32229127 - Wijaya I Putu Janardana

Introduction

Web servers are the most important things to consider since they take vital roles in making web browsers properly work. If there are some problems throughout the process existing on web servers, they might cause inconvenience for us while browsing something. This work therefore becomes meaningful for us, especially for a computer science student like me. By doing this work, I will hopefully be well-informed about how the web servers actually work behind the screen so that I, at least, know what to do next if certain problems occur such as slow loading times, incorrect file path, and so on.

Background

Since I am not familiar with this area, I still need to learn some basic knowledge about socket programming itself at first. I have learned that socket programming is a programming method that allows two or more processes (apps or services) to interact with each other via sockets so that they can exchange data over a network. There are two types of sockets which are client sockets and server sockets. Client sockets are the ones that initiate the connection between the client and the server by specifying the IP address and port they want to connect to. Meanwhile, Server sockets are the ones that listen on specific ports waiting for incoming requests from clients that want to establish a connection. To implement the socket programming, there are APIs for that and we can use them by implementing some programming languages. Since the programming language that I am good at is JavaScript, I will therefore use that to implement this work combined with HTML. Another thing I have to learn is how to code JavaScript and HTML as I have not used them for a while. I, therefore, need some recalls by reading the documentation of how to use them using the W3school website.

Design

To understand better how the web servers actually work and what is happening behind the scenes between client and server, I plan to make simple web pages consisting of one JavaScript server file and three HTML files (index file, response file, and testing file). The JavaScript file will be the server to give some instructions on handling the GET and POST methods during the interaction process between the client and the server. Meanwhile, the HTML file will be displayed as the web pages according to their purposes. The index file will be displayed when the local host is accessed. It consists of a simple form asking users to input any message. Once they submit the message, they will be directed to the response file which sends back the message they submitted by displaying it on the screen. The testing file is displayed when I change the URL path to “localhost/blabla.html” to check if the web server can handle a request to access another HTML besides the index file.

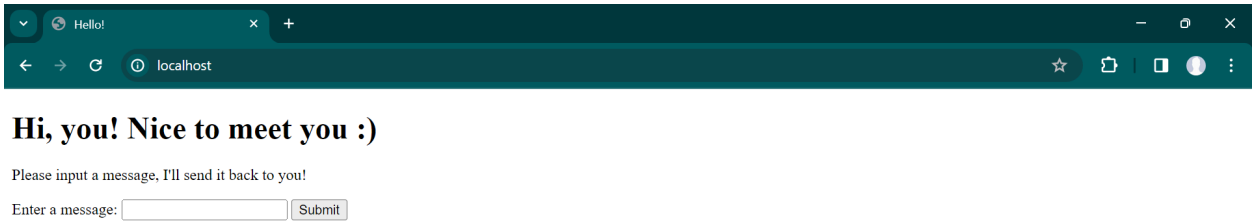
Implementation

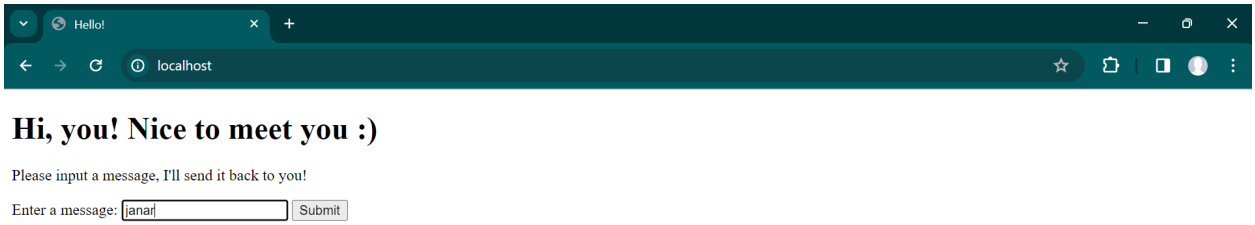
In practice, the part that is the hardest to do is making the server file as I am not used to dealing with server-side programming, especially when dealing with socket programming. Here are the steps I have taken to implement this work.

1. Preparing three HTML files and an additional file (the 404 file) for an error purpose
2. Creating a JavaScript file for the server file using a socket API that NodeJs provides, namely Net Module
3. Running the server with NodeJS installed on my laptop machine

```
PS D:\Cloud Computing> node server.js
Server is listening on port 80...
```

4. I have done some experiments to see what data is sent behind the TCP process.
   a. Accessing the index file via the root path (“localhost/”) and inputting a message





The information logged:

```
Request Received:
POST / HTTP/1.1
Host: localhost
Connection: keep-alive
Content-Length: 13
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
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
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

```
Request Received:
GET /favicon.ico HTTP/1.1
Host: localhost
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9


Request Line: GET /favicon.ico HTTP/1.1
Method: GET
```

The page displayed after clicking the submit button on the index file:

POST Request Received!

Message: message=janar

The information logged:

```
Request Received:
POST / HTTP/1.1
Host: localhost
Connection: keep-alive
Content-Length: 13
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
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
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

```
Message received: message=janar
Request Received:
GET /favicon.ico HTTP/1.1
Host: localhost
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9


Request Line: GET /favicon.ico HTTP/1.1
Method: GET
```

b. Accessing the Homepage file via the specific path (“localhost/Homepage.html”)

Hi, you! Welcome to our homepage!

The information logged:

```
Request Received:
GET / HTTP/1.1
Host: localhost
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
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
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
;v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
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
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9


Request Line: GET /Homepage.html HTTP/1.1
Method: GET
```

```
Request Received:
GET /favicon.ico HTTP/1.1
Host: localhost
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="123", "Not:A-Brand";v="8", "Chromium";v="123"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost/Homepage.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9


Request Line: GET /favicon.ico HTTP/1.1
Method: GET
```

Evaluation

After I had done the experiments previously, I found some points based on what the web server logs and prints to the console or terminal.

1. Since I use Google Chrome as the web browser, the system will eventually generate two requests at once, which are the first one for the client request (GET or POST method) and the second one for the favicon.ico (Several web browsers might use that kind of services).
2. The first request, the one that comes from the client request, always has similar information containing various headers such as ‘Host’, ‘Connection’, ‘User-Agent’, and ‘Accept’.
3. Before the headers, there is a line consisting of the method used (GET or POST), the path, and the protocol.

Conclusion

Before I did this work, I had no clue at all about how web servers handle requests from clients and send them to the servers. A lot of things happen behind the process. Tons of information passed over the network that I was not familiar with. At least, I am getting to know what data exists on every request made, and how they are being processed until the clients receive the appropriate responses from the servers.

Reference

https://www.ibm.com/docs/en/i/7.3?topic=communications-socket-programming
https://www.w3schools.com/js/default.asp
https://www.w3schools.com/html/default.asp