HTTP Client/Server

Networks

Name: Khaled Abdelfattah

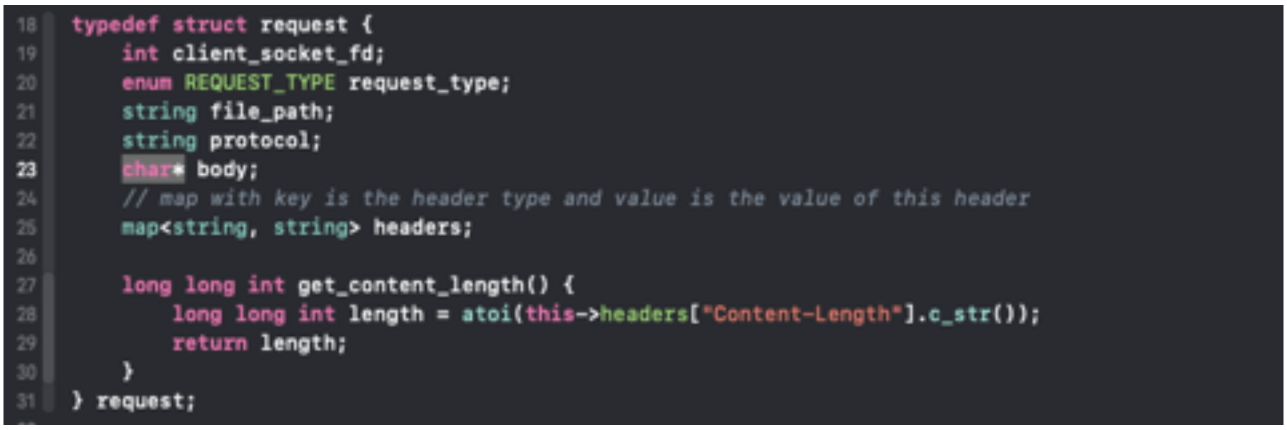
ID: 22

Code Structure

Server Side:

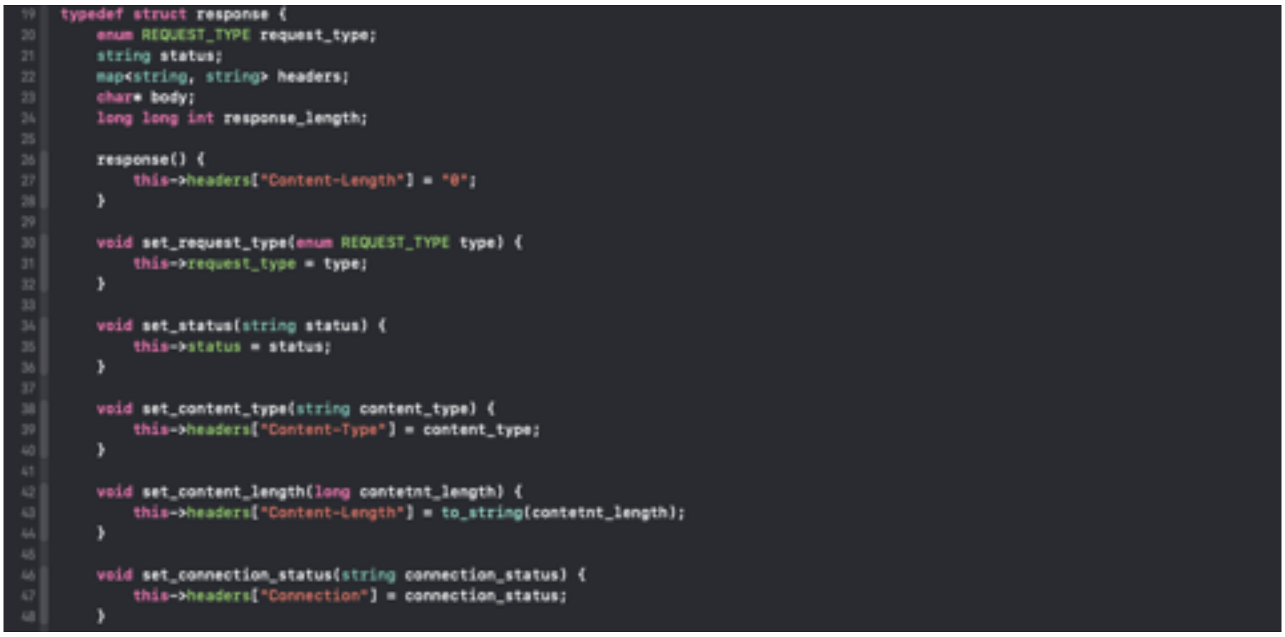
- Server manager:
    - Contains server class which contains server's parameters and initiate server function for initiating the server side.
    - initiate function: create server and bind it to port number and initiate welcoming socket for incoming connections then detach a thread to process each new connection with the server.
- Connection worker:
    - connection worker file contains the main strategy of processing connections
    - establish connection function, after accepting connection from client this function watching the connection's socket read incoming requests then parse them and send the responses back to client
    - While watching the socket worker waits for 60 seconds if there's no requests server close the connection between the client
- Request handler:
    - Receives requests from manager and parse them by passing requests' bodies to request parsers and then build their responses by passing the requests to response builder.
- Request parser:
    - Parse incoming requests from client and return request pointer
- Response builder:
    - After parsing client's requests it passes those requests to response builder header to process the request and build its response and send response pointer back.

