Tugas 1 - SisDis

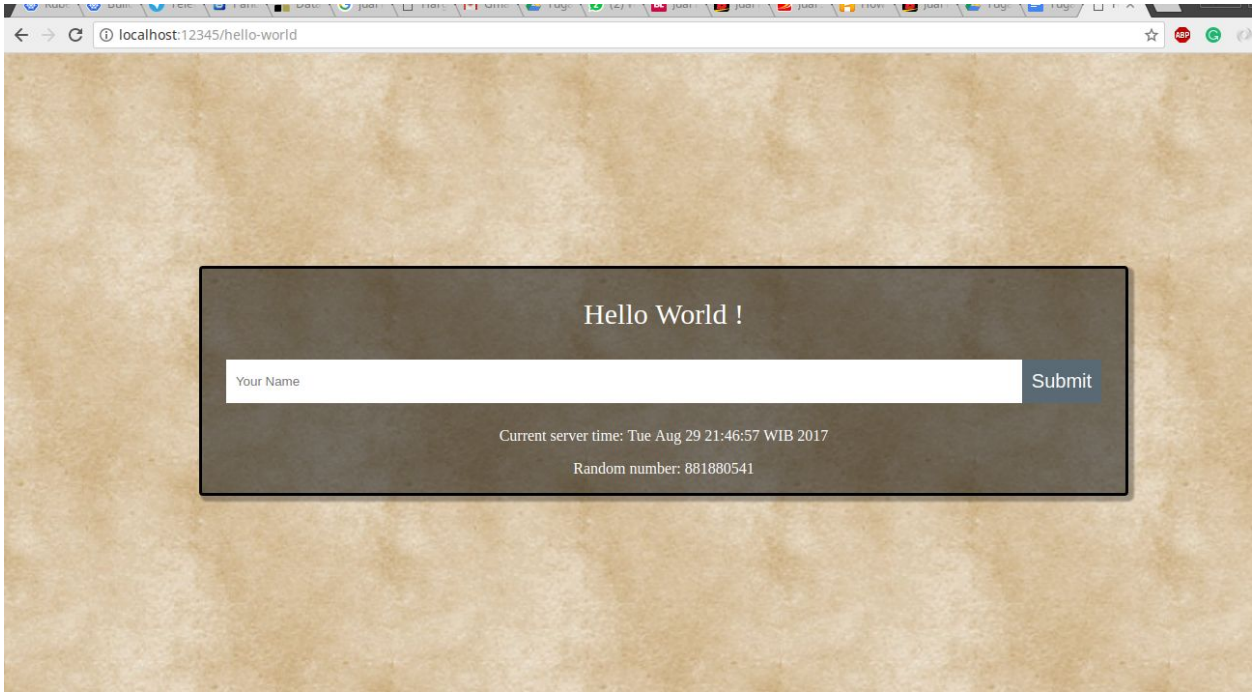
Pada tugas ini anda diharapkan bisa mengerti HTTP protocol (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) secara mendalam, oleh karena itu anda diminta untuk megimplementasikan sebuah HTTP Server.
Anda boleh menggunakan bahasa pemograman apapun untuk menyelesaikan tugas ini, namun **TIDAK** dibolehkan menggunakan library yang sudah menyediakan fitur untuk HTTP protocol, seperti https://hc.apache.org/ di Java, https://golang.org/pkg/http di Golang, https://docs.python.org/2/library/simplehttpserver.html di Python, https://nodejs.org/dist/latest-v8.x/docs/api/http.html di NodeJS, dan sejenisnya.
Implementasi harus menggunakan socket programming.

Anda diberikan 3 file pendukung, yaitu:

1. `background.jpg`
2. `style.css`
3. `hello-world.html`

Silahkan memahami file file tersebut.

Program contoh (untuk coba coba) juga sudah di berikan (dalam bentuk program java yang sudah dicompile, `Main.class`), anda bisa menjalankan `java Main <port>` (contoh: `java Main 12345`) dan setelah itu buka http://localhost:12345.



Hal-hal yang harus dipenuhi

1. Jika request tidak bisa diparsing (misalkan request bukan dalam format HTTP/1.0 atau HTTP/1.1) anda harus mengembalikan HTTP Response **`400 Bad Request`**.
2. Jika HTTP Method selain **`GET`** dan **`POST`** anda harus mengembalikan HTTP Response **`501 Not Implemented`**.
3. Untuk mempermudah pengerjaan tugas, fitur HTTP Persistent Connection (https://en.wikipedia.org/wiki/HTTP_persistent_connection) dan HTTP Pipelining (https://en.wikipedia.org/wiki/HTTP_pipelining) harus dimatikan, dengan kata lain, setiap HTTP Response harus memiliki header **`Connection: close`**.
4. Setiap HTTP Response harus memiliki header **`Content-Type`** dan **`Content-Length`** yang sesuai.
5. Server juga harus bisa parsing HTTP Request header **`Content-Type`** dan **`Content-Length`**.
6. Server harus bisa parsing **`query`** part yang ada di defenisi URL (https://en.wikipedia.org/wiki/URL).
7. **`GET /`** dan **`POST /`** harus mengembalikan HTTP Response **`302 Found`**, dan redirect ke **`/hello-world`**.
8. **`GET /style`** harus mengembalikan isi dari file **`style.css`**.
9. **`GET /background`** harus mengembalikan isi dari file **`background.jpg`**.
10. **`GET /hello-world`** harus mengembalikan isi dari file **`hello-world.html`**, namun **`__HELLO__`** pada baris ke 9 harus diganti dengan string **`World`**.
11. **`POST /hello-world`** harus melakukan parsing dari HTTP Request yang diberikan oleh client, jika **`Content-Type`** dari HTTP Request bukan **`application/x-www-form-urlencoded`**, maka harus mengembalikan **`400 Bad Request`**.
    Jika kondisi itu terpenuhi, maka server harus mengembalikan isi dari file **`hello-world.html`**, namun **`__HELLO__`** pada baris ke 9 harus diganti dengan isi form **`name`** sesuai dari hasil parsing-an dari HTTP Request (dalam format **`x-www-form-urlencoded`** tentunya).
    Hal ini sama persis seperti penggunaan **`$_POST`** ketika menggunakan bahasa pemograman PHP.
12. **`GET /info`** harus bisa menerima query pada URL (https://en.wikipedia.org/wiki/URL), server harus megembalikan HTTP Response dengan **`Content-Type: text/plain; charset=UTF-8`**,
    Jika form **`type`** adalah **`time`**, server harus mengembalikan waktu sekarang.
    Jika form **`type`** adalah **`random`**, server harus megembalikan integer random.
    Jika form **`type`** selain itu (atau tidak di defenisikan di HTTP Request) maka server harus megembalikan string **`No Data`**.
    Untuk info lebih lanjut bisa dilihat di file **`hello-world.html`** pada baris 22 dan 25
    Hal ini sama persis seperti penggunaan **`$_GET`** ketika menggunakan bahasa pemograman PHP.

13. Untuk HTTP Request selain yang di atas, harus mengembalikan HTTP Response `404 Not Found`.

BONUS:

- Implementasi menggunakan multithreading.
- Implementasi dengan selain bahasa JAVA
- Implementasi ETag (https://en.wikipedia.org/wiki/HTTP_ETag) untuk `/style` dan `/background`

TIPS: (dengan asumsi anda menggunakan Java)

- Untuk point (10) dan (11) anda hanya perlu baca file `hello-world.html` (secara keseluruhan, bukan perbaris), simpan ke string, kemudian
  `str.replaceAll("__HELLO__", name);`
  Jangan terlalu dipermasalahkan tentang XSS Security Issue
- Format URL Query (point (12)) juga dalam bentuk `x-www-form-urlencoded` jadi anda bisa membuat fungsi utility untuk itu, reuse dari point (11)
- Gunakan `java.net.URLDecoder` untuk mendekode `x-www-form-urlencoded` (https://en.wikipedia.org/wiki/Percent-encoding), tidak usah bikin code sendiri.
- Karena `InputStream` dan `OutputStream` menggunakan `byte[]`, bukan `String`, gunakan UTF-8 sebagai charset, contoh
  `String str = new String(bytes, "UTF-8");`
  `byte[] bytes = str.getBytes("UTF-8");`
- Gunakan `java.io.BufferedInputStream` untuk parsing HTTP Request, karena class ini memiliki fitur `mark()` dan `reset()` sehingga lebih memudahkan ketika parsing, contoh code `readLine`

```
private static byte[] readLine(BufferedInputStream input) throws Exception {
	ByteArrayOutputStream buffer = new ByteArrayOutputStream();

	// HTTP spec said that \r\n must be used to terminate a line
	// but some OS using only \r, only \n, or both, so we must handle it
	while(true) {
		int data = input.read();
		if(data == -1 || data == '\n') {
			return buffer.toByteArray();
		} else if (data == '\r') {
			// BufferedInputStream always support marking
			input.mark(1);
			data = input.read();
			if(data != '\n') {
				input.reset();
			}
			return buffer.toByteArray();
```

```
            } else {
                  buffer.write(data);
            }
      }
}
```

- Karena HTTP header case-insensitive, maka convertlah dulu ke lower/upper case sebelum di bandingkan dengan string lain, contoh
  `if(str.toLowerCase().equals("content-length")) { ...`
- Gukanan Linux, dan program `curl` untuk testing (bukan browser), karena program ini memperlihatkan detail dari setiap HTTP Header, contoh

```
bash$ curl -XOPTION -v 'http://localhost:12345'
* Rebuilt URL to: http://localhost:12345/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 12345 (#0)
> OPTION / HTTP/1.1
> Host: localhost:12345
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 501 Not Implemented
< Connection: close
< Content-Type: text/plain; charset=UTF-8
< Content-Length: 35
<
* Closing connection 0
501 Not Implemented: Reason: OPTION

bash$ curl -v 'localhost:12345/hello-world' -d "name=John%20Cena"
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 12345 (#0)
> POST /hello-world HTTP/1.1
> Host: localhost:12345
> User-Agent: curl/7.55.1
> Accept: */*
> Content-Length: 16
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 16 out of 16 bytes
< HTTP/1.1 200 OK
```

```
< Connection: close
< Content-Type: text/html; charset=UTF-8
< Content-Length: 893
<
<!DOCTYPE html>
<html>
<head>
      <title>Hello World</title>
      <link rel="stylesheet" href="/style">
</head>
<body>
      <div id="container">
            <p id="title">Hello John Cena !</p>
            <form id="form" method="POST">
                  <input id="form-name" type="text" name="name"
placeholder="Your Name" />
                  <input id="form-submit" type="submit" value="Submit">
            </form>
            <p>Current server time: <span id="time"></span></p>
            <p>Random number: <span id="random"></span></p>
      </div>
      <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></scr
ipt>
      <script type="text/javascript">
            var timeSpan = $("#time");
            var randomSpan = $("#random");
            setInterval(function() {
                  $.get("/info?type=time", function(data) {
                        timeSpan.html(data);
                  });
                  $.get("/info?type=random", function(data) {
                        randomSpan.html(data);
                  });
            }, 2000);
      </script>
</body>
</html>
* Closing connection 0
```