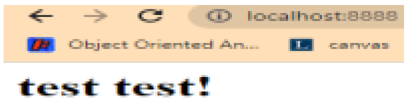
Computer Networks

Assignment 2

Melat Haile

Nahomie Haile

The initial step involved setting up a server that can receive connections and provide a predetermined, valid HTTP/HTML response. This was done to observe the process of connecting to the server using a web browser. The figure below displays the predefined HTML content that is visible when accessing http://localhost:8888/ in the web browser.


localhost:8888

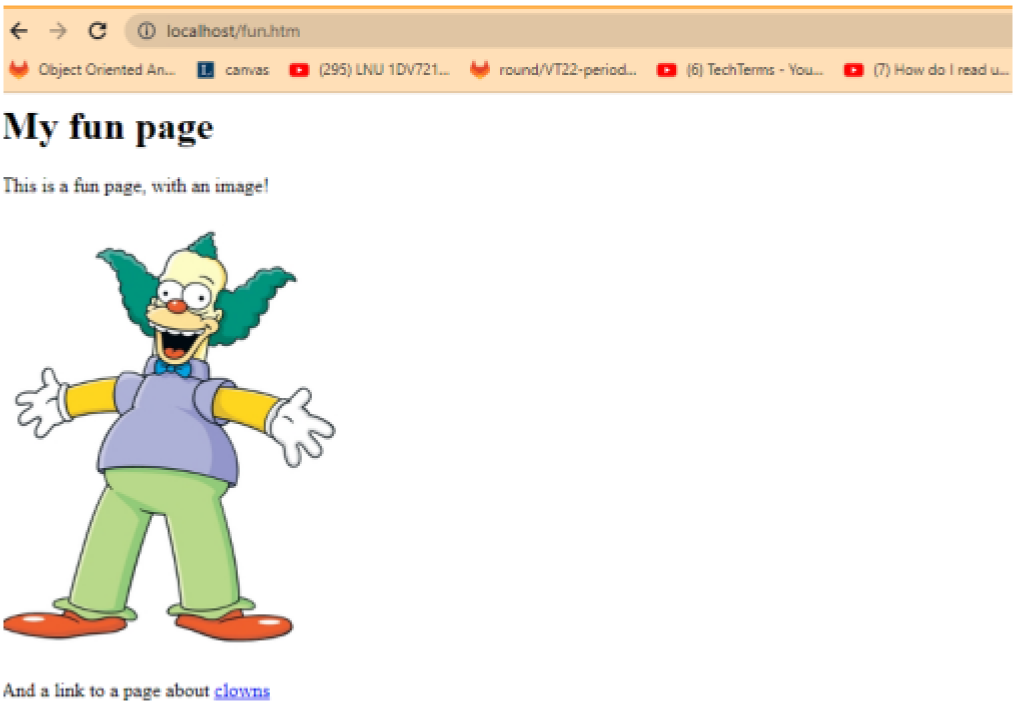
Object Oriented An... canvas

test test!

In order to enhance the code's functionality to read HTML and Image files from disk, certain modifications were made. Firstly, the code was updated to accept two arguments: the port number and the relative path to the "/public" directory. Validation checks were implemented to ensure that the provided directory path is indeed relative. Additionally, a function was created to verify if the correct number of arguments has been passed.

Once the correct arguments, including the valid path and port number, are obtained, a ServerSocket is created on the specified port, allowing it to listen for incoming client connections. Upon a new client connection, a new thread is assigned to handle it. The client socket and public directory path are then passed to a newly created WebServerHandler class, which is responsible for managing the client's request.

Within the WebServerHandler class, a method is implemented to determine the file type and return it as a string. If the file path ends with ".html" or ".htm", the method returns "text/html". Similarly, if the file path ends with ".png", the method returns "image/png". This ensures appropriate handling of different file types during the serving process.

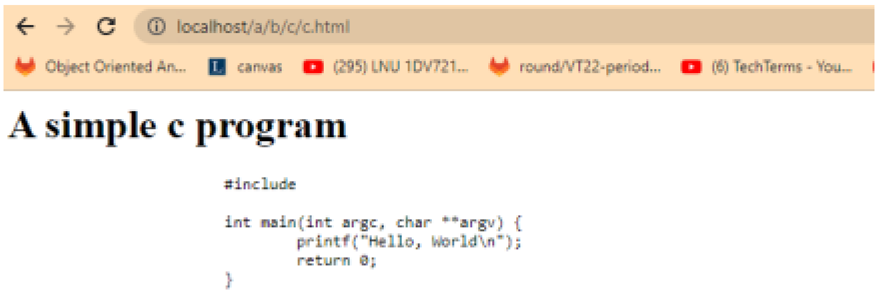

localhost/fun.htm

Object Oriented An... canvas (295) LNU 1DV721... round/VT22-period... (6) TechTerms - You... (7) How do I read u...