- Request structure:



```
typedef struct request {
    int client_socket_fd;
    enum REQUEST_TYPE request_type;
    string file_path;
    string protocol;
    char* body;
    // map with key is the header type and value is the value of this header
    map<string, string> headers;

    long long int get_content_length() {
        long long int length = atoi(this->headers["Content-Length"].c_str());
        return length;
    }
} request;
```

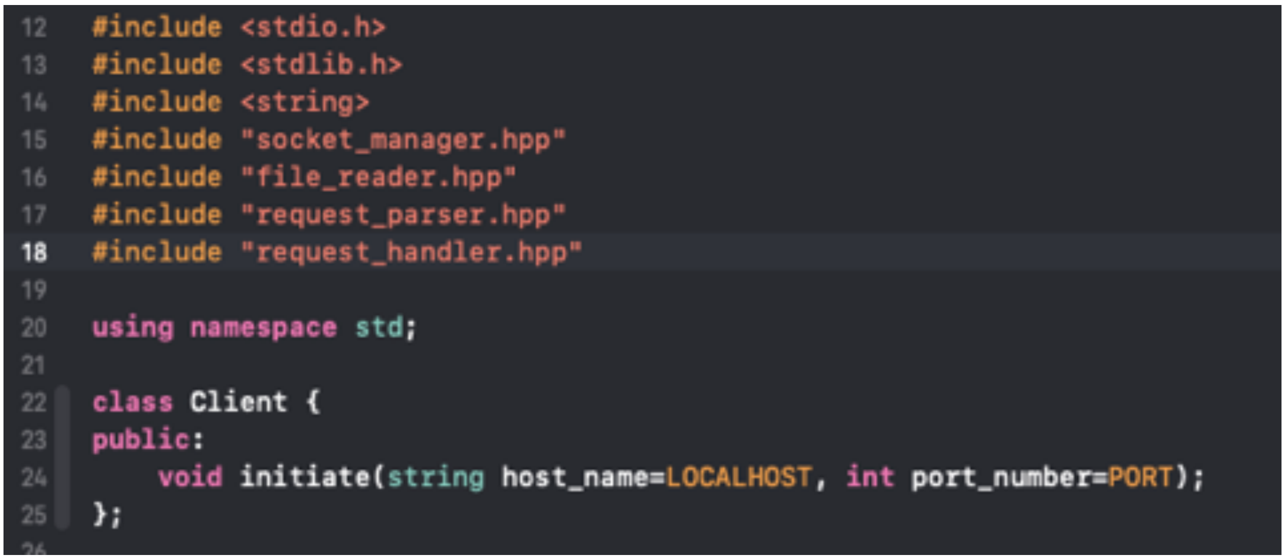
- Response structure:



```
typedef struct response {
    enum REQUEST_TYPE request_type;
    string status;
    map<string, string> headers;
    char* body;
    long long int response_length;

    response() {
        this->headers["Content-Length"] = "0";
    }

    void set_request_type(enum REQUEST_TYPE type) {
        this->request_type = type;
    }

    void set_status(string status) {
        this->status = status;
    }

    void set_content_type(string content_type) {
        this->headers["Content-Type"] = content_type;
    }

    void set_content_length(long contetnt_length) {
        this->headers["Content-Length"] = to_string(contetnt_length);
    }

    void set_connection_status(string connection_status) {
        this->headers["Connection"] = connection_status;
    }
```

Client Side:

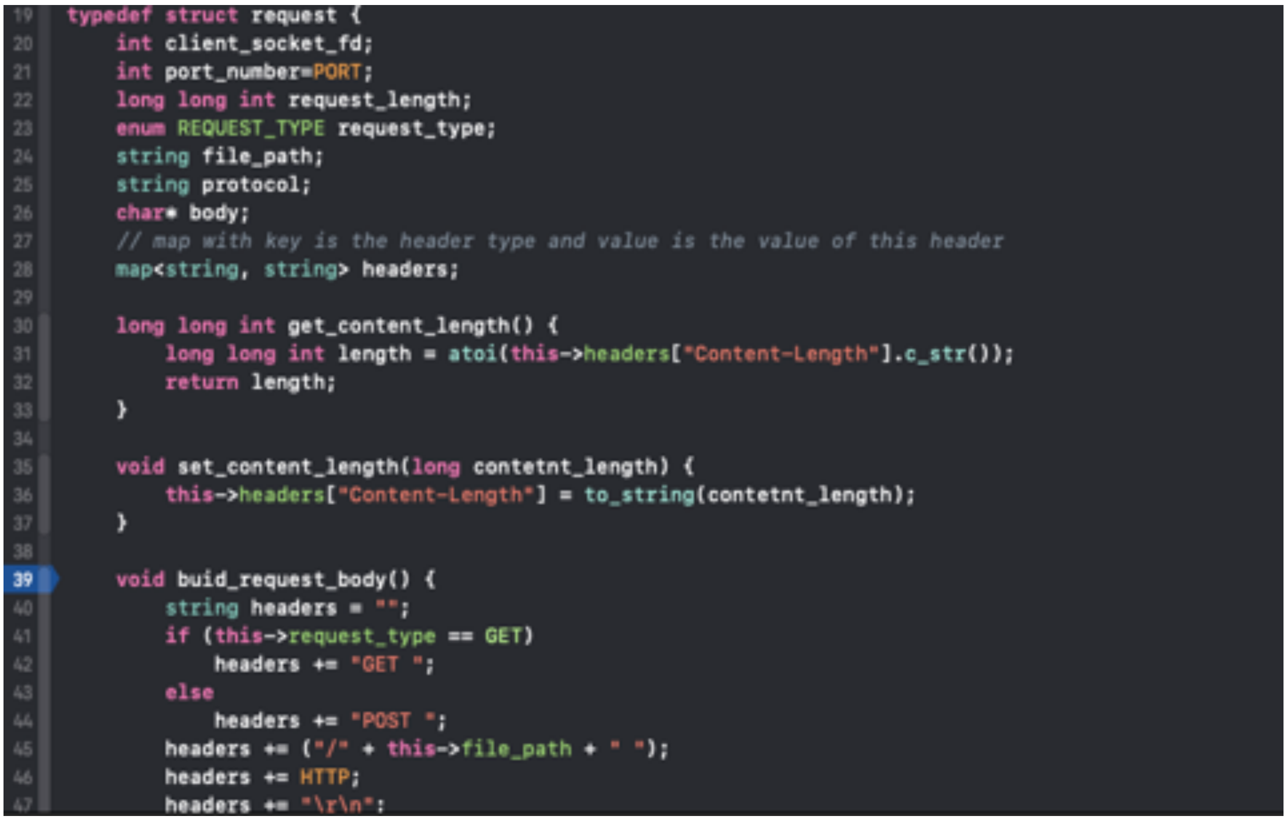
- Client manager:
    - Contains client class and initiate client function for initiating client and bind it with the server



```cpp
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include "socket_manager.hpp"
#include "file_reader.hpp"
#include "request_parser.hpp"
#include "request_handler.hpp"

using namespace std;

class Client {
public:
    void initiate(string host_name=LOCALHOST, int port_number=PORT);
};
```

