

## LAPORAN PROYEK AKHIR

**PENERAPAN DAN ANALISIS HTTP/2 SERVER-SENT EVENTS DAN  
WEBSOCKET UNTUK WEB APPLICATION PADA SISTEM RUMAH PINTAR**

***IMPLEMENTATION AND ANALYSIS HTTP/2 SERVER-SENT EVENTS AND  
WEBSOCKET FOR WEB APPLICATION ON SMART HOME SYSTEM***



**Disusun oleh :**

**MUHAMMAD RUSMINTO HADIYONO**

**15/386767/SV/10153**

**PROGRAM SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET**

**DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA**

**SEKOLAH VOKASI**

**UNIVERSITAS GADJAH MADA**

**YOGYAKARTA**

**2019**

**PENERAPAN DAN ANALISIS HTTP/2 SERVER-SENT EVENTS DAN  
WEBSOCKET UNTUK *WEB APPLICATION* PADA SISTEM RUMAH PINTAR**

**Proyek Akhir**

**Program Studi Teknologi Rekayasa Internet**

**Diajukan kepada**  
**Departemen Teknik Elektro dan Informatika**  
**Fakultas Sekolah Vokasi Universitas Gadjah Mada**  
**Sebagai syarat kelengkapan studi jenjang Sarjana Terapan (D-IV)**  
**Dalam memperoleh derajat Sarjana Terapan Teknik**  
**Program Studi Teknologi Rekayasa Internet**

**Oleh :**

**MUHAMMAD RUSMINTO HADIYONO**

**15/386767/SV/10153**

**PROGRAM SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET**  
**DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA**  
**SEKOLAH VOKASI**  
**UNIVERSITAS GADJAH MADA**  
**YOGYAKARTA**  
**2019**

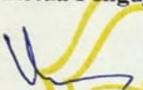
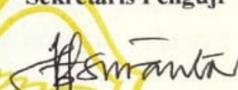
## LEMBAR PENGESAHAN

Judul : Penerapan dan Analisis HTTP/2 Server-Sent Events dan WebSocket untuk *Web Application* pada Sistem Rumah Pintar  
Nama : Muhammad Rusminto Hadiyono  
Program Studi : Teknologi Rekayasa Internet  
Dosen Pembimbing : Muhammad Arrofiq, S.T, M.T, Ph.D.  
Waktu Ujian : Selasa, 23 Juli 2019, Pukul 08:00, Ruang Sidang 1 Sekip Unit IV

Telah dipertanggungjawabkan dan diuji oleh tim penguji serta disetujui dan disahkan  
Sebagai syarat kelengkapan studi jenjang Sarjana Terapan Teknik (S.Tr.T)

Program Studi Teknologi Rekayasa Internet  
Sekolah Vokasi Universitas Gadjah Mada

Yogyakarta, 23 Juli 2019

Diterima dan disetujui oleh,  
**Ketua Penguji**  **Sekretaris Penguji**   
Unan Yusmaniar O, S.T., M.Sc., Ph.D. Hidayat Nur Isnianto, S.T., M.Eng  
NIKA. 111198210201109201 NIP. 197305282002121001

Anggota Penguji / Dosen Pembimbing

  
Muhammad Arrofiq, S.T, M.T, Ph.D.  
NIP. 197311271999031001

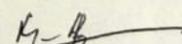
Mengetahui,

**Ketua Departemen**  
**Teknik Elektro dan Informatika**



Nur Rohman Rosyid, S.T., M.T., D.Eng.  
NIKA. 111197510201206101

**Ketua Program Studi**  
**Sarjana Terapan Teknologi**  
**Rekayasa Internet**



Muhammad Arrofiq, S.T, M.T, Ph.D.  
NIP. 197311271999031001

### **PERNYATAAN BEBAS PLAGIASI**

Saya yang bertanda tangan di bawah ini :

Nama : Muhammad Rusminto Hadiyono  
NIM : 15/386767/SV/10153  
Tahun terdaftar : 2015  
Program Studi : Teknologi Rekayasa Internet  
Fakultas/Sekolah : Sekolah Vokasi

Menyatakan bahwa dalam dokumen ilmiah Proyek Akhir ini tidak terdapat bagian dari karya ilmiah lain yang telah diajukan untuk memperoleh gelar akademik di suatu Lembaga Pendidikan Tinggi, dan juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang/ Lembaga lain, kecuali yang secara tertulis disisipkan dalam dokumen ini dan disebutkan sumbernya secara lengkap dalam daftar pustaka.

Dengan demikian saya menyatakan bahwa dokumen ilmiah ini bebas dari unsur-unsur plagiasi dan apabila dokumen ilmiah Proyek Akhir ini di kemudian hari terbukti merupakan plagiasi dari hasil karya penulis lain dan/atau dengan sengaja mengajukan karya atau pendapat yang merupakan hasil karya penulis lain, maka penulis bersedia menerima sanksi akademik dan/atau sanksi hukum yang berlaku.

Yogyakarta, 15 Juli 2019  
  
Muhammad Rusminto Hadiyono

15/386772/SV/10158

## KATA PENGANTAR

Puji dan rasa syukur senantiasa penulis panjatkan atas kehadiran Allah SWT, karena berkat limpahan rahmat, hidayah, dan inayah-Nya Proyek Akhir ini dapat diselesaikan dengan baik. Salam dan sholawat semoga senantiasa selalu tercurahkan pada baginda Rasulullah Muhammad SAW.

Proyek Akhir yang berjudul "*Penerapan dan Analisis HTTP/2 Server-Sent Events dan WebSocket untuk Web Application pada Sistem Rumah Pintar*" ini disusun sebagai syarat untuk mendapatkan gelar Sarjana Terapan (D-IV) Program Studi Teknologi Rekayasa Internet Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada.

Penulis mengucapkan rasa terimakasih yang sebesar-besarnya atas semua bantuan yang sudah diberikan kepada penulis, baik secara langsung maupun tidak langsung selama proses pembuatan dan penyusunan laporan proyek akhir ini hingga selesai. Terlebih secara khusus penulis mengucapkan rasa terimakasih kepada :

1. Bapak, Ibu, serta Saudara yang selalu memberikan doa serta dukungan secara penuh kepada penulis dalam proses masa studi selama 4 tahun ini.
2. Bapak Muhammad Arrofiq, S.T., M.T., Ph.D. selaku Dosen pembimbing proyek akhir, karena sudah berkenan membimbing dan memberikan arahan serta dukungan selama penulis menyelesaikan proyek akhir.

3. Bapak Nur Rohman Rosyid, S.T., M.T., D.Eng. selaku dosen wali yang telah memberikan bimbingan selama penulis mengikuti masa studi di Program Studi D4 Teknologi Rekayasa Internet, Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada.
4. Seluruh dosen serta staff dan karyawan Program Studi D4 Teknologi Rekayasa Internet, Sekolah Vokasi, Universitas Gadjah Mada, atas ilmu, bimbingan dan bantuannya hingga penulis selesai menyusun proyek akhir ini
8. Berbagai pihak yang tidak bisa disebutkan satu persatu yang telah membantu dalam penyelesaian proyek akhir ini.

Terakhir penulis berharap, semoga proyek akhir ini dapat memberikan hal yang bermanfaat dan menambah wawasan bagi pembaca dan khususnya bagi penulis juga.

Yogyakarta, 15 Juli 2019

Penulis

## DAFTAR ISI

<b>HALAMAN SAMPUL.....</b>	<b>i</b>
<b>LEMBAR PENGESAHAN.....</b>	<b>iii</b>
<b>PERNYATAAN BEBAS PLAGIASI.....</b>	<b>iv</b>
<b>KATA PENGANTAR.....</b>	<b>v</b>
<b>DAFTAR ISI.....</b>	<b>vii</b>
<b>DAFTAR GAMBAR.....</b>	<b>ix</b>
<b>DAFTAR TABEL.....</b>	<b>x</b>
<b>ABSTRAK.....</b>	<b>xi</b>
<b>ABSTRACT.....</b>	<b>xii</b>
<b>BAB I PENDAHULUAN.....</b>	<b>1</b>
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	3
1.3 Batasan Masalah.....	3
1.4 Tujuan Penelitian.....	4
1.5 Manfaat Penelitian.....	4
1.6 Sistematika Penulisan.....	5
<b>BAB II TINJAUAN PUSTAKA.....</b>	<b>6</b>
2.1 Mikrokontroler.....	6
2.2 Konsep <i>Internet of Things</i> .....	7
2.3 Rumah Pintar.....	8
2.4 <i>Message Queuing Telemetry Transport</i> .....	8
2.5 <i>Binary Protocol</i> dan <i>Plain Text Protocol</i> .....	9
2.6 <i>Hypertext Transfer Protocol</i> .....	10
2.7 <i>Server-Sent Events</i> .....	13
2.8 <i>WebSocket</i> .....	14
2.9 <i>Response Time</i> .....	16
2.10 Hipotesis.....	19
<b>BAB III METODE PENELITIAN.....</b>	<b>20</b>