My fun page

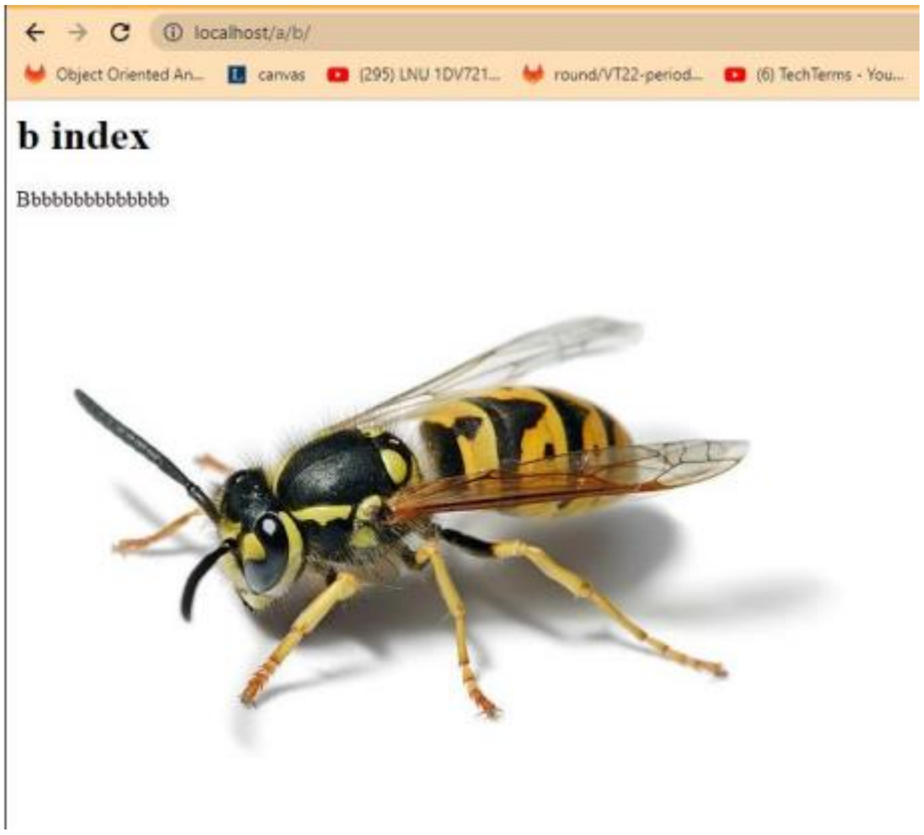
This is a fun page, with an image!

And a link to a page about clowns



A simple c program

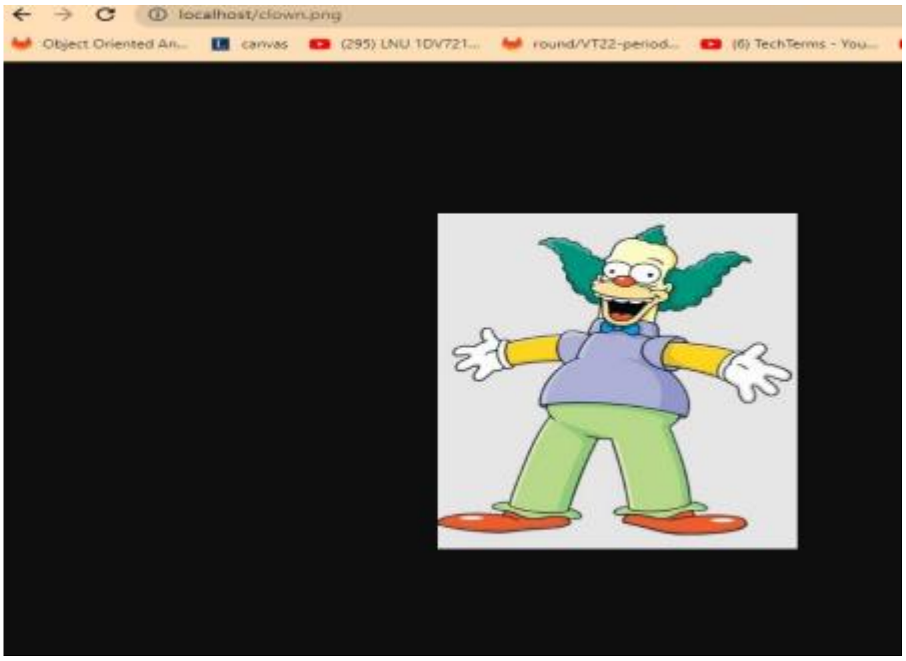
```
#include

int main(int argc, char **argv) {
        printf("Hello, World\n");
        return 0;
}
```



b index

Bbbbbbbbbbbbbb

As previously stated, a method was created to determine the file type and return a string representation. When the file path ends with ".png", the method identifies it as an image file and returns the MIME type "image/png" specifically for PNG images. The response for serving a .png request can be observed in the figure below, demonstrating the successful retrieval of the requested image.