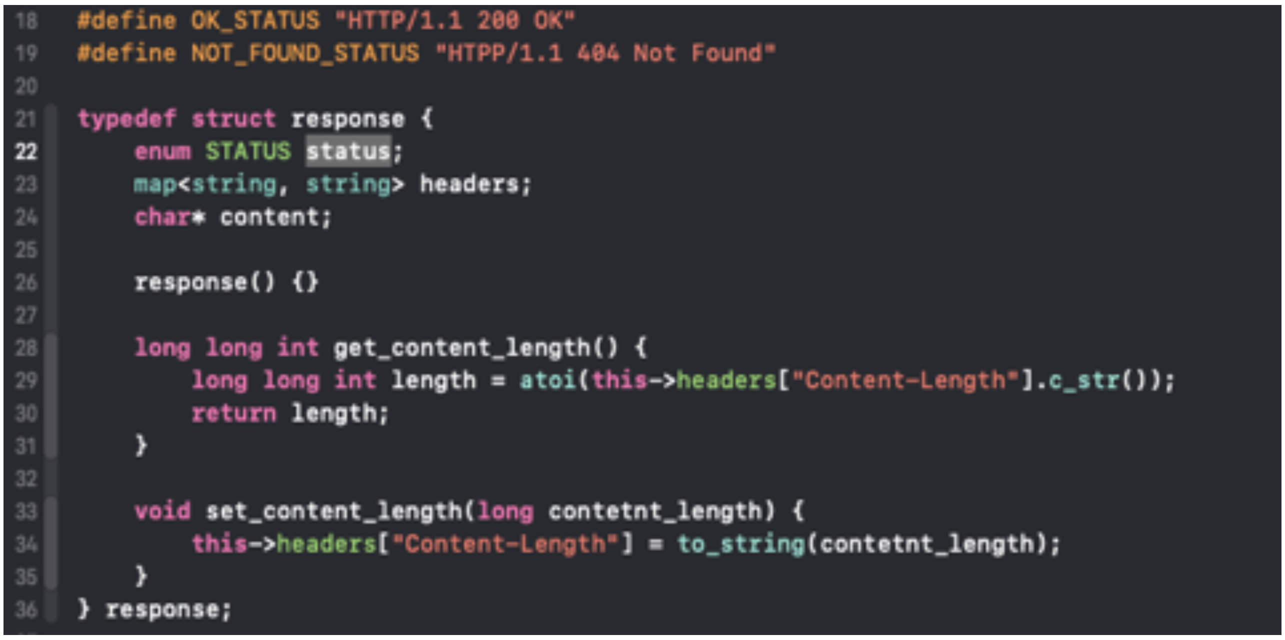
- File reader:
    - Reads client's requests from input file
- Client requests parser:
    - Parsers client request and combine the request for the same server together in client requests map
- Request handler:
    - Handles clients request by building requests body and then all requests to the server and waits for responses from server
- Request parser:
    - Parses client's requests and return request pointers of http requests
- Response manager:
    - After sending requests client reads the responses and manage them by divide responses for each request sent by client and parse them if the request was GET request and the response's status 200 OK client saves the content of the response in the storage directory and so on.
- Socket manager:
    - Socket manager for binding on server socket, send requests through socket, and read response from the socket

- Request:



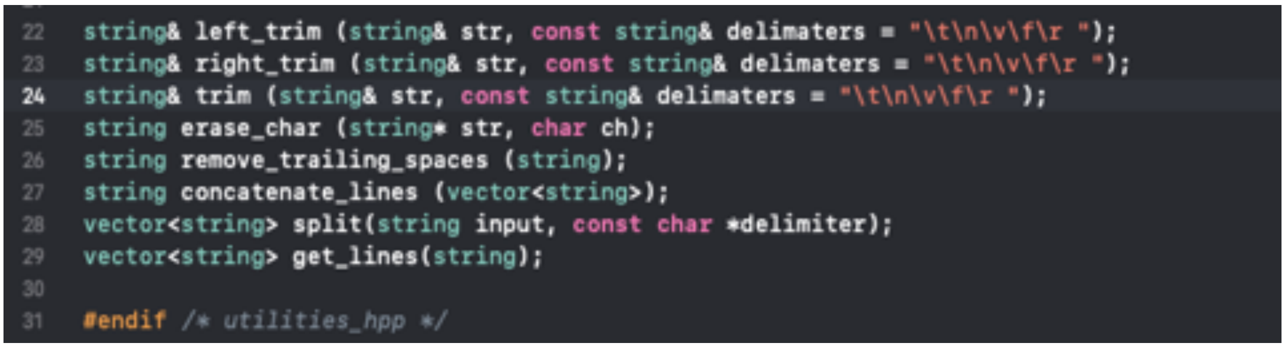
```
typedef struct request {
    int client_socket_fd;
    int port_number=PORT;
    long long int request_length;
    enum REQUEST_TYPE request_type;
    string file_path;
    string protocol;
    char* body;
    // map with key is the header type and value is the value of this header
    map<string, string> headers;

    long long int get_content_length() {
        long long int length = atoi(this->headers["Content-Length"].c_str());
        return length;
    }

    void set_content_length(long contetnt_length) {
        this->headers["Content-Length"] = to_string(contetnt_length);
    }

    void buid_request_body() {
        string headers = "";
        if (this->request_type == GET)
            headers += "GET ";
        else
            headers += "POST ";
        headers += ("/" + this->file_path + " ");
        headers += HTTP;
        headers += "\r\n";
```

- Response:



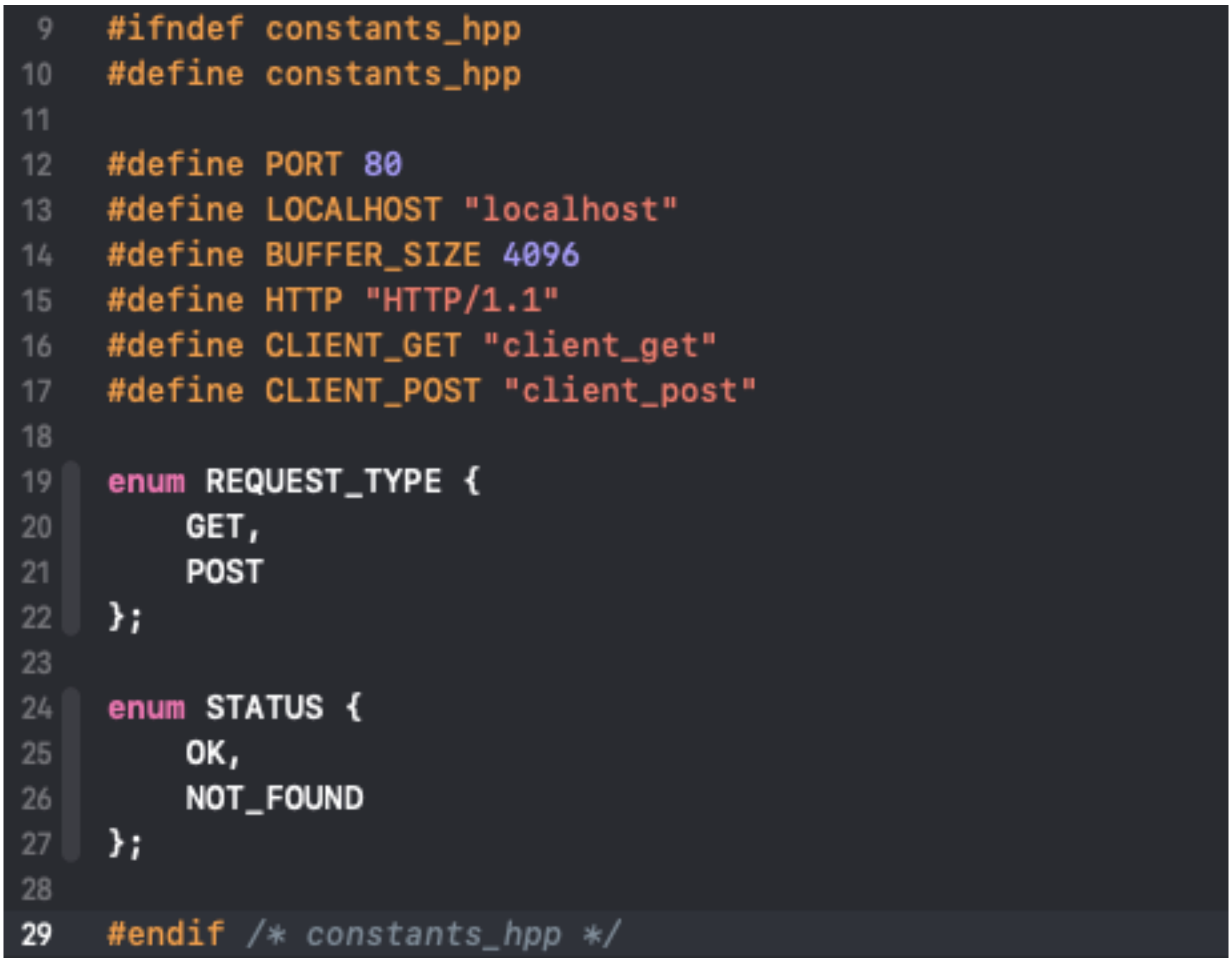
```
#define OK_STATUS "HTTP/1.1 200 OK"
#define NOT_FOUND_STATUS "HTPP/1.1 404 Not Found"

typedef struct response {
    enum STATUS status;
    map<string, string> headers;
    char* content;

    response() {}

    long long int get_content_length() {
        long long int length = atoi(this->headers["Content-Length"].c_str());
        return length;
    }

    void set_content_length(long contetnt_length) {
        this->headers["Content-Length"] = to_string(contetnt_length);
    }
} response;
```

- Helper File
    - Utilities:
        - Contains the main functions used in string processing manner