3.1 Peralatan.....	20
3.2 Bahan.....	21
3.3 Tahapan Penelitian.....	22
3.4 Perancangan Topologi dan Model.....	24
3.4.1. Perancangan Topologi.....	24
3.4.2. Perancangan Model.....	31
3.5 Pembuatan <i>Web API</i> serta <i>Web Server</i> .....	33
3.5.1 Penerapan Node.js serta Vue.js.....	34
3.5.2 Penerapan <i>WebSocket</i> dan <i>Server-Sent Events</i> .....	35
3.5.3 Penerapan TLS.....	36
3.6 Metode Pengambilan Data.....	36
3.6.1 <i>Response Time</i> .....	37
3.6.2 Penggunaan CPU.....	39
<b>BAB IV HASIL PENELITIAN DAN PEMBAHASAN.....</b>	<b>42</b>
4.1 Hasil Perbandingan <i>Response Time</i> .....	42
4.1.1 Skenario Jaringan Lokal.....	43
4.1.2 Skenario Jaringan Internet.....	44
4.2 Hasil Perbandingan Penggunaan CPU.....	46
<b>BAB V PENUTUP.....</b>	<b>48</b>
5.1 Kesimpulan.....	48
5.2 Saran.....	48
<b>DAFTAR PUSTAKA.....</b>	<b>49</b>
<b>LAMPIRAN.....</b>	<b>52</b>

## DAFTAR GAMBAR

<b>Gambar 2.1 Berbagai metode pengiriman data ke <i>web browser</i>.....</b>	<b>12</b>
<b>Gambar 3.1 Diagram alir penelitian.....</b>	<b>22</b>
<b>Gambar 3.2 Bagan Topologi Perangkat.....</b>	<b>25</b>
<b>Gambar 3.3 Bagan Topologi Data.....</b>	<b>27</b>
<b>Gambar 3.3 <i>Activity Diagram</i> sistem.....</b>	<b>32</b>
<b>Gambar 3.4 <i>Use case diagram</i> web server.....</b>	<b>33</b>
<b>Gambar 3.5 Tampilan pengujian <i>response time</i> melalui Serveo.....</b>	<b>37</b>
<b>Gambar 3.6 Pengaktifan Serveo melalui Raspberry Pi.....</b>	<b>38</b>
<b>Gambar 3.7 Pemilahan aplikasi node di dalam htop.....</b>	<b>39</b>
<b>Gambar 3.8 Hasil perintah top dengan pemilahan pid.....</b>	<b>40</b>
<b>Gambar 3.9 Hasil perintah top secara normal.....</b>	<b>40</b>
<b>Gambar 4.1 Grafik rata - rata <i>response time</i> pada skenario jaringan lokal.....</b>	<b>43</b>
<b>Gambar 4.2 Grafik rata - rata <i>response time</i> pada skenario jaringan internet.....</b>	<b>44</b>
<b>Gambar 4.3 Grafik rata - rata penggunaan CPU.....</b>	<b>46</b>

## **DAFTAR TABEL**

<b>Tabel 2.1 Ringkasan tinjauan pustaka.....</b>	<b>17</b>
<b>Tabel 4.1 Rata - rata <i>response time</i> pada skenario lokal.....</b>	<b>44</b>
<b>Tabel 4.2 Rata - rata <i>response time</i> pada skenario internet.....</b>	<b>45</b>

## **ABSTRAK**

### **PENERAPAN DAN ANALISIS HTTP/2 SERVER-SENT EVENTS DAN WEBSOCKET UNTUK *WEB APPLICATION* PADA SISTEM RUMAH PINTAR**

Salah satu hal yang sering menjadi kendala dalam penerapan sistem kendali rumah pintar berbasis *website* adalah cara mereka agar bisa terhubung dengan perangkat apapun di dalam rumahnya melalui internet. Kendala tersebut dapat diselesaikan dengan memanfaatkan beberapa teknologi sekaligus, semisal dengan menggabungkan WebSocket atau HTTP/2 Server-Sent Events (SSE) dengan MQTT serta Serveo. Di antara WebSocket serta HTTP/2 SSE perlu dilakukan pemilihan metode pengiriman yang lebih *real-time* untuk digunakan pada sistem kendali rumah pintar. Dengan alasan tersebut, pengujian *response time* dilakukan dengan memvariasikan jeda antar pengiriman perintah yang dimasukkan melalui *website*. Selain itu, perangkat yang digunakan untuk membangun sistem kendali rumah pintar cenderung sederhana sehingga dibutuhkan metode pengiriman yang memiliki nilai penggunaan CPU terkecil. Oleh karena itu, pengujian penggunaan CPU dilakukan dengan memvariasikan jumlah pengakses *website*. Hasil yang didapatkan adalah WebSocket memiliki nilai rata - rata *response time* terkecil sebesar 1154.5 milidetik ketika melalui internet dan dengan berkurangnya jeda antar pengiriman menyebabkan bertambahnya nilai *response time*. Selain itu, HTTP/2 SSE memiliki nilai presentase penggunaan CPU terkecil dalam pengujian pengiriman dua arah serta dengan bertambahnya jumlah pengguna dapat menaikkan presentase penggunaan CPU di *server*.

Kata Kunci : *Internet of Things*, WebSocket, *Server-Sent Events*, HTTP/2, *smart home*.

## ***ABSTRACT***

### ***IMPLEMENTATION AND ANALYSIS HTTP/2 SERVER-SENT EVENTS AND WEBSOCKET FOR WEB APPLICATION ON SMART HOME SYSTEM***

*One of the obstacle to create a smart home is how to control their system with internet. Many ways can be used to reach that goal, such as combining WebSocket or Server-Sent Events (SSE), MQTT with Serveo. Among those methods (WebSocket and HTTP/2 SSE), we still need to choose a method that more real-time. With that reason, response time examination was done by vary the delay between transmission data. Moreover, devices that used for create smart home tend to have low specifications. Therefore CPU usage examination was done by vary the amount of users. The result from examination is WebSocket has the lowest response time average compared to other methods with value 1154.5 milliseconds and reduction in intervals between transmission can increase response time values. Furthermore, HTTP/2 SSE has the lowest CPU usage percentage on two-ways transmission and increment of users can increase CPU usage percentage.*

*Keywords : Internet of Things, WebSocket, Server-Sent Events, HTTP/2, smart home.*

## **BAB I**

### **PENDAHULUAN**

#### **1.1 Latar Belakang**

Dalam menghadapi era Industri 4, Indonesia sudah memiliki banyak perkembangan di bidang teknologi dibandingkan era sebelumnya terutama di dalam pemanfaatan internet. Hal ini terlihat dari data statistik pengguna internet yang diterbitkan oleh Asosiasi Penyelenggara Jasa Internet Indonesia, pada tahun 2016 telah terdapat 132,7 juta pengguna sedangkan pada tahun 2017 sudah terdapat 143,26 juta pengguna dari total populasi penduduk Indonesia sebanyak 262 juta orang (APJII, 2017). Pesatnya pertumbuhan pengguna internet disebabkan oleh semakin cepatnya proses pengiriman data melalui internet serta semakin mudahnya cara untuk mendapatkan akses internet. Hal ini pula yang mendorong semakin beragamnya perangkat yang mampu terhubung dan saling terintegrasi atau lebih dikenal dengan istilah *Internet of Things*.

Konsep yang paling penting dari *Internet of Things* adalah mengintegrasikan semua hal yang ada di dunia nyata ke dalam dunia *digital* (Kwan et al. 2016). Salah satu penerapan dari *Internet of Things* adalah pengendalian rumah pintar melalui *web browser*. Hal yang perlu diperhatikan dalam pembuatan rumah pintar adalah kecepatan transaksi informasi, yang mana sebaiknya memiliki nilai *response time* serendah mungkin (Saito and Menga 2015). Untuk mendapatkan nilai *response time* yang rendah, dibutuhkan infrastruktur jaringan yang bagus serta proses pengiriman data yang cepat dan tepat.

Walaupun demikian, penerapan rumah pintar seringkali menggunakan infrastruktur jaringan seadanya serta menggunakan dana seminimal mungkin.

Dalam proses pengiriman data menuju *web browser* dengan rentang waktu sekecil mungkin, terdapat berbagai pilihan yang dapat diterapkan pada rumah pintar, seperti halnya *Polling*, *Long-Polling*, *WebSocket* dan *Server-Sent Events*. Dari beberapa pilihan tersebut, teknologi *Polling* dan *Long-Polling* kurang cocok digunakan dalam sistem kendali rumah pintar karena keduanya membutuhkan *bandwidth* dan *response time* yang besar, berbeda halnya dengan *WebSocket* ataupun *Server-Sent Events* (Souders 2009).

