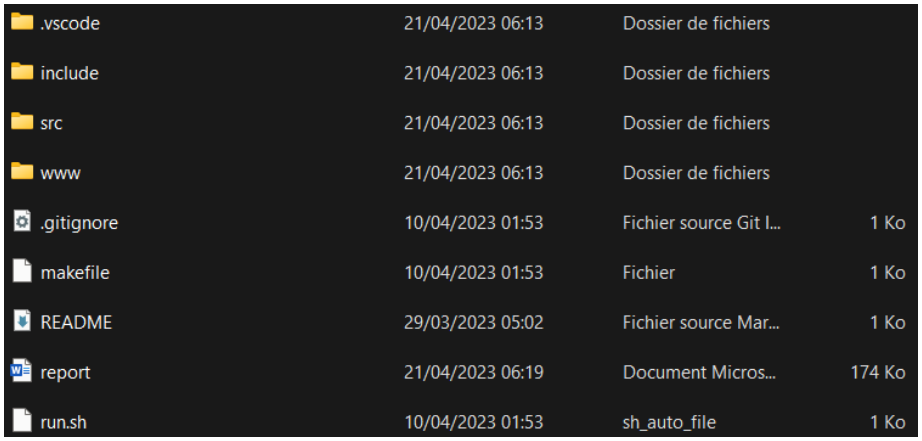
CITRINI Mathias

Project 1: Web server

I. Architecture of the project



Makefile	makefile
README and .gitignore	Useful for GitHub dev
include/	Contains Header
src/	Contains source files (.c)
www/	Contains website and all the assets (css, images)

II. Commands (in root)

- To compile the project, or clean it:

  Command: `make`
  Command: `make clean`

- To start the server, use:

  Command: `./server <port>`

  Replace <port> with your own port (ex: 8077)