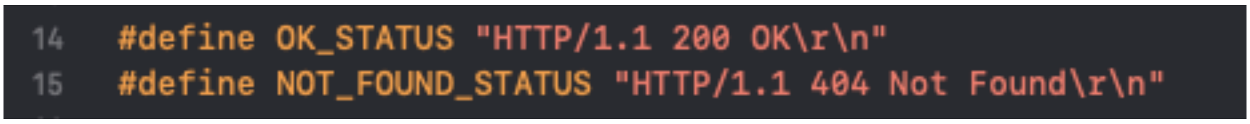


```
string& left_trim (string& str, const string& delimaters = "\t\n\v\f\r ");
string& right_trim (string& str, const string& delimaters = "\t\n\v\f\r ");
string& trim (string& str, const string& delimaters = "\t\n\v\f\r ");
string erase_char (string* str, char ch);
string remove_trailing_spaces (string);
string concatenate_lines (vector<string>);
vector<string> split(string input, const char *delimiter);
vector<string> get_lines(string);

#endif /* utilities_hpp */
```

- Constants files:



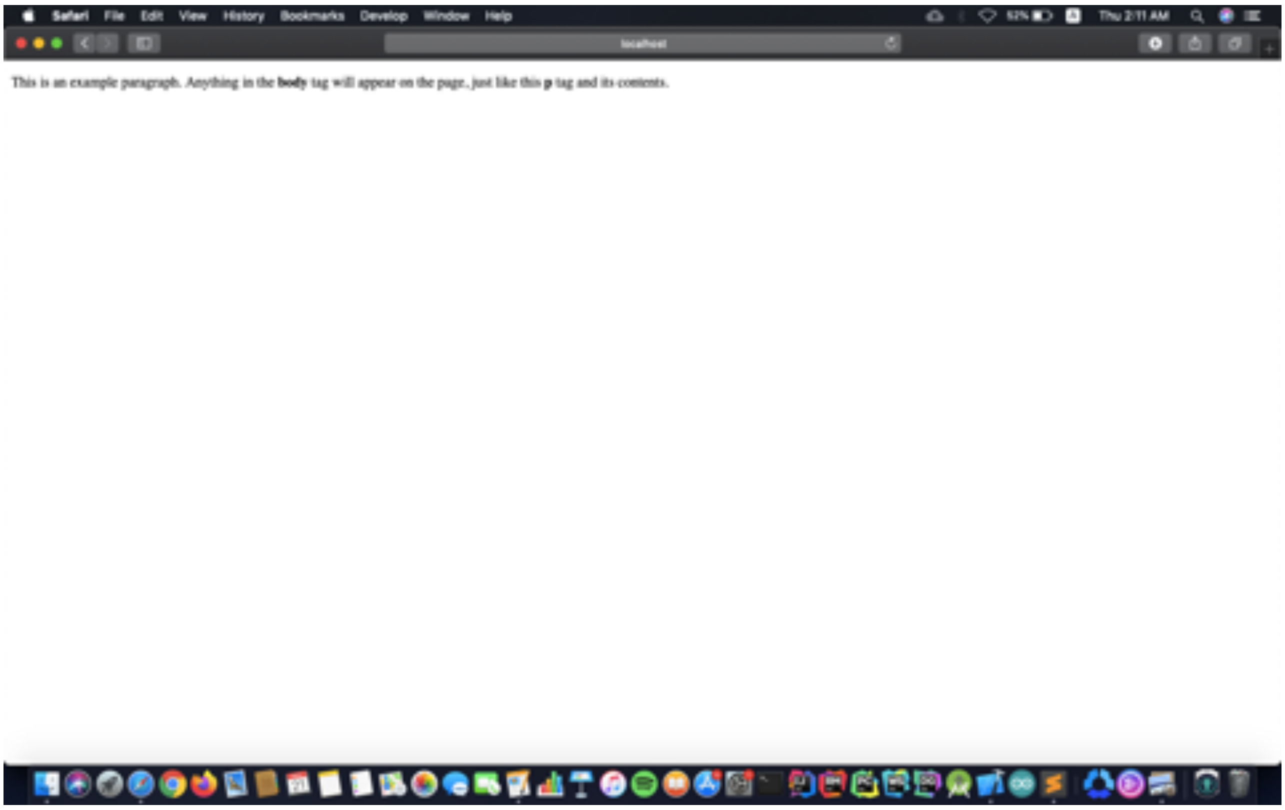
```
#ifndef constants_hpp
#define constants_hpp

#define PORT 80
#define LOCALHOST "localhost"
#define BUFFER_SIZE 4096
#define HTTP "HTTP/1.1"
#define CLIENT_GET "client_get"
#define CLIENT_POST "client_post"

enum REQUEST_TYPE {
    GET,
    POST
};

enum STATUS {
    OK,
    NOT_FOUND
};

#endif /* constants_hpp */
```



```
#define OK_STATUS "HTTP/1.1 200 OK\r\n"
#define NOT_FOUND_STATUS "HTTP/1.1 404 Not Found\r\n"
```