*WebSocket* serta *Server-Sent Events* memungkinkan *web browser* menerima data dari *server* tanpa perlu *request* data baru setiap ada data baru, sehingga data yang telah tersedia di *server* akan dapat langsung dikirimkan ke *web browser*. Dengan berkurangnya waktu yang dibutuhkan untuk mengirimkan data, keduanya mampu untuk lebih *real-time* dibandingkan ketika menggunakan metode *Polling* maupun *Long-Polling*.

Server-Sent Events dapat diterapkan dengan menggunakan protokol HTTP/1.1, HTTP/1.1 over TLS (HTTPS) serta HTTP/2. Protokol HTTP/2 merupakan protokol baru yang masih jarang digunakan namun memiliki berbagai keunggulan yang telah mampu mengatasi permasalahan pada HTTP/1.1.

Di sisi lain, proses pengiriman data pada mikrokontroler juga memiliki pengaruh pula terhadap nilai *response time* dari *web browser*. Maka dari itu, dibutuhkan metode pengiriman yang cepat dan tepat untuk *Machine-to-Machine*. Dari beberapa protokol *Machine-to-Machine*, protokol MQTT yang berarsitektur *publish-broker-subscribe*

memiliki rata-rata *response time* yang tergolong rendah (Kayal and Perros 2017). Dengan menggabungkan arsitektur *publish-broker-subscribe* yang dimiliki oleh MQTT dan kelebihan yang dimiliki *WebSocket* ataupun *Server-Sent Events*, proses pengiriman data dari piranti – piranti elektronik yang berada di dalam rumah menuju *web browser* akan berlangsung lebih cepat terutama ketika melalui internet dengan menggunakan Serveo.

Selain dari kecepatan transaksi data, rumah pintar cenderung dibuat secara sederhana. Karena itu, *server* yang digunakan untuk rumah pintar seringkali memiliki spesifikasi CPU yang lebih rendah. Dalam pembuatan sistem kendali rumah pintar, metode pengiriman data juga dapat memengaruhi nilai presentase penggunaan CPU di *server*.

## 1.2 Rumusan Masalah

Rumusan permasalahan pada penelitian ini adalah bagaimana cara menerapkan HTTP/2 *Server-Sent Events* ataupun *WebSocket* pada sistem rumah pintar berbasis website serta untuk mengetahui hasil perbandingan *response time* serta presentase penggunaan CPU pada HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE dan *WebSocket*.

## 1.3 Batasan Masalah

Beberapa batasan masalah yang akan dilakukan selama proses penelitian proyek akhir adalah sebagai berikut :

1. Piranti elektronik yang terpasang dengan mikrokontroler bukanlah piranti elektronik yang benar-benar digunakan pada kehidupan sehari-hari di dalam rumah.

2. Dikarenakan kebutuhan pengiriman data dua arah dari *web browser* maupun dari *web API, Server-Sent Events* tidak diuji secara satu arah tetapi pengujian dilakukan secara dua arah dengan bantuan *request POST* sesuai dengan protokol yang digunakan.
3. Tidak membahas keamanan pada jaringan maupun data.

#### **1.4 Tujuan Penelitian**

Tujuan dari penelitian dalam proyek akhir ini adalah mengimplementasikan serta membandingkan metode pengiriman data melalui HTTP/2 SSE serta WebSocket dari rumah pintar menuju *browser* pengguna dan sebaliknya menggunakan parameter *response time* serta presentase penggunaan CPU.

#### **1.5 Manfaat Penelitian**

Manfaat yang dapat diambil dari penelitian dan penggerjaan proyek akhir ini adalah sebagai berikut :

1. Memberi informasi mengenai implementasi HTTP/2 *Server-Sent Events*, serta *WebSocket* dari *server API* menuju *web browser*.
2. Memberi informasi mengenai cara mudah mengawasi dan mengubah status perangkat dari jarak jauh.
3. Hasil perbandingan *response time* dari ketika *web browser* meminta sampai mendapatkan data dari *server API* dengan metode yang berbeda (HTTP/2 *Server-Sent Events* dan *WebSocket*).
4. Hasil perbandingan presentase penggunaan CPU server ketika diterapkan metode pengiriman yang berbeda (HTTP/2 *Server-Sent Events* dan *WebSocket*).

## **1.6 Sistematika Penulisan**

Untuk menggambarkan secara menyeluruh mengenai masalah yang akan dibahas dalam laporan proyek akhir ini, maka dibuat sistematika penulisan yang terbagi dalam lima bab :

BAB I, PENDAHULUAN, memuat latar belakang masalah, perumusan masalah, batasan masalah, tujuan penulisan, kegunaan penulisan dan sistematika penulisan.

BAB II, TINJAUAN PUSTAKA, berupa uraian sistematis tentang informasi yang relevan dan mutakhir yang terkait dengan lingkup materi penelitian atau teknologi yang akan diterapkan. Uraian dalam tinjauan pustaka ini selanjutnya menjadi dasar teori yang digunakan oleh penulis dalam melaksanakan penelitian dan menyajikan argumentasi dalam pembahasan hasil penelitian.

BAB III, BAHAN DAN METODE PENELITIAN, memuat bahan, peralatan, tahapan penelitian, dan rancangan sistem serta analisis data yang ada pada penelitian ini.

BAB IV, ANALISIS HASIL DAN PEMBAHASAN, memuat semua temuan ilmiah yang diperoleh sebagai data hasil penelitian, atau hasil unjuk kerja prototipe yang dibuat. Pada bagian ini peneliti menyusun secara sistematis disertai argumentasi yang rasional tentang hasil unjuk kerja yang diperoleh dari hasil penelitian.

BAB V, PENUTUP, memuat kesimpulan serta saran dari penelitian yang dilakukan pada proyek akhir ini.

## BAB II

### TINJAUAN PUSTAKA

#### 2.1 Mikrokontroler

*Microprocessor* telah banyak membantu pekerjaan manusia, baik dalam hal pengendalian suatu peralatan maupun pemantauan. *Microprocessor* dapat diibaratkan sebagai otak dari suatu komputer, yang berisi *Control Unit* (CU) dan *Aritmetic and Logic Unit* (ALU). Dalam penggunaannya, *microprocessor* yang dikenal sebagai *single-chip computer* memerlukan *chip* tambahan untuk dapat bekerja. Sebagai hasilnya, banyak *chip* yang perlu disatukan dan membutuhkan daya yang semakin besar dalam penggunaannya. Di samping itu dengan banyaknya *chip* yang digunakan, biaya yang diperlukan dalam pembuatan pun juga akan semakin mahal dan pemecahan masalah akan semakin sulit dikarenakan rumitnya sambungan antar *chip* (Ibrahim 2014).

Untuk menyelesaikan permasalahan tersebut dibuatlah mikrokontroler, yang dibuat dengan merangkai *microprocessor* bersama dengan *chip-chip* tertentu menjadi sebuah *chip* sehingga mempermudah dalam penggunaan maupun perawatannya. Sebuah mikrokontroler terdiri dari sebuah atau beberapa *microprocessor*, *memory*, ADC (*Analog-to-Digital Controller*), DAC (*Digital-to-Analog Controller*), *Parallel I/O interface*, *Serial I/O interface*, serta penghitung waktu. Dalam penerapannya, mikrokontroler telah dapat ditemukan pada berbagai peralatan elektronik yang telah digunakan sehari-hari seperti mesin cuci, *printer*, *keyboard* maupun beberapa komponen mesin mobil (Udayashankara and S Mallikarjunaswamy 2009).

Saat ini mikrokontroler telah mampu terhubung dengan jaringan internet. Dengan terhubungnya mikrokontroler dengan internet, suatu mikrokontroler mampu mengirim maupun menerima data melalui jaringan menuju sistem lain yang mampu mengelola data-data tersebut untuk ditampilkan ke berbagai antarmuka maupun disimpan. Konsep inilah yang dinamakan *Internet of Things*.

## 2.2 Konsep *Internet of Things*

*Internet of Things* merupakan penamaan atas suatu sistem yang menghubungkan suatu atau beberapa *smart object* dengan *smart object* lainnya ataupun dengan sistem informasi lainnya melewati jaringan IP (*Internet Protocol*) (Cirani et al. 2018). Setiap *smart object* terdiri dari *microprocessor*, perangkat komunikasi, sensor atau aktuator serta sumber energi listrik. *Microprocessor* berguna untuk memberikan kemampuan komputasi pada *smart object*. Perangkat komunikasi memungkinkan *smart object* untuk berkomunikasi dengan *smart object* lainnya ataupun sistem lainnya. Sensor atau aktuator menghubungkan *smart object* dengan lingkungannya, memungkinkan mereka untuk mengukur besaran fisis tertentu sampai halnya mengendalikan obyek tertentu. Sumber energi listrik dibutuhkan untuk menjalankan perangkat elektronik pada *smart object*, yang mana dapat berupa baterai ataupun dari sumber energi listrik lainnya (Vasseur et al. 2010). *Internet of Things* dapat diterapkan pada berbagai skenario seperti rumah pintar, *smart cities* serta pengolahan agrikultur (Cirani et al. 2018).