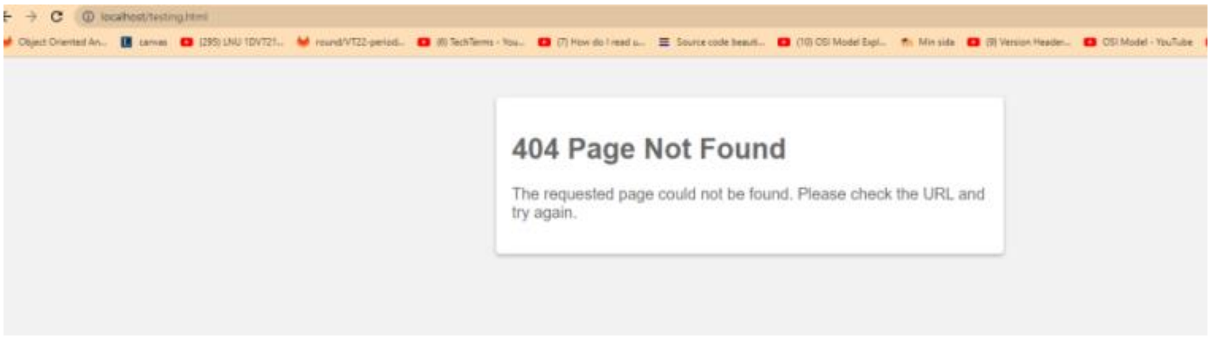


1.3

To ensure protection against directory traversal attacks, the implemented code includes a safety measure. It utilizes the line **String canonicalPath = file.getCanonicalPath();** to obtain the canonical path of the requested file. This ensures that any symbolic links are resolved, relative path elements like ".." are eliminated, and an absolute path is obtained. Following this, the code verifies if the canonical path of the file begins with the expected base directory using the **startsWith()** method. If the canonical path does not have the base directory as its prefix, it indicates that the requested file is located outside the root directory. In such cases, the code invokes the **Handler404()** method to provide a "Not Found" response to the client.

By employing the canonical path, this code effectively defends against attempts to navigate outside the designated directory and prevents unauthorized access.

some file that are in “test” directory was tried to access to see what is the response will be. In the figure below shows the result that was 404 Page not found.



404 Page Not Found

The requested page could not be found. Please check the URL and try again.

1.4

Several exceptions are effectively managed within the code:

1. NumberFormatException: This exception is utilized to prevent the program from crashing in case an invalid port number is provided.

2. IOException: This exception is employed to safeguard against program crashes during server startup or while waiting for client connections. It is utilized in the listening method.

3. IOException: Additionally, this exception is used in the run() method to handle a 500 internal server error.

4. IllegalArgumentException: To ensure that the program does not attempt to access an invalid path or create a File object for an invalid directory, this exception is employed.

5. SecurityException: The publicDir() method utilizes this exception to catch any security-related issues and prints an error message if the program lacks permission to access the public directory.

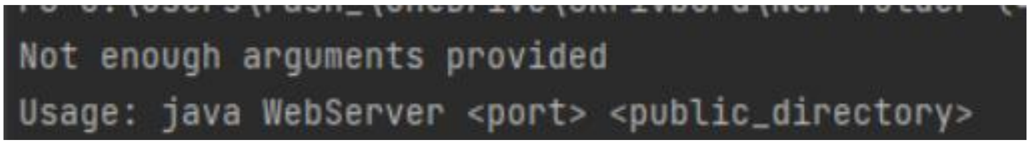
6. SecurityException: This exception is once again used when examining the canonical path of a requested file.

7. SocketException: Whenever there is a problem with the network connection or if the client unexpectedly closes the connection, this exception is employed.

By handling these exceptions appropriately, the code can gracefully handle various error scenarios and maintain the stability and security of the program.