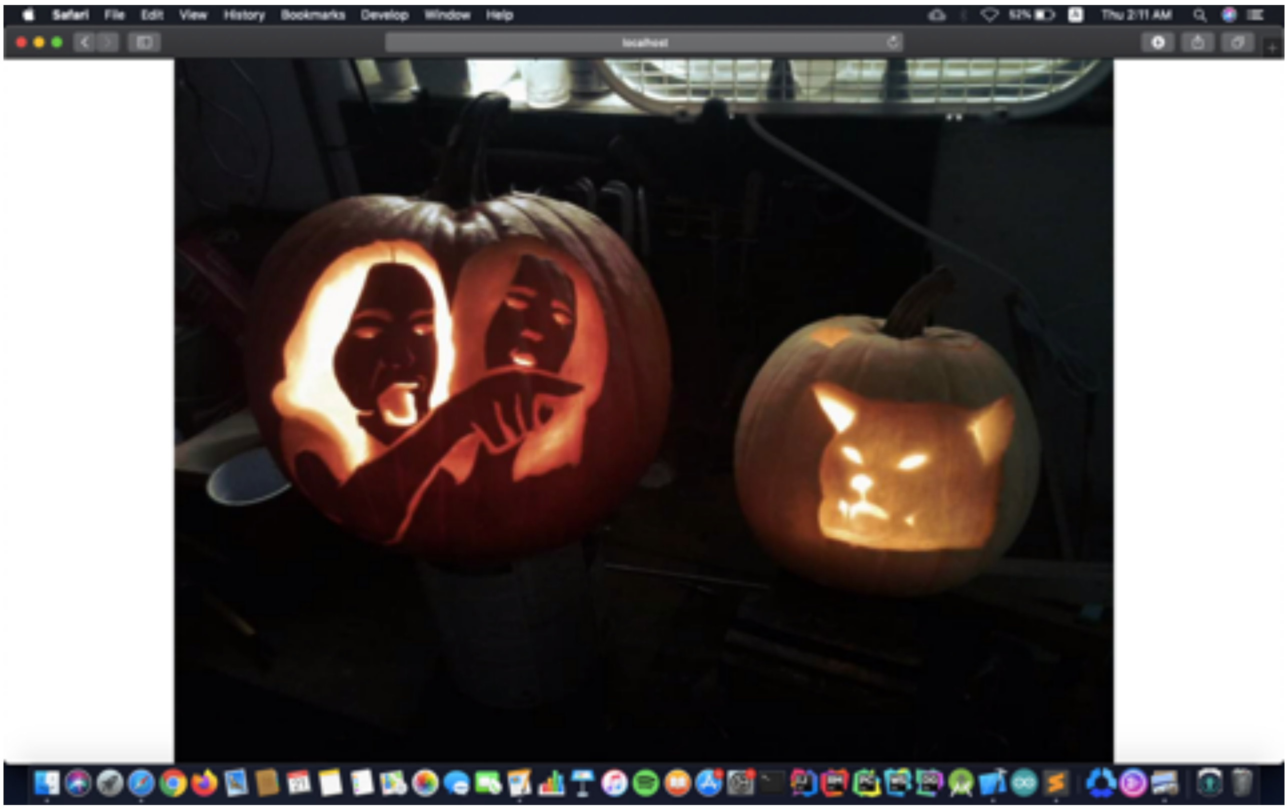
Sample Runs:

- Get simple html file

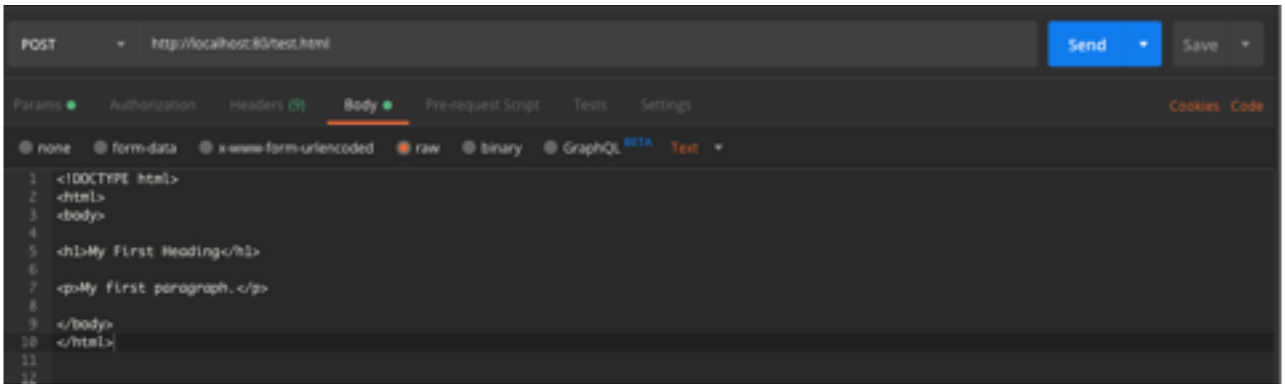


This is an example paragraph. Anything in the **body** tag will appear on the page., just like this **p** tag and its contents.