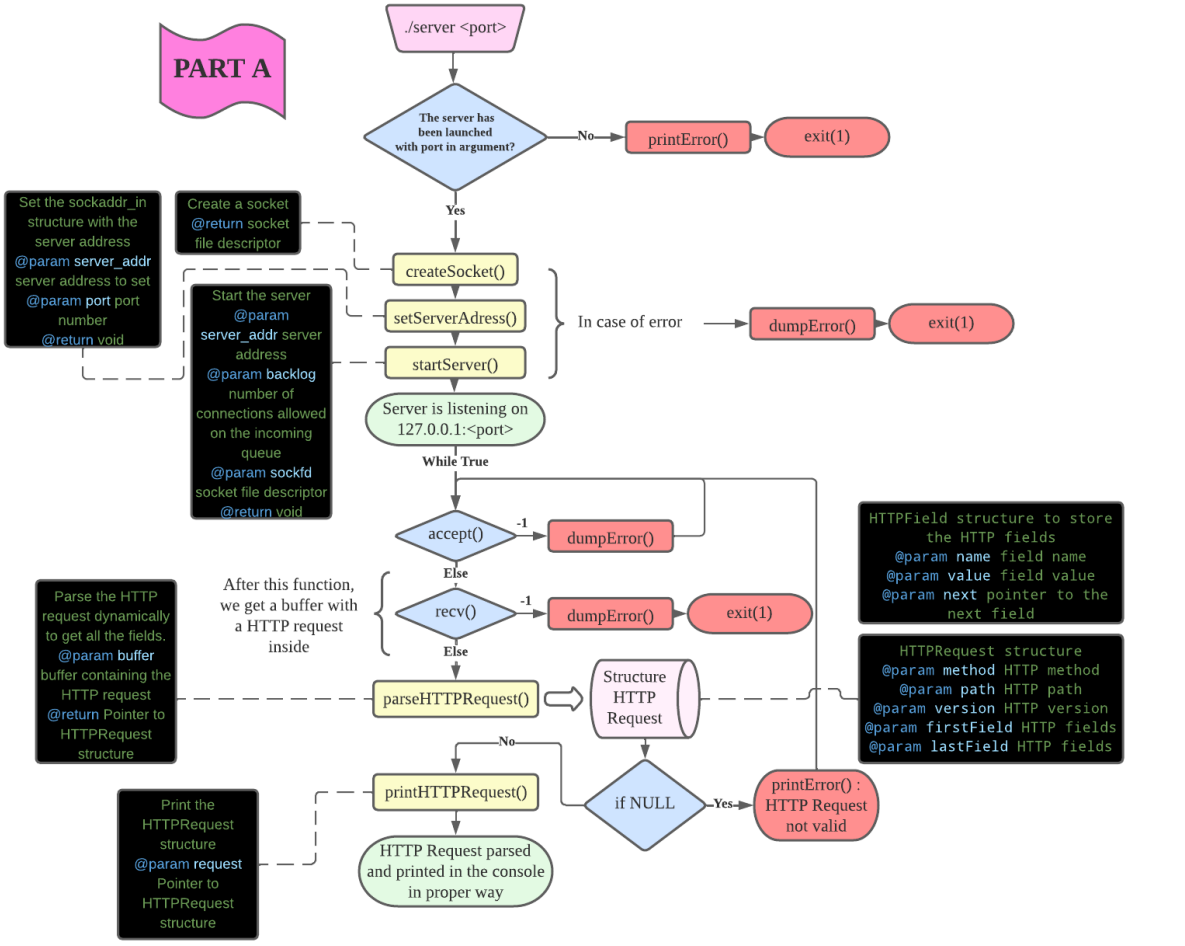
III. Webserver hub

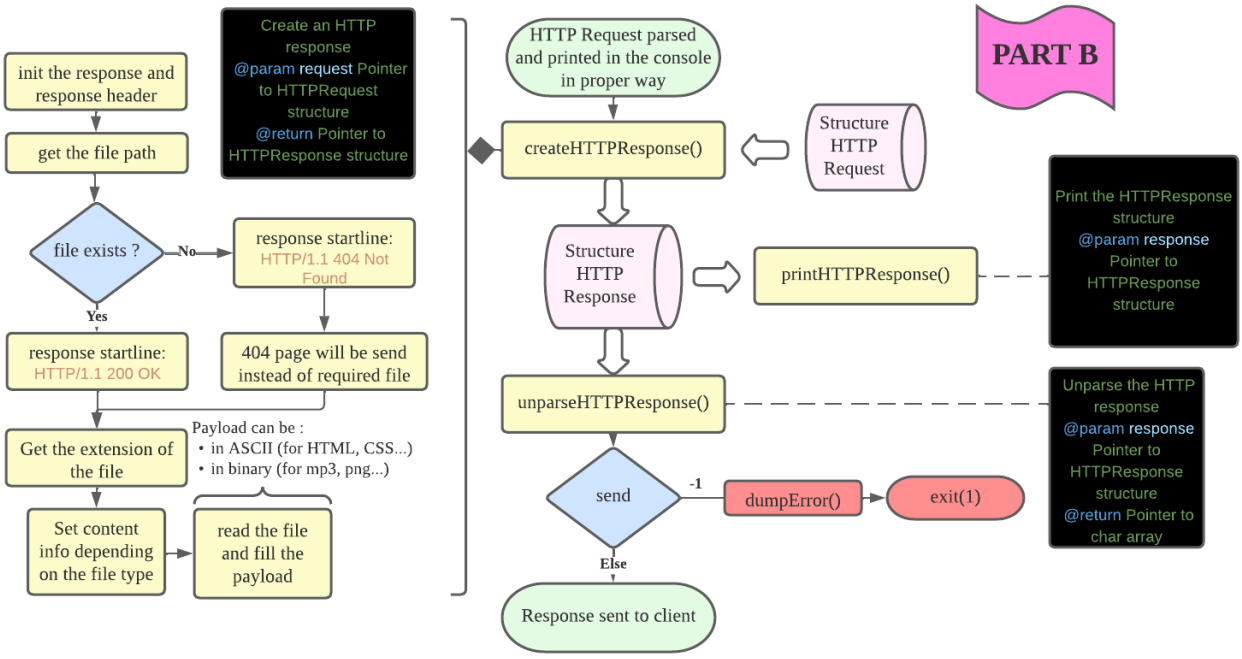
I created a dummy website to test every requirement easily. This website is a basic one and use only one js script and one css file. It can display images, has video integration…

To access this website, please go to: <ip address>:<port>/index.html or <ip address>:<port>/

Use Mozilla Firefox to have better performance.

IV. How the server works?





V. Server's outputs analysis

❖ Server status messages

```
Socket created successfully
Server's address: 127.0.0.1:8000
Binding socket...
Listening...
Server has been started successfully
```

```
Server got connection from 127.0.0.1 !
Buffer is not a valid HTTP request ! Skipping...
```

The server can print status messages in the console. Green messages are confirmations of successful operations, red messages are errors and white and grey messages are server status and information.

❖ Request received.

```
Server got connection from 127.0.0.1 !
Method: GET
Path: /mp3/audio.mp3
Version: HTTP/1.1
Host: 127.0.0.1:8000
User agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/112.0
Accept: audio/webm,audio/ogg,audio/wav,audio/*;q=0.9,application/ogg;q=0.7,video/*;q=0.6,*/
Accept language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept encoding: bytes=0-
Connection: keep-alive
Upgrade insecure requests: http://127.0.0.1:8000/
```

In this situation, we can see that the server received an HTTP request from a client (IP:127.0.0.1). The client is running the web browser Mozilla Firefox on Ubuntu and is requesting an mp3 file (audio.mp3). The path of the file is /mp3/audio.mp3. HTTP requests are printed in orange/yellow in the console.

❖ Response sent.

```
File path: www/mp3/audio.mp3
Extension of the accepted file: mp3
Server Response:
HTTP/1.1 200 OK
Content-Type: audio/mp3
Content-Length: 3672117

Binary file data
```

In this situation the server is responding to the previous request received (see above). The server gets the "real" file path form the path sent by client (all the website files are in www folder). The file exists so status code of the message is 200. Then, the server set the content-type to audio/mp3 and fill the payload with the file content. Moreover, Content-length will contain the size of the requested file.
HTTP response are printed in blue in the console, but only text content is printed (not binary).

VI. Difficulties

The first difficulty was when I had to parse the HTTP request. Sometimes, I had core dumped so I had to choose the best size for each field and optimize the memory allocation. So, I used malloc almost every time.

The second difficulty was to understand how to send binary files (Images, Audio …). In fact, I took some time to understand how binary files work and how to read it because I did not use to. I learned the best way was to use void* instead of char*, and so, memcpy instead of strcat. After long hours of pain, I finally managed to get the proper size of the file, read it with fread and copy the binary data in a string to send it to the client.

To summarize, memory problem and binary data were my biggest difficulties in this project.