1.5

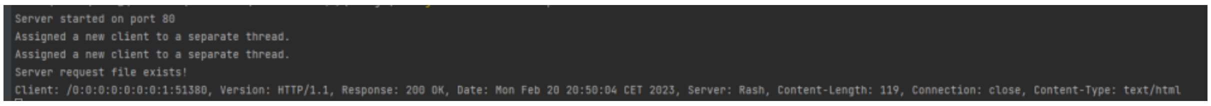
The program expects two arguments: <port> and <public_directory>. It verifies whether the program is initiated with the correct arguments, which include the listening port and the path to the public directory. If the arguments are not provided in the correct order, the code displays an error message. Furthermore, if the program is executed without any arguments, it presents a basic "how to use" guide and informs users about the required arguments to be used, as depicted in the figure below.



```
Not enough arguments provided
Usage: java WebServer <port> <public_directory>
```

1.6

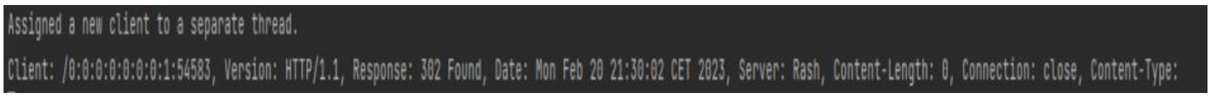
The code incorporates console output that provides information about the server. Initially, it displays a message confirming the successful startup of the web server, including the port number and the designated public directory. Following that, it showcases the server fulfilling a file request. The subsequent output includes details such as the client's IP address, HTTP version, response status code, server name, Content-Length, and Content-Type. The figure below illustrates this output.



```
Server started on port 80
Assigned a new client to a separate thread.
Assigned a new client to a separate thread.
Server request file exists!
Client: /0:0:0:0:0:0:0:1:51380, Version: HTTP/1.1, Response: 200 OK, Date: Mon Feb 20 20:50:04 CET 2023, Server: Rash, Content-Length: 119, Connection: close, Content-Type: text/html
```