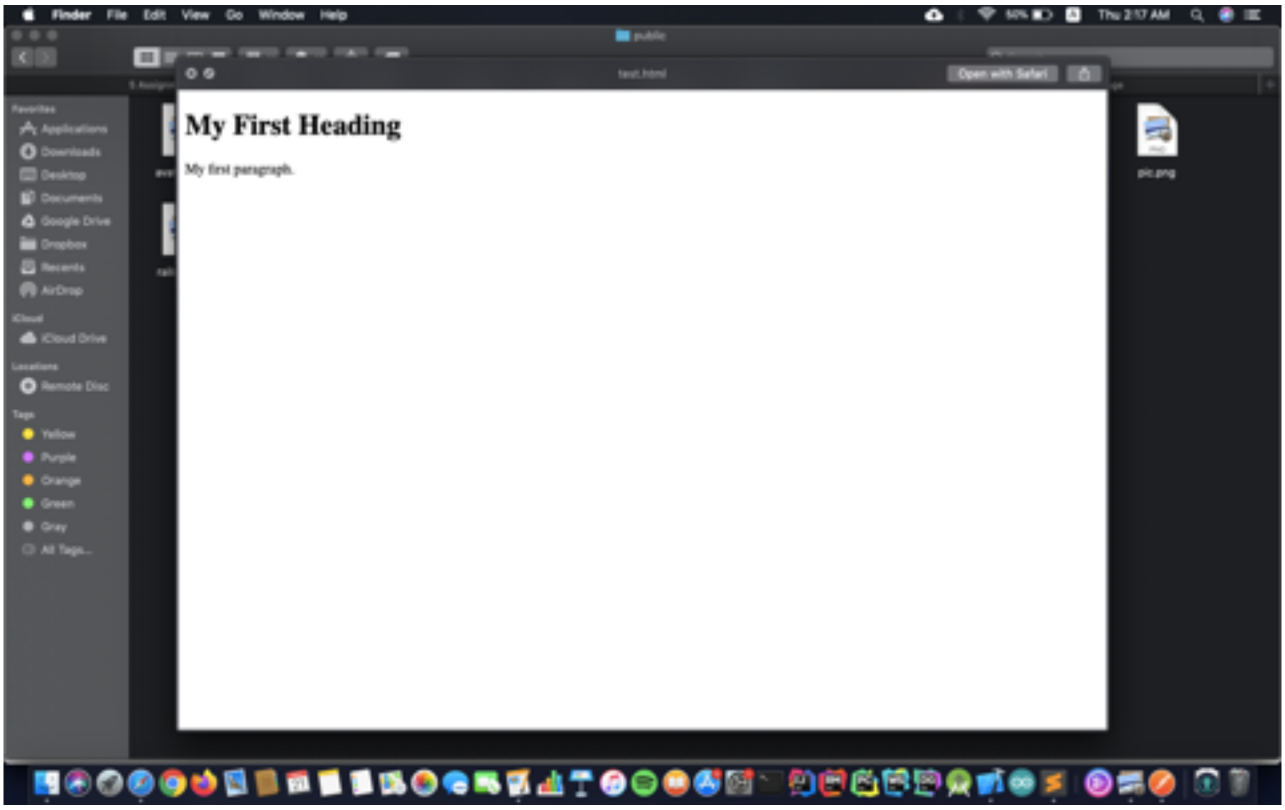
- Get picture



- Post simple html



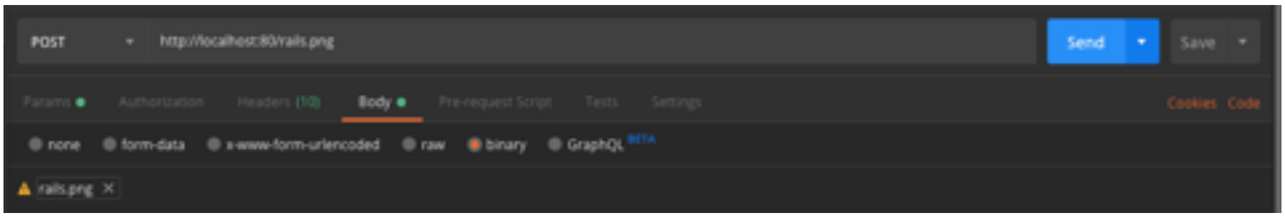
```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

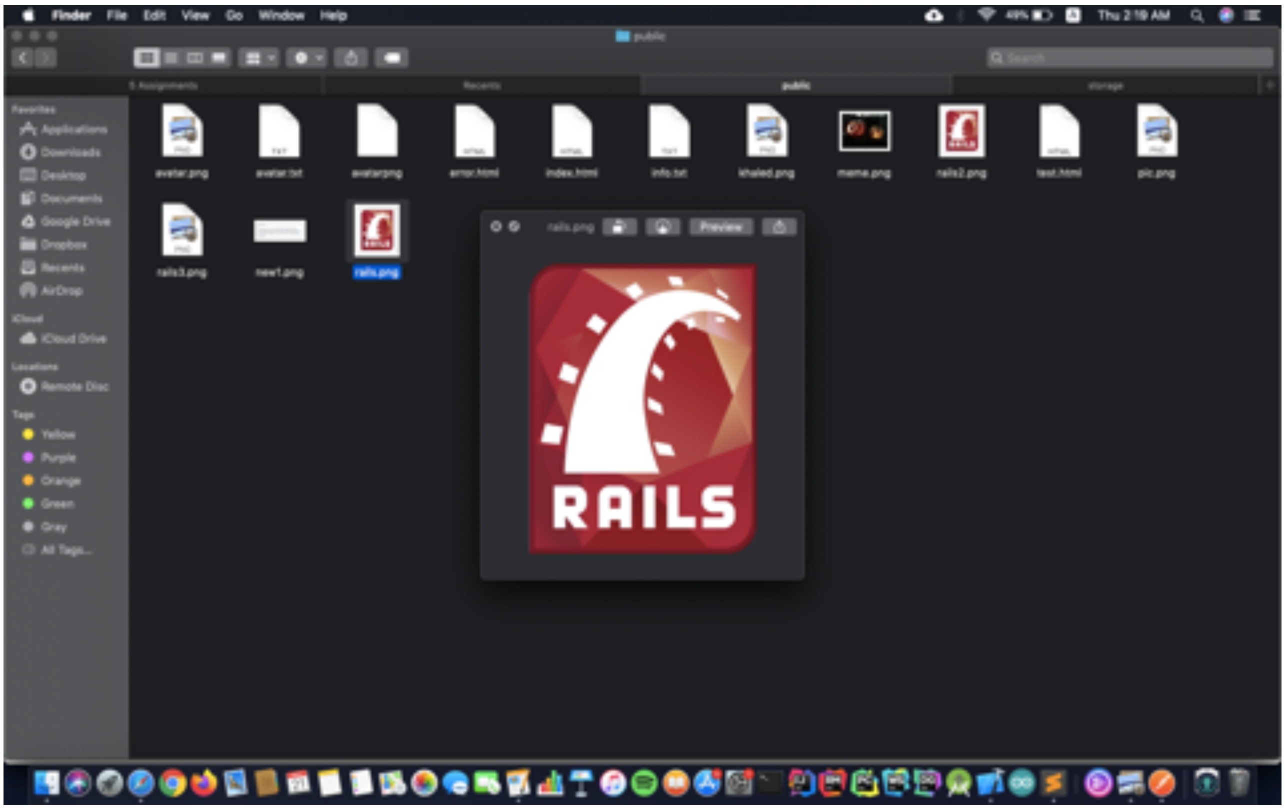


My First Heading

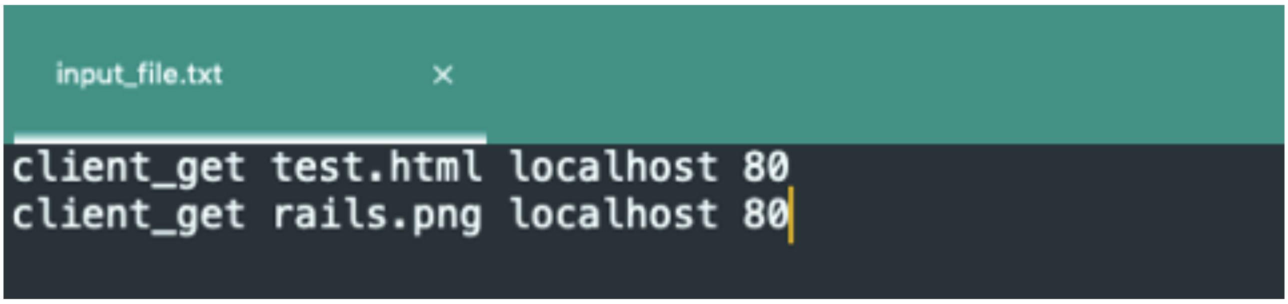
