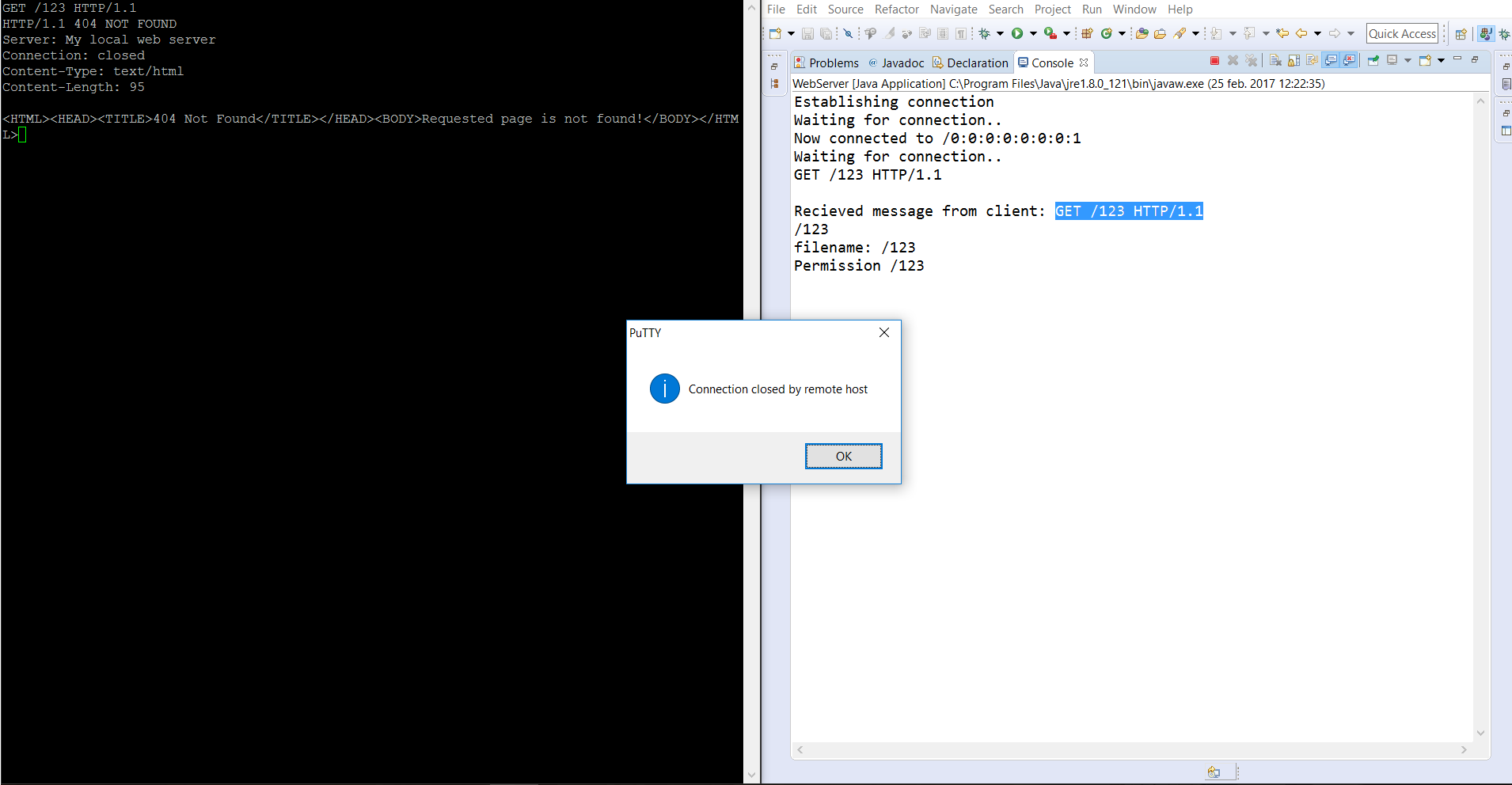
In the final task for this laboration we are to do the same request but instead using a telnet client simulation through putty.

**Solution:**



Here we can see that the telnet client is receiving the content as text. The status code, header and body is displayed in order.





Both request are displayed with their entity bodies we can also see that the connection is closed after each request. The difference that lies between requesting in the browser and requesting in a telnet session is the output. In the telnet session we can also manipulate the headers, in a browser the headers are created for us.

In the image below we can see the result of requesting to view an image through the putty terminal. Since the client cannot show images and is restricted to text display, the image appears as a bunch of nonsensically arranged characters. The raw binary data that makes up the image file is UTF-8 encoded for display in the putty terminal, because that is the default character encoding which is used. A more suitable encoding for the requested image would be Base64 encoding which is able to encode arbitrary binary values. UTF-8 is used for Unicode strings instead.



An issue which we ran into while trying to request images inside putty was that the client would throw a network and cease operations if the image file requested was too large. Smaller images would consistently be requested correctly, however larger ones would fail. Therefor care needs to be taken when requesting files as ones too large will cause the request to be terminated as too much time will be spent performing the request.