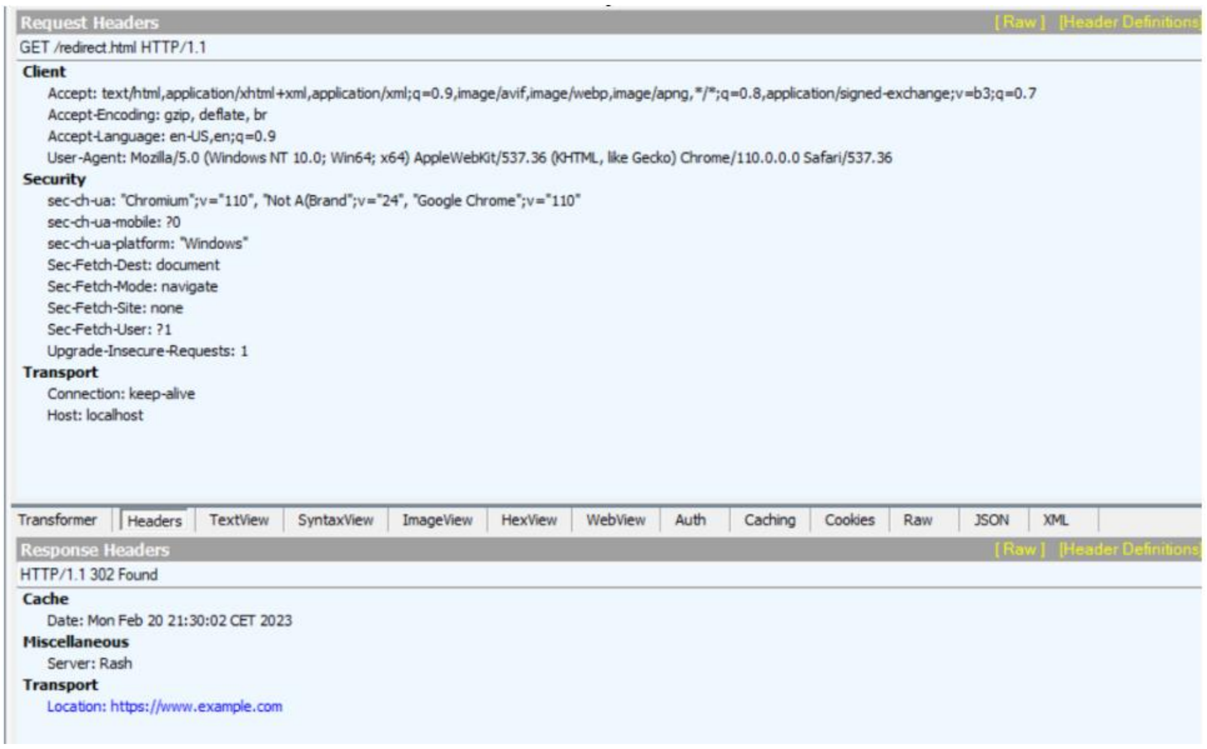
Task 2

In order to generate a 302 response, the client needs to send a request for a resource that has been temporarily relocated to a different location. In our scenario, to test the code, the client should request the "/redirect.html" resource. The server will recognize the request for this specific resource and reply with a 302 status code. We have hardcoded the redirection to occur when the Handler 302 method is invoked for the www.example.com domain. The figure below displays the output in the console, confirming that the response received was a 302 Found status.



```
Assigned a new client to a separate thread.
Client: /0:0:0:0:0:0:0:1:54583, Version: HTTP/1.1, Response: 302 Found, Date: Mon Feb 20 21:30:02 CET 2023, Server: Rash, Content-Length: 0, Connection: close, Content-Type:
```

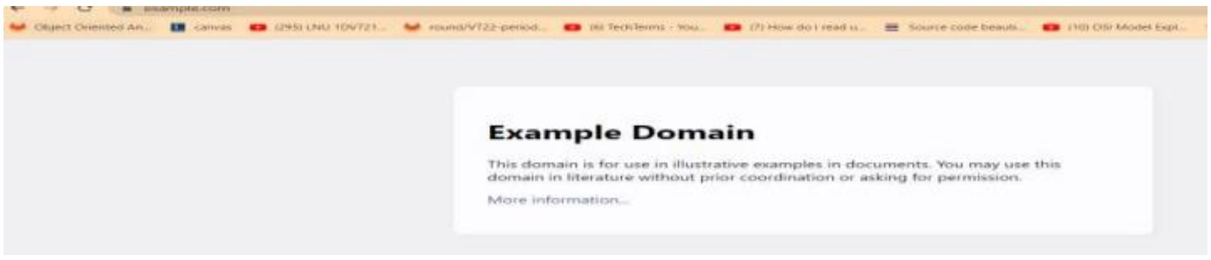
The outcome presented in the figure below illustrates the output obtained from another application called Fiddler, which is capable of capturing and analyzing HTTP and HTTPS traffic. The captured data reveals that the response status received was 302 Found.



```
GET /redirect.html HTTP/1.1
Client
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
    Accept-Encoding: gzip, deflate, br
    Accept-Language: en-US,en;q=0.9
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36
Security
    sec-ch-ua: "Chromium";v="110", "Not A(Brand";v="24", "Google Chrome";v="110"
    sec-ch-ua-mobile: ?0
    sec-ch-ua-platform: "Windows"
    Sec-Fetch-Dest: document
    Sec-Fetch-Mode: navigate
    Sec-Fetch-Site: none
    Sec-Fetch-User: ?1
    Upgrade-Insecure-Requests: 1
Transport
    Connection: keep-alive
    Host: localhost
```

```
HTTP/1.1 302 Found
Cache
    Date: Mon Feb 20 21:30:02 CET 2023
Miscellaneous
    Server: Rash
Transport
    Location: https://www.example.com
```

In the figure displayed below, we can observe the website www.example.com, which has been intentionally set up as a redirect site for testing purposes.



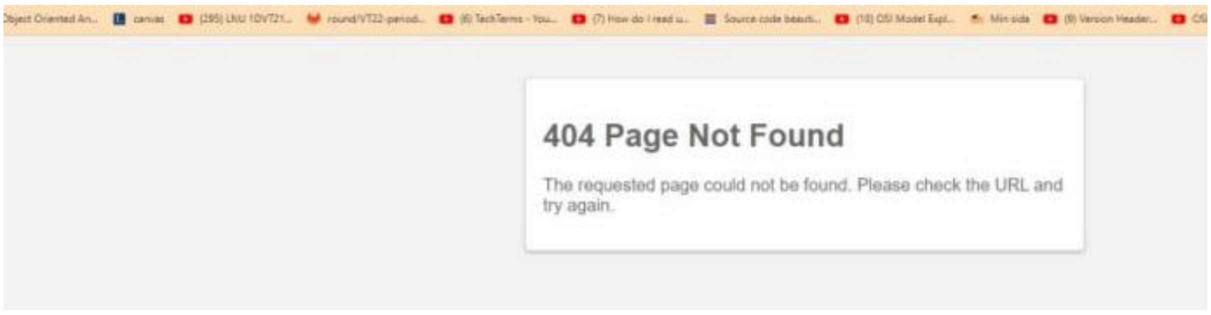
Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...

2.2

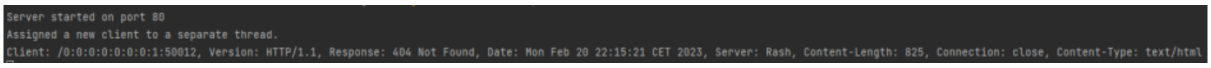
The figure depicted below, illustrates the scenario where the client sends a request to the server for a particular resource that the server is unable to locate or access. As a result, the server responds with a 404 "Not Found" status code.



404 Page Not Found

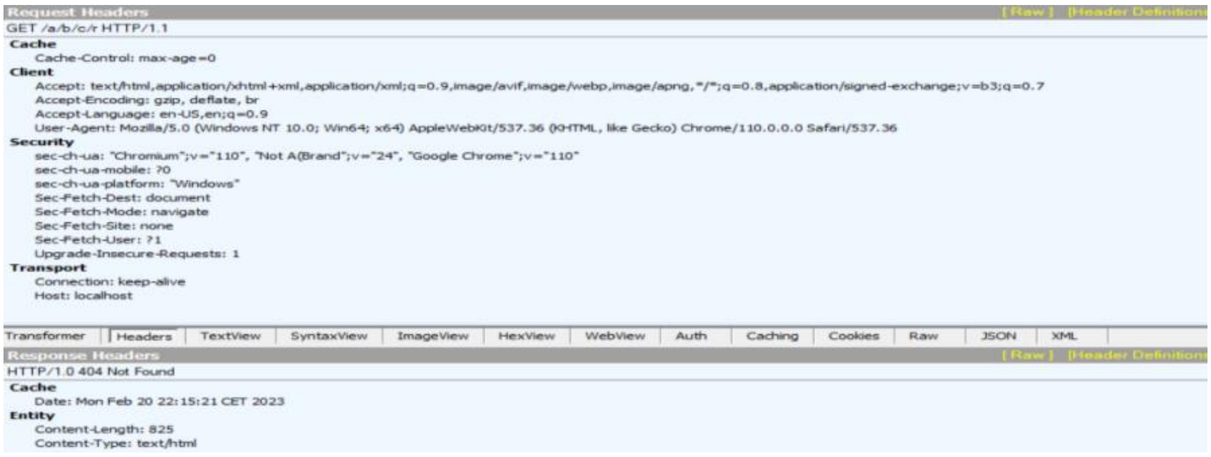
The requested page could not be found. Please check the URL and try again.

Displayed in the figure below is the output console along with the corresponding response, which in this case is a 404 "Not Found" error.



```
Server started on port 80
Assigned a new client to a separate thread.
Client: /0:0:0:0:0:0:0:1:50012, Version: HTTP/1.1, Response: 404 Not Found, Date: Mon Feb 20 22:15:21 CET 2023, Server: Rash, Content-Length: 825, Connection: close, Content-Type: text/html
```

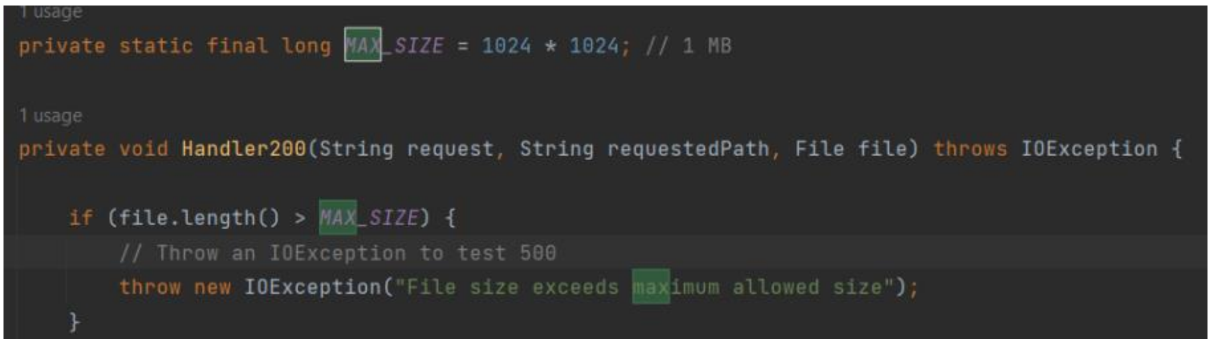
As illustrated in the figure below, the Fiddler application displays the result, indicating a response status of 404 "Not Found".



```
Request Headers                                                        [ Raw ]   [Header Definitions
GET /a/b/c/r HTTP/1.1
Cache
    Cache-Control: max-age=0
Client
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
    Accept-Encoding: gzip, deflate, br
    Accept-Language: en-US,en;q=0.9
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36
Security
    sec-ch-ua: "Chromium";v="110", "Not A(Brand";v="24", "Google Chrome";v="110"
    sec-ch-ua-mobile: ?0
    sec-ch-ua-platform: "Windows"
    Sec-Fetch-Dest: document
    Sec-Fetch-Mode: navigate
    Sec-Fetch-Site: none
    Sec-Fetch-User: ?1
    Upgrade-Insecure-Requests: 1
Transport
    Connection: keep-alive
    Host: localhost

Transformer | Headers | TextView | SyntaxView | ImageView | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML
Response Headers                                                       [ Raw ]   [Header Definitions
HTTP/1.0 404 Not Found
Cache
    Date: Mon Feb 20 22:15:21 CET 2023
Entity
    Content-Length: 825
    Content-Type: text/html
```