### 2.3 Rumah Pintar

Rumah pintar adalah istilah yang digunakan untuk sekumpulan perangkat yang digunakan dalam kehidupan sehari-hari yang saling terhubung dalam suatu jaringan. Perangkat yang terhubung bisa berupa lampu, kunci pintu, gerbang, pintu garasi, *DVD player*, CCTV, sensor suhu, sensor asap sampai halnya *laptop* ataupun *server*. Dengan menerapkan rumah pintar, status atau informasi yang dimiliki dari setiap perangkat dapat diperoleh ataupun diubah melalui perangkat lainnya (Briere and Hurley 2011).

Dalam penerapannya, terdapat banyak pilihan protokol yang dapat digunakan untuk mengendalikan *smart object* yang berada di dalam rumah menuju *server*. Walaupun demikian, dalam penyusunan rumah pintar dibutuhkan protokol yang memiliki proses pengiriman yang cepat dan tepat. Protokol MQTT adalah salah satu protokol yang sering digunakan untuk keperluan pengiriman data *machine-to-machine*.

### 2.4 Message Queuing Telemetry Transport

*Message Queuing Telemetry Transport* (MQTT) adalah protokol yang ringan dan bekerja dengan sistem *publish-broker-subscribe*. Protokol MQTT bekerja di atas protokol *Transmission Control Protocol / Internet Protocol* (TCP/IP). MQTT sangat cocok digunakan untuk pengiriman data yang bersifat *real-time* (Hillar 2017). Selain itu, dalam penerapannya semakin singkat jeda antar pengiriman data melalui protokol MQTT semakin kecil nilai *delta time*-nya serta nilai integritas data yang dikirim dan diterima mencapai 100% (Rochman, Primananda, and Nurwasito 2017). Walaupun demikian, protokol MQTT tidak bisa digunakan secara langsung ke *web browser*, sehingga protokol

MQTT perlu dibungkus dengan *WebSocket* atau dilewatkan ke *server* lain untuk mengubah protokol MQTT ke HTTP. Di dalam penerapannya, membungkus MQTT dengan *WebSocket* dapat diwujudkan dengan mudah. Namun dalam kasus-kasus tertentu, semisal diperlukannya *Application Programming Interface* (API) untuk proses penyimpanan, pengumpulan, serta pengolahan data maka pengubahan metode pengiriman dari MQTT menuju ke HTTP untuk ditampilkan ke *web browser* diperlukan. Selain HTTP masih banyak protokol yang dapat digunakan untuk mengirimkan data dari *web API* menuju *web browser*. Protokol – protokol tersebut memiliki berbagai karakteristik yang berbeda, namun berdasarkan format data yang dikirimkan protokol-protokol tersebut dapat dikategorikan menjadi dua, yakni *Plain Text Protocol* dan *Binary Protocol*.

## **2.5 *Binary Protocol* dan *Plain Text Protocol***

*Plain Text Protocol* adalah protokol yang dapat dengan mudah dibaca oleh manusia, contohnya adalah HTTP/1.1 dan SMTP. Sedangkan *Binary Protocol* adalah protokol yang ditujukan untuk dibaca langsung oleh mesin dengan tujuan mempercepat proses penafsiran data. Contoh dari protokol yang termasuk kategori *Binary Protocol* adalah HTTP/2 serta *WebSocket* (Rhee and Hyun Yi 2015). Data yang dikirimkan melalui *Binary Protocol* cenderung lebih kecil daripada ketika dikirimkan melalui *Plain Text Protocol*. Hal ini dikarenakan data yang dikirimkan melalui *Binary Protocol* lebih berorientasi ke struktur data dibandingkan *Plain Text Protocol* yang lebih cenderung ke *Text String*. Dengan adanya susunan yang lebih mengutamakan struktur data, dapat memungkinkan pengiriman angka dalam bentuk *integer* bukan sebagai beberapa karakter dilakukan. Tidak hanya

*integer* saja, tetapi hal ini juga berlaku untuk berbagai tipe data lainnya yang telah ditetapkan oleh suatu *Binary Protocol*. Dari kedua kategori tersebut, HTTP adalah protokol komunikasi yang termasuk ke dalam keduanya.

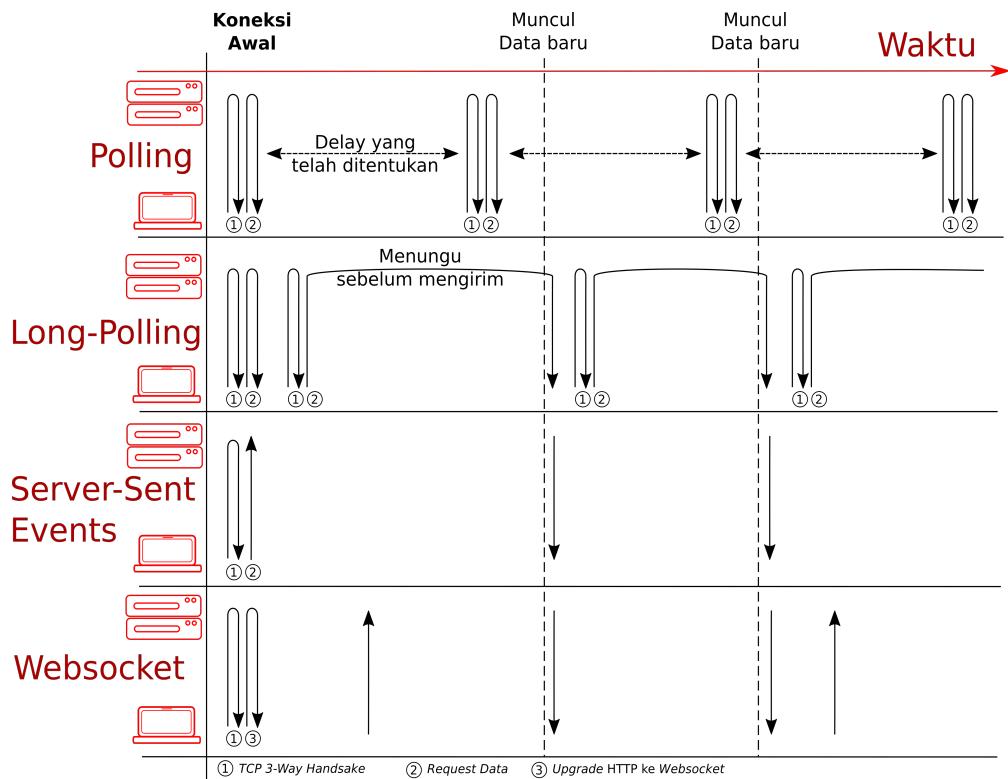
## 2.6 Hypertext Transfer Protocol

*Hypertext Transfer Protocol* (HTTP) adalah protokol komunikasi yang berguna untuk komunikasi antara *web browser* dengan *web server* yang telah digunakan oleh *World-Wide Web* sejak 1990. Protokol HTTP berada pada *application-layer* di dalam *OSI Layer*. Dalam perkembangannya, HTTP memiliki beberapa generasi. Diawali dengan HTTP/0.9 yang berupa protokol sederhana untuk pengiriman *raw data* di *internet*. Setelah itu muncul generasi berikutnya yakni HTTP/1.0 yang memungkinkan pengiriman informasi dalam *format* seperti *MIME*. Sayangnya, HTTP/1.0 tidak sesuai dengan jaringan yang memakai *proxy*, *caching*, ataupun *virtual host* serta jaringan yang membutuhkan koneksi yang bertahan lama (Leach et al. 1999).

Kemudian muncul HTTP/1.1, HTTP/1.1 telah memperbaiki masalah yang ada pada HTTP/1.0. Dengan wajib menyertakan *Host header*, memungkinkan HTTP/1.1 untuk melakukan *virtual hosting* ataupun melayani beberapa pengguna yang berbeda pada sebuah alamat IP. Keunggulan dari HTTP/1.1 daripada HTTP/1.0 adalah munculnya *OPTIONS method*, *upgrade* pada *header*, pemampatan dengan *transfer-encoding* maupun *pipelining* (*Ludin and Garza 2017*). Pada generasi ini pula, muncul *WebSocket* yang memanfaatkan kemampuan *upgrade* pada *header* HTTP/1.1.

Setelah HTTP/1.1 bertahan cukup lama, munculah HTTP/2 yang memungkinkan penggunaan jaringan yang lebih efisien dengan melakukan pemampatan *header* menggunakan HPACK . Selain itu, HTTP/2 mampu menangani *Head of Blocking* yakni kejadian ketika data yang diterima dari *server* harus mengantre untuk bisa dimuat pada halaman HTML. Hal ini mengakibatkan beberapa data yang dikirim maupun diterima dapat dilakukan dalam satu waktu tanpa saling menghalangi dengan kata lain HTTP/2 mampu melakukan *multiplexing*. HTTP/2 juga tidak membutuhkan koneksi tambahan untuk memungkinkan koneksi paralel dan hanya cukup menggunakan satu koneksi HTTP/2 (Peon and Ruellan 2015). Sebagai informasi tambahan, saat ini HTTP/2 masih harus menggunakan koneksi yang aman (TLS/SSL).