My first paragraph.

- Post picture

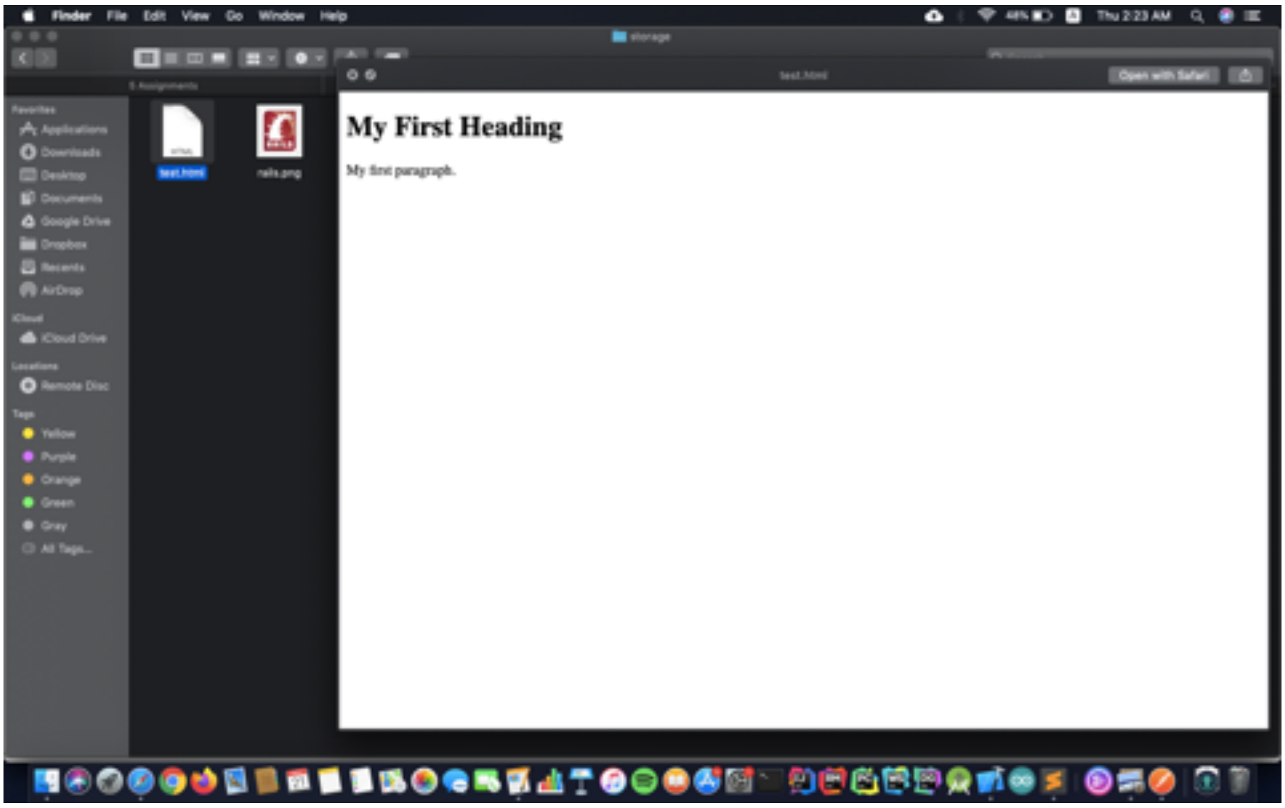




- Get html file and picture from server using client



```
client_get test.html localhost 80
client_get rails.png localhost 80
```



My First Heading

My first paragraph.