2.3

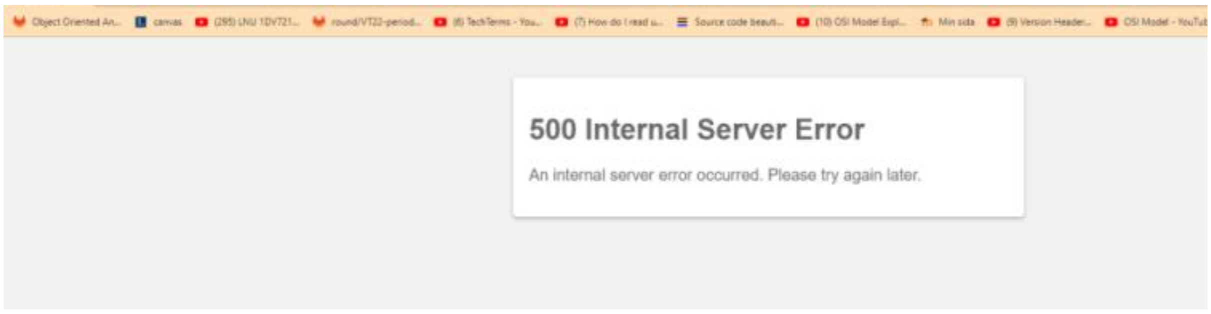
To simulate a 500 internal request, intentional IOExceptions were thrown during the implementation. Specifically, the handler200() method was modified to throw an IOException in specific scenarios. Additionally, the code was adjusted to throw an IOException if the requested file exceeded the maximum allowable size, denoted by MAX_SIZE in the figure below



```
1 usage
private static final long MAX_SIZE = 1024 * 1024; // 1 MB

1 usage
private void Handler200(String request, String requestedPath, File file) throws IOException {

    if (file.length() > MAX_SIZE) {
        // Throw an IOException to test 500
        throw new IOException("File size exceeds maximum allowed size");
    }
```

In the event that the file size surpasses this threshold, the Handler200() method will deliberately throw an IOException. This IOException is then caught within the run() method of the WebServerHandler class, prompting the execution of the Handler500() method to transmit a 500 Internal Server Error response back to the client.

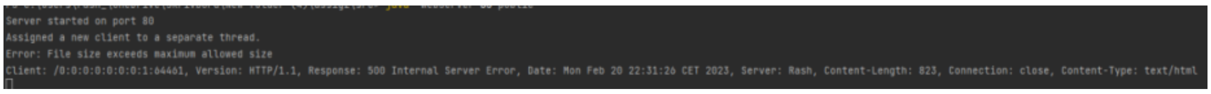
To validate this functionality, a request for the "world.png" file, which exceeds the file size limit defined in the modified code, was made to the server. As depicted in the figure below, the server appropriately responds with a 500 Internal Server Error.



500 Internal Server Error

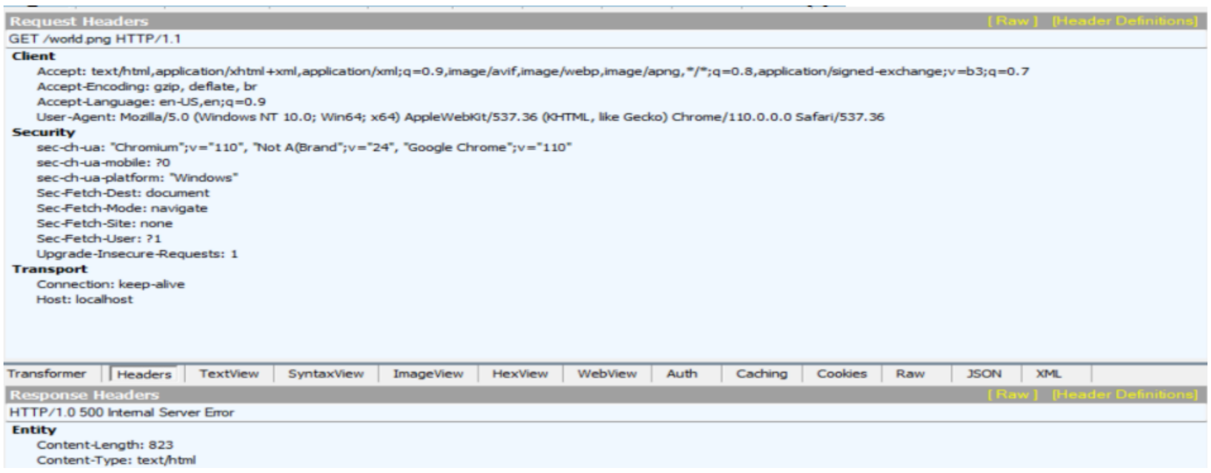
An internal server error occurred. Please try again later.

The output console, as illustrated in the figure below, displays the result of the server's response, which indicates a 500 Internal Server Error.



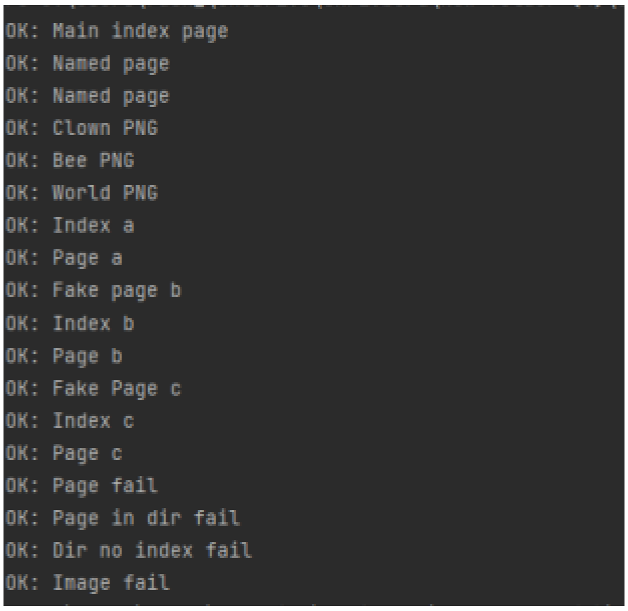
```
Server started on port 80
Assigned a new client to a separate thread.
Error: File size exceeds maximum allowed size
Client: /0:0:0:0:0:0:0:1:64461, Version: HTTP/1.1, Response: 500 Internal Server Error, Date: Mon Feb 20 22:31:26 CET 2023, Server: Rash, Content-Length: 823, Connection: close, Content-Type: text/html
```

The result displayed in the Fiddler app, as depicted in the figure below, is consistent with the previous observation of a 500 Internal Server Error.



```
Request Headers                                                    [ Raw ]  [Header Definitions]
GET /world.png HTTP/1.1
Client
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
    Accept-Encoding: gzip, deflate, br
    Accept-Language: en-US,en;q=0.9
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36
Security
    sec-ch-ua: "Chromium";v="110", "Not A(Brand";v="24", "Google Chrome";v="110"
    sec-ch-ua-mobile: ?0
    sec-ch-ua-platform: "Windows"
    Sec-Fetch-Dest: document
    Sec-Fetch-Mode: navigate
    Sec-Fetch-Site: none
    Sec-Fetch-User: ?1
    Upgrade-Insecure-Requests: 1
Transport
    Connection: keep-alive
    Host: localhost

Response Headers                                                   [ Raw ]  [Header Definitions]
HTTP/1.0 500 Internal Server Error
Entity
    Content-Length: 823
    Content-Type: text/html
```

Testing was conducted using the testa2u1.py script, and the outcome is presented in the figure below.



```
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
```

Summary

Melat Haile: 50%,

Nahomie Haile: 50%.

Instructions: How to Run

To run the application, follow the steps below:

1. Open the terminal.

2. Change the current directory to "src" using the command: `cd src`.

3. Compile all Java source files using the command: `javac *java`.

4. Start the web server by running the following command: `java WebServerHandler <port> <public_directory>`, where `<port>` is the desired port number and `<public_directory>` is the path to the directory containing the public files.

Instructions: Running testa2u1.py

To execute the testa2u1.py script, please perform the following steps:

1. Ensure that the server is already running by following the instructions provided in the "How to Run" section.

2. Open another terminal.

3. Change the current directory to "src" using the command: `cd src`.

4. Execute the testa2u1.py script by running the command: `python .\testa2u1.py`.

Triggering Error Conditions

To trigger specific error conditions, use the following guidelines:

- To trigger a 500 error, uncomment lines 155, 169 and 170 in the WebServerHandler class.

- To trigger a 302 error, request the resource "/redirect.html".

- To trigger a 404 error, request a resource that the server cannot find or access.