Dalam kasus pemantauan sensor maupun status aktuator terbaru dari *web browser*, dibutuhkan metode pengambilan data terbaru dari *server* secara terus-menerus. Pengambilan data terbaru secara terus-menerus dapat dilakukan menggunakan metode *Polling*, *Long-Polling*, *Server-Sent Events*, maupun *WebSocket*. Dari beberapa pilihan tersebut, perlu dipilih metode yang membutuhkan penggunaan memori dan CPU serendah mungkin dan juga tidak menggunakan banyak *bandwidth*. Untuk cara kerja masing-masing metode dapat diamati pada Gambar 2.1.



Gambar 2.1 Berbagai metode pengiriman data ke *web browser*

Metode *Polling* atau yang lebih sering dikenal dengan ‘AJAX’ bekerja dengan cara *web browser* melakukan permintaan data dalam setiap kurun waktu yang telah ditentukan. Sehingga apabila tersedia data baru di sisi server, maka data tersebut tidak akan langsung diterima oleh *web browser*. Dengan data yang tidak langsung diterima, dapat mengakibatkan kenaikan nilai *response time* yang dibutuhkan untuk setiap data baru yang diterima. Selain itu, dengan begitu banyaknya permintaan data untuk setiap data baru yang

diterima, metode ini dapat memakan *bandwidth* lebih banyak dibandingkan metode-metode lainnya.

Untuk mengatasi data yang tidak langsung diterima pada metode *Polling*, munculah metode *Long Polling*. Metode ini bekerja dengan melakukan permintaan data terlebih dahulu dan dilanjutkan dengan menunggu tersedianya data baru dari sisi *server*. Apabila data baru telah diterima, *web browser* akan melakukan permintaan data lagi sampai mendapatkan data terbaru dari *server*. Walaupun demikian, metode ini masih memakan banyak *bandwidth*, apalagi dalam kondisi ketika *server* menyediakan data baru setiap beberapa *milliseconds*. Masalah lain yang timbul ketika menggunakan metode *Long-Polling* adalah ketika terjadi beberapa permintaan data dari satu *web browser* secara serentak, hal ini dapat menyebabkan data yang diterima saling tumpang tindih ataupun terjadinya duplikasi data secara berlebih. Selain itu, *Long-Polling* memperumit arsitektur server ketika digunakan pada beberapa server yang saling berbagi kondisi *session's client* (Abyl 2018). Metode lain yang dapat digunakan selain *Polling* dan *Long Polling* pada protokol HTTP adalah *Server-Sent Events*.

## 2.7 *Server-Sent Events*

*Server-Sent Events* memungkinkan *server* untuk mengirimkan data baru ke *web browser* setiap muncul data baru dari sisi *server* (*streaming*). Berbeda dengan *Long-Polling* maupun *Polling*, metode ini tidak perlu melakukan permintaan data untuk setiap data baru yang muncul pada *server* sehingga dapat mengurangi penggunaan *bandwidth*, CPU serta memori berlebih (Örnmyr and Appelqvist 2017). Setiap kali *web browser* mengirimkan

permintaan data ke *server*, metode ini mampu membuka koneksi selama *server* tidak menghentikkannya. Selama itu pula, data baru yang tersedia di *server* dapat langsung dikirimkan ke *web browser* dalam bentuk potongan-potongan data (*chunk*).

Kemampuan lain yang dimiliki oleh *Server-Sent Events* adalah kemampuan untuk terhubung kembali apabila terjadi putus koneksi antara *web browser* dengan *server*. Namun karena itu pula, server tidak mampu mengetahui kapan *web browser* terputus tanpa mencoba mengirimkan data terlebih dahulu. Selain itu, *Server-Sent Events* pada *web browser* yang diterapkan menggunakan bahasa pemrograman Javascript (*EventSource object*) tidak mampu melakukan penambahan *field* pada *headers*, *field* yang digunakan akan selalu sama yakni hanya berisikan “Content-Type: text/event-stream”. Tidak mampu mengubah *headers* berdampak pada ketidakmampuan *web browser* menggunakan *Authorization* yang berguna untuk memastikan keamanan pengiriman data ataupun fungsi *field headers* lainnya. Selain itu, dalam penerapannya di HTTP/1.1, *Server-Sent Events* hanya terbatas memiliki enam koneksi yang terbuka dalam satu *web browser*. Namun, dengan dikeluarkannya HTTP/2 koneksi yang mampu terbuka di saat bersamaan semakin bertambah. Sebagai catatan tambahan, Internet Explorer tidak mendukung *Server-Sent Events* maupun *WebSocket* (Elman and Lavin 2014).

## 2.8 *WebSocket*

Protokol *WebSocket* memungkinkan komunikasi *full duplex* antara *web browser* dengan *server* melalui jaringan internet. Dalam penerapannya, protokol *WebSocket* perlu melakukan permintaan *upgrade* koneksi pada protokol HTTP/1.1. Hal ini dilakukan guna

berganti jalur dari HTTP ataupun HTTPS menuju protokol *WebSocket*. Apabila *server* memutuskan untuk *upgrade* koneksi, *server* akan mengirimkan pesan "101 Switching Protocols", namun jika *server* menolak maka permintaan akan diabaikan. Setelah protokol berpindah ke protokol *WebSocket*, *handshake* pembuka akan dilakukan. Pada *WebSocket*, *handshake* pembuka berupa membukanya suatu koneksi baru. Setelah terbukanya koneksi baru, *server* maupun *web browser* dapat saling bertukar pesan (MDN 2019).

Untuk menggunakan protokol *WebSocket* pada suatu aplikasi, dibutuhkanlah *WebSocket API*. *WebSocket API* dikembangkan oleh *World Wide Web Consortium* (W3C) sedangkan untuk protokol *WebSocket* dikembangkan oleh *Internet Engineering Task Force* (IETF). Dengan menggunakan *WebSocket API*, tindakan untuk membuka dan menutup koneksi, mengirim pesan, maupun menangkap pesan dari *server* melalui protokol *WebSocket* dapat dilakukan (Wang, Salim, and Moskovits 2013). Salah satu kelebihan protokol *WebSocket* adalah memiliki rata-rata *response time* lebih rendah dalam kondisi pengguna *resource CPU* yang terus naik (Kayal and Perros 2017). Walaupun demikian penggunaan protokol *WebSocket* secara langsung tanpa menggunakan (TLS/SSL) rentan terhalangi oleh *proxy server*. Namun hal ini tidak berlaku pada *Secure WebSocket* maupun HTTP/2 dikarenakan data telah terenkripsi dengan TLS/SSL (Ramlil, Jarin, and Suryadi 2018).

Dalam penerapannya baik *WebSocket* maupun *Server-Sent Events*, metode yang digunakan untuk pengiriman data dapat mempengaruhi nilai *response time*. Penelitian untuk menguji *delay (one-way trip)* antara *WebSocket* dengan *Server-Sent Events* juga

pernah dilakukan dan diperoleh hasil bahwa adalah rata-rata *delay* dari penggunaan *Server-Sent Events* lebih kecil dibandingkan ketika menggunakan *WebSocket* (Muhammad, Yahya, and Basuki 2018).

### **2.9 Response Time**

*Response Time* merupakan parameter yang digunakan untuk mengukur waktu yang dibutuhkan oleh *web browser* untuk mendapatkan balasan dari *server*. Nilai *response time* dapat dipengaruhi oleh banyak hal. Beberapa faktor yang mempengaruhi nilai *response time* adalah infrastruktur jaringan, lama pengolahan data maupun *web browser* yang digunakan (Estep 2013). Dalam penelitian ini, menghitung *response time* berarti menghitung lamanya waktu yang dibutuhkan dari saat pengguna mengubah status perangkat dari *web browser* sampai mendapatkan status perangkat terbaru dari mikrokontroler.

**Tabel 2.1 Ringkasan tinjauan pustaka**

No	Judul Penelitian	Penulis & Tahun	Metode	Tipe Penelitian	Pengiriman Data	Kesimpulan
1	<i>A Comparison of IoT Application Layer Protocols Through a Smart Parking Implementation</i>	(Paridhika Khayal, dkk. 2017)	Membandingkan <i>response time</i> untuk protokol MQTT, CoAP, XMPP dan MQTT melalui WebSocket	Simulasi	Ubuntu 14.04 (MQTT, CoAP, XMPP dan MQTT-WS)	Ketika penggunaan <i>resource server</i> dinaikkan, <i>response time</i> WebSocket relatif tetap
2	<i>A Real-time Application Framework for Speech Recognition Using HTTP/2 and SSE</i>	(Kalamullah Ramli, dkk. 2018)	Membandingkan HTTP/2 SSE dan WebSocket dalam penerapan <i>Speech Recognition</i> pada ns-3	Simulasi	Ubuntu (HTTP/2 dan WebSocket)	Besar latensi aplikasi pada penggunaan HTTP/2 SSE serta WebSocket relatif sama serta WebSocket lebih rentan di- <i>block</i> oleh <i>proxy server</i> dibandingkan HTTP/2
3	Analisis Perbandingan Kinerja Protokol WebSocket dengan Protokol SSE pada Teknologi <i>Push Notification</i>	(Panser Brigade Muhammad, dkk. 2018)	Analisis Perbandingan Kinerja Protokol WebSocket dengan Protokol SSE pada Teknologi <i>Push Notification</i>	Purwarupa	Ubuntu – smartphone (WebSocket dan HTTP/1.1)	Rata-rata delay pada protokol SSE lebih kecil dibandingkan dengan WebSocket begitu juga dengan

						presentase penggunaan CPU
4	<i>Performance comparison of XHR polling, Long-Polling, Server-Sent Events and WebSockets</i>	(Oliver Örnmyr, dkk. 2017)	Membandingkan penggunaan memori dan CPU dari 100 perangkat virtual yang terhubung dengan server menggunakan <i>XHR Polling, Long-Polling , Server-Sent Events</i> dan <i>WebSockets</i>	Simulasi	Ubuntu (HTTP/1.1 dan Webscoket)	Penggunaan CPU, memori maupun bandwith pada SSE serta WebSocket lebih kecil dibandingkan dengan <i>XHR Polling</i> serta <i>Long-Polling</i>
5	<i>Mobile HTML5: Efficiency and Performance of WebSockets and Server-Sent Events</i>	(Elliot Estep, 2013)	Membandingkan performa browser ketika menggunakan WebSockets dan <i>Server-Sent Events</i> dalam berbagai jaringan <i>smartphone</i> (WiFi, 3G dan 4G)	Simulasi	Windows 7 (WebSocket dan HTTP/1.1)	Performa SSE maupun Websocket juga dipengaruhi oleh <i>web browser</i> serta konfigurasi jaringan yang digunakan

Penelitian yang dilakukan membandingkan *response time* serta persentase penggunaan CPU antara HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE dan WebSocket pada sistem kendali rumah pintar berbasis *web. Server* yang digunakan berupa Raspberry Pi 3 serta mampu diakses melalui internet.

## 2.10 Hipotesis

Berdasarkan kajian dari Tinjauan Pustaka, dapat dibuat hipotesis bahwa HTTP/2 *Server Sent Event* memiliki nilai *response time* terendah sedangkan HTTPS memiliki nilai *response time* tertinggi. Namun dari keempat metode yang digunakan, nilai penggunaan CPU keempatnya relatif sama.

## **BAB III**

### **METODE PENELITIAN**

Pada bab ini dijelaskan mengenai kebutuhan peralatan dan bahan serta perangkat lunak pendukung untuk melakukan pengembangan serta pengambilan data metode HTTP/1.1 *Server-Sent Events*, HTTP/2 *Server-Sent Events* dan *WebSocket* untuk *website* pada sistem rumah pintar. Selanjutnya dijelaskan juga mengenai metode penelitian dan skenario pengambilan data perbandingan.

#### **3.1 Peralatan**

Berikut merupakan daftar peralatan yang akan digunakan selama proses penelitian berlangsung :

1. Komputer sebagai *client* dengan spesifikasi:
  - CPU : Intel Celeron 1.50GHz x 4
  - Hardisk : 750 GB
  - RAM : 8 GB
  - OS : Linux Mint 18.2 Cinnamon 64-bit
2. Raspberry Pi 3 Model B sebagai *broker* sekaligus *web server* dan *server API* sebanyak 1 unit dengan spesifikasi:
  - CPU : 1.2 GHz quad-core ARM
  - Memory : 1 GB LPDDR2-900 SDRAM
  - USB Port : 4
  - Network : 10/100 Mbps Ethernet, 802.11 n Wireless LAN
3. NodeMCU 1 Unit dengan spesifikasi:
  - MCU : Xtensa Dual-Core 32-bit L106
  - Wifi : 802.11 b/g/n HT40
  - Typical Frequency: 160 MHz
  - Tipe ESP : ESP32

4. NodeMCU 1 Unit dengan spesifikasi:
  - MCU : Xtensa Single-Core 32-bit L106
  - Wifi : 802.11 b/g/n HT20
  - Typical Frequency: 80 MHz
  - Tipe ESP : ESP8266
5. *Access Point* sebanyak – 1 unit
6. Sensor DHT11 – 2 unit
7. LED Putih – 13 unit
8. Servo Motor SG90 – 2 unit
9. *Motor DC* 3 volt – 1 unit
10. *Cooling fan DC* 5 volt – 1 unit
11. Kabel Jumper

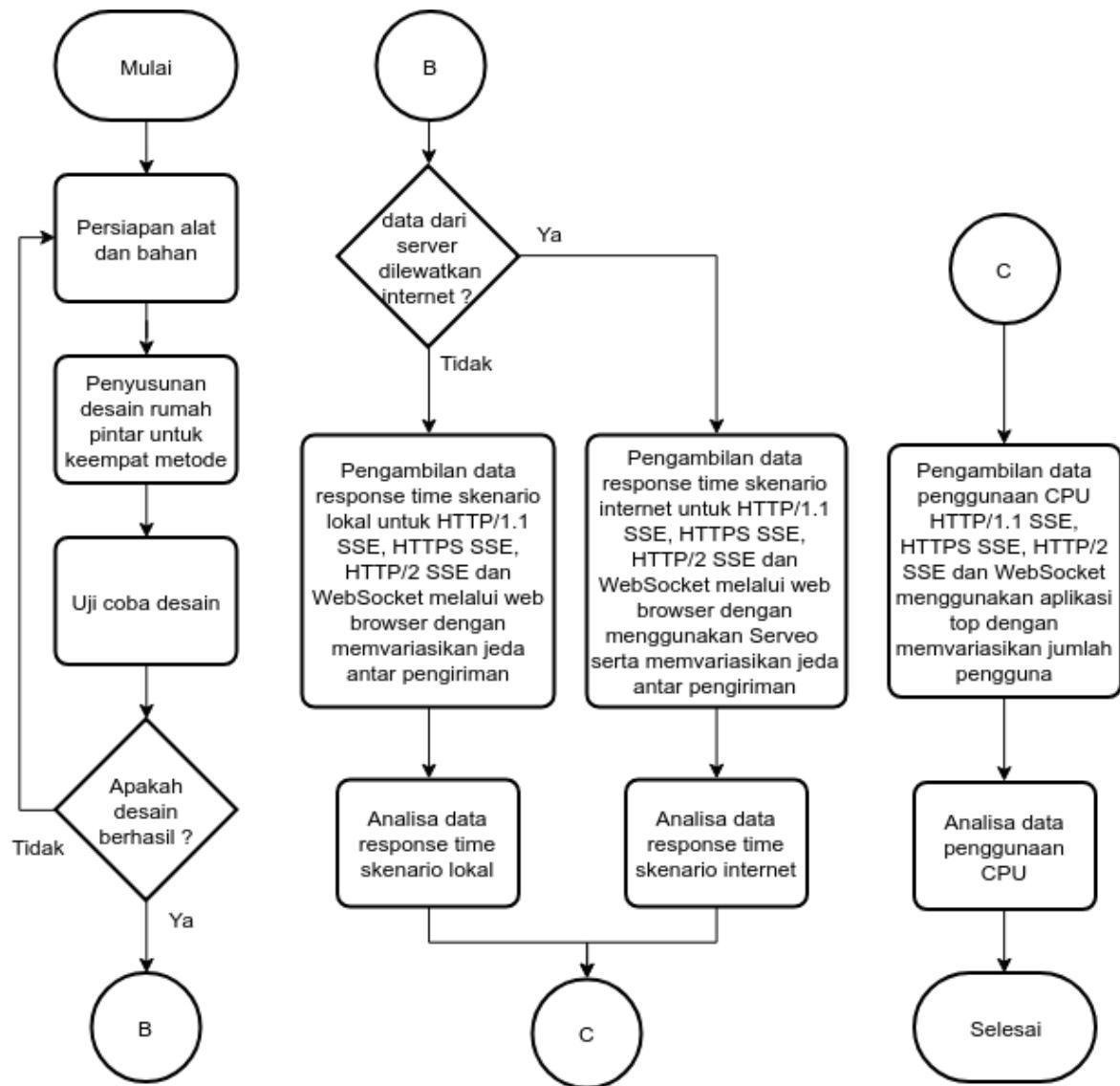
### 3.2 Baham

Berikut merupakan daftar perangkat lunak yang akan digunakan selama penelitian ini berlangsung :

1. Mosquitto, sebagai broker untuk protokol MQTT
2. Node.js, sebagai *web server* yang khusus Javascript
3. Vue.js, sebagai javascript framework untuk pembuatan *web application*
4. OpenSSL, untuk pembuatan *public key* dan *private key* pada HTTPS serta HTTP/2
5. Serveo, untuk menjadikan *local server* mampu diakses secara publik
6. Google Chrome, untuk mengakses *website*

### 3.3 Tahapan Penelitian

Metode penelitian yang dilakukan penulis dalam penelitian ini dapat dilihat pada *flow chart* pada Gambar 3.1.



Gambar 3.1 Diagram alir penelitian

### 1. Tahap Instalasi.

Tahap pertama yang dilakukan adalah menginstal semua perangkat lunak yang dibutuhkan. Adapun perangkat lunak yang *di-install* pada tahap ini adalah Mosquitto, Node.js, Vue.js dan OpenSSL yang akan dipasang pada Raspberry Pi.

### 2. Tahap Pengembangan Sistem.

Pada tahap pengembangan sistem, terdapat beberapa fase yang harus dilalui, yakni :

#### a. Perancangan Sistem

Fase perancangan aplikasi memiliki tujuan untuk memberikan gambaran umum mengenai bagaimana proses yang akan berjalan pada sistem yang akan diterapkan.

#### b. Pembuatan Sistem

Fase pembuatan sistem bertujuan untuk mengimplementasikan *WebSocket API* serta *Server-Sent Events API* baik pada sisi *client* maupun *server*. Selain itu, pada fase ini akan diterapkan pula penerapan MQTT di sisi mikrokontroler.

### 3. Tahap Pengambilan Data

Di dalam tahap ini akan dilakukan proses pengambilan data untuk metode HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE serta Websocket yang telah diterapkan di dalam tahap pengembangan sistem. Parameter yang akan digunakan untuk pengambilan data adalah presentase penggunaan CPU serta *response time* untuk setiap metode yang digunakan. Pengambilan data *response time* dilakukan dengan menghitung waktu antara pengubahan status perangkat yang dilakukan dari *web browser* dengan memberikan variasi jeda antar pengiriman.

#### 4. Tahap Analisis.

Analisis merupakan tahapan paling akhir dalam penelitian ini. Pada tahap ini hasil pengambilan data yang telah dilakukan pada tahap sebelumnya akan dikumpulkan, diurai, dibedakan dan dipilah untuk selanjutnya digolongkan dan dikelompokkan berdasarkan kriteria tertentu kemudian dibuat suatu kesimpulan dari hasil yang didapat.

### **3.4 Perancangan Topologi dan Model**

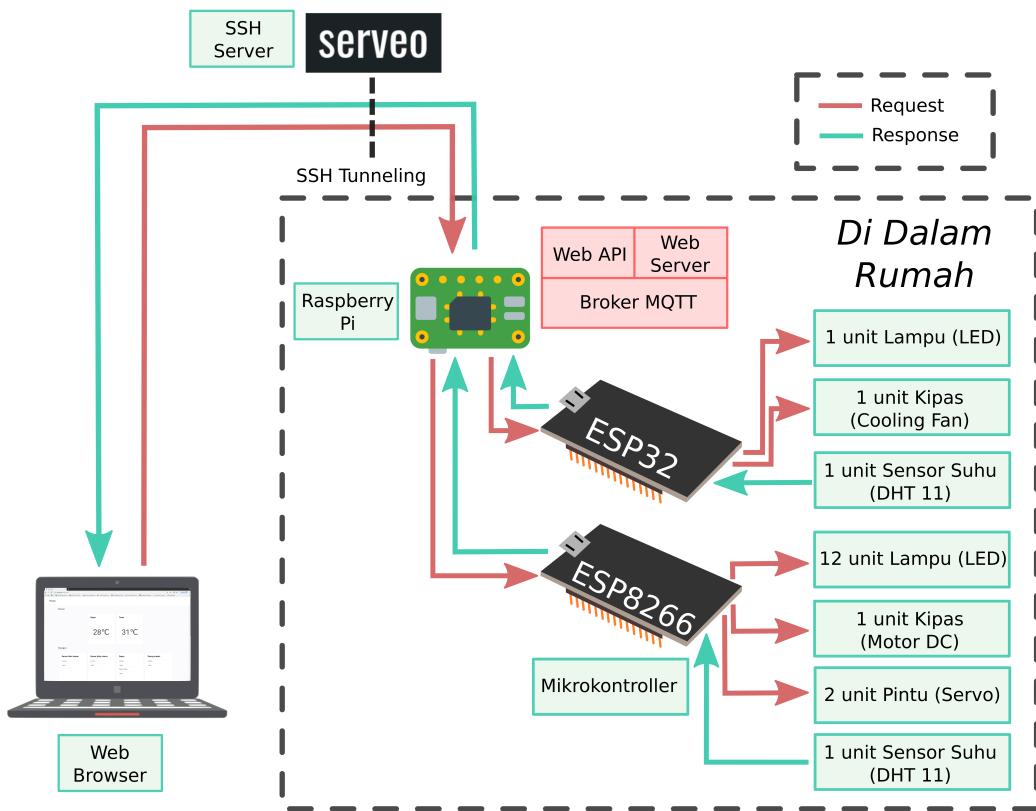
Dalam tahap pengembangan aplikasi, dilakukan dua tahap perancangan, yakni perancangan topologi dan perancangan model.

#### **3.4.1. Perancangan Topologi**

Perancangan topologi dibuat dengan memperhatikan gambaran umum proses yang terjadi pada sistem beserta hubungan antar komponen di dalamnya. Perancangan topologi dapat dibagi menjadi perancangan topologi perangkat serta perancangan topologi data. Perancangan topologi perangkat berguna untuk mendeskripsikan alur kerja sistem berdasarkan perangkat yang digunakan maupun dilalui sedangkan perancangan topologi data untuk menggambarkan lalu lintas data yang terjadi pada sistem.

##### **1. Perancangan Topologi Perangkat**

Berdasarkan perangkat yang digunakan, terdapat tiga komponen utama yang dibutuhkan supaya sistem mampu berjalan yakni *web browser*, *server* serta mikrokontroler. Untuk lebih lengkapnya dapat diamati pada Gambar 3.2.



Gambar 3.2 Bagan Topologi Perangkat

*Web browser* merupakan komponen yang berperan sebagai antarmuka pada sistem yang akan dibuat. Data berupa HTML, CSS maupun Javascript yang akan ditampilkan pada *web browser* berasal dari *web server*. Untuk data yang berupa JSON maupun teks seperti halnya suhu berasal dari *web API*.

Komponen *server* dapat dibagi menjadi tiga bagian, yakni *web server*, *web API*, serta *MQTT Broker*. *Web server* berperan untuk mengirimkan data *web application* yang berupa HTML, CSS maupun javascript. *Web application* selanjutnya akan ditampilkan di *web browser* penghuni rumah. Untuk pembuatan *web application* dapat dilakukan dengan

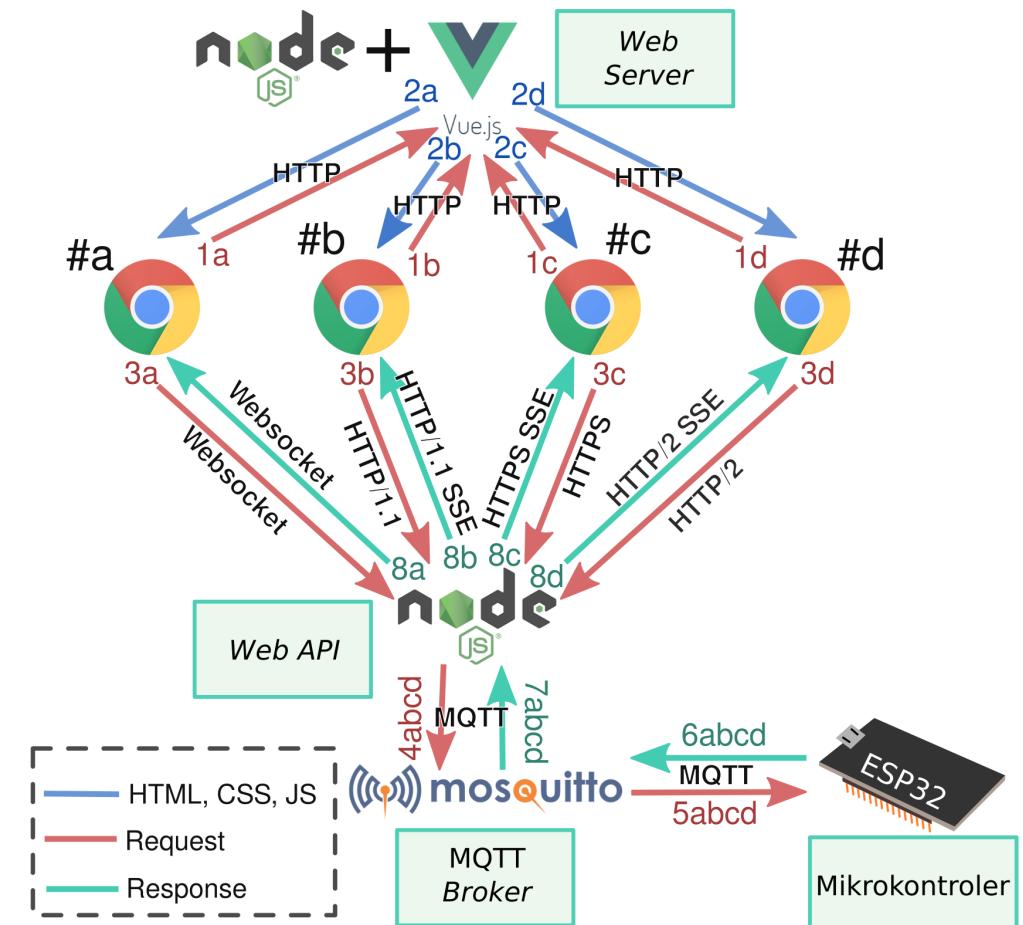
berbagai metode. Dalam penelitian ini, *web application* dibuat menggunakan *javascript framework* yakni “Vue.js”. Selanjutnya, terdapat bagian *web API* yang bertugas untuk mengolah data yang berasal maupun menuju MQTT *Broker*. Baik web API maupun web server, keduanya bekerja dengan menggunakan perangkat lunak Node.js. Kemudian data yang telah masuk ke MQTT *Broker* akan diteruskan menuju mikrokontroler .

Dalam penerapannya, *server* tanpa menggunakan Serveo hanya dapat diakses pada jaringan lokal. Serveo adalah *server* SSH yang digunakan untuk *tunneling* dari suatu komputer sehingga mampu untuk diakses dari luar jaringan. Terdapat pula pihak ketiga yang serupa dengan Serveo, yakni Ngrok. Pemilihan Serveo daripada Ngrok didasarkan pada biaya yang dikeluarkan. Ngrok membutuhkan biaya tambahan ketika melewatkkan data melalui HTTPS, padahal HTTPS diperlukan ketika menggunakan HTTP/2.

Komponen utama terakhir yang dibutuhkan adalah mikrokontroler. Setiap mikrokontroler membutuhkan perangkat yang mampu untuk menghubungkan mikrokontroler dengan jaringan yang akan digunakan, sehingga dalam penelitian ini digunakanl ESP32 serta ESP8266 sebagai pengirim sekaligus penerima data dari MQTT *Broker*. Selain itu, terdapat pula DHT11 yang digunakan untuk mengukur suhu serta komponen lainnya seperti LED, *motor* maupun kipas yang dapat dikendalikan oleh mikrokontroler.

## 2. Perancangan Topologi Data

Topologi data berguna untuk menjelaskan alur pengiriman data yang berlangsung selama suatu metode pengiriman digunakan. Untuk lebih detailnya, bagan perancangan topologi data dapat diamati pada Gambar 3.3.



Gambar 3.3 Bagan Topologi Data

Pada sistem ini digunakan empat metode pengiriman untuk komunikasi antara *web browser* dengan *web API*. Hal ini ditunjukkan oleh Gambar 3.3 dengan notasi a, b, c dan d yang mewakili setiap skenario pengiriman data. Notasi a mewakili skenario program

pengirim melalui WebSocket, notasi b untuk HTTP/1.1 SSE, notasi c untuk HTTPS sedangkan notasi d untuk HTTP/2 SSE. Untuk mempermudah penyebutan keempat skenario digunakan notasi abcd.

Secara keseluruhan terdapat tiga proses utama yang berlangsung selama sistem berjalan yakni pemuatan *web application* di *web browser* (**Notasi 1a/1b/1c/1d menuju 2a/2b/2c/2d**), pengiriman perintah dari *web browser* menuju mikrokontroler (**Notasi 3a/3b/3c/3d menuju 8a/8b/8c/8d**) serta pengiriman data suhu dari mikrokontroler menuju *web browser* (**Notasi 6abcd menuju 8a/8b/8c/8d**). Dari ketiga proses tersebut, pemuatan *web application* merupakan proses yang pertama kali berjalan ketika penghuni rumah berinteraksi dengan sistem.

Proses yang pertama diawali setelah penghuni rumah memasukkan alamat *web server* ke dalam kolom alamat *website* di suatu *web browser* guna meminta pemuatan *web application* (**Notasi 1a/1b/1c/1d**). *Web server* yang berjalan menggunakan bantuan Node.js akan membalas permintaan tersebut dengan mengirimkan dokumen HTML (**Notasi 2a/2b/2c/2d**). Dokumen tersebut selanjutnya akan disusun oleh *web browser* untuk ditampilkan. Apabila dalam penyusunan tersebut dibutuhkan dokumen tambahan seperti CSS, Javascript maupun tipe dokumen lainnya, maka *web browser* akan meminta dokumen tersebut menuju alamat *website* yang telah dicantumkan di dokumen HTML. Alamat *website* tidak harus menggunakan alamat *web server* yang menyediakan *web application*, seringkali beberapa dokumen maupun data dibutuhkan dari alamat lainnya semisal dari

*web API* (**Notasi 3a/3b/3c/3d**). Hal inilah yang terjadi pada proses pengiriman perintah maupun permintaan data suhu oleh *web browser*.

Dalam penelitian ini, data jumlah piranti elektronik yang terhubung dengan mikrokontroler yang digunakan tidaklah tercantum di halaman HTML. Data yang tercantum di halaman tersebut hanyalah data jumlah mikrokontroler serta jumlah sensor yang digunakan. Untuk mendapatkan data jumlah piranti elektronik yang terhubung dengan mikrokontroler beserta statusnya perlu dilakukan proses pengiriman perintah dari *web browser*. Hal ini dikarenakan data status piranti elektronik yang terhubung dengan mikrokontroler hanya didapatkan setelah perintah diterima oleh mikrokontroler (**Notasi 3a/3b/3c/3d menuju 8a/8b/8c/8d** dikerjakan). Untuk menanggulangi hal tersebut dikirimkanlah perintah kosong menuju mikrokontroler setiap kali *web application* dimuat.

Untuk memenuhi kondisi tujuan penelitian ini, proses pengiriman perintah dapat dilakukan melalui berbagai protokol sesuai dengan URL yang digunakan pada kolom alamat *website*. Protokol yang dicantumkan untuk HTTPS dan HTTP/2 adalah ‘https’, untuk HTTP/1.1 adalah ‘http’ sedangkan untuk WebSocket adalah ‘ws’. Walaupun melalui berbagai metode pengiriman data yang berbeda, pada dasarnya perintah yang dikirim berisikan data ruangan, piranti elektronik serta status piranti elektronik yang diinginkan oleh penghuni rumah. Selanjutnya, perintah yang telah diterima oleh *web API* akan diteruskan menuju MQTT *Broker* (**Notasi 4abcd**) menggunakan struktur data yang lebih sederhana daripada struktur data yang diterima oleh *web API* (JSON). Struktur data ini dibuat supaya memudahkan pengembangan dari sisi mikrokontroler.

Setelah data diterima oleh mikrokontroler (**Notasi 5abcd**), data tersebut akan digunakan untuk menentukan perangkat mana yang statusnya akan diubah. Data status piranti elektronik tersebut kemudian disimpan di mikrokontroler untuk dikirimkan kembali setelah status piranti elektronik berhasil diubah sesuai keinginan penghuni rumah. Data yang dikirimkan menuju *web API* (**Notasi 6abcd**) tidak hanya data piranti elektronik yang berhasil diubah saja melainkan data status seluruh piranti elektronik yang berada pada mikrokontroler tersebut. Selanjutnya, data tersebut dikirimkan menuju *web API* menggunakan format penyusunan data yang berbeda dengan struktur data yang digunakan untuk menerima data dari *web API*. Struktur dibuat dengan sesederhana mungkin untuk menanggulangi keterbatasan jumlah karakter yang mampu dikirimkan dari mikrokontroler. Setelah data tersebut diterima oleh *web API* (**Notasi 7abcd**), data tersebut akan diteruskan menuju *web browser* menggunakan struktur data JSON (**Notasi 8a/8b/8c/8d**). Terakhir, data tersebut kemudian diolah oleh *web browser* sehingga mampu untuk dilihat serta digunakan kembali oleh penghuni rumah.

Berbeda dengan proses pengiriman perintah, proses pengiriman suhu dilakukan dari mikrokontroler menuju *web browser* tanpa membutuhkan permintaan data terlebih dahulu (**Notasi 6abcd menuju 8a/8b/8c/8d**). Hal ini dapat terjadi dikarenakan keempat metode mampu membuka koneksi dengan rentang waktu yang cukup lama. Proses pembacaan sensor suhu yang dilakukan oleh sensor DHT11 dilakukan setiap beberapa detik. Setelah data suhu didapatkan, data suhu kemudian dikirim menuju *web API* untuk diolah (**Notasi 6abcd**) dan dikirimkan menuju *web browser* (**Notasi 8abcd**).

Dari ketiga proses tersebut, *web API* memiliki peran yang penting untuk mengolah data terutama data yang diterima dari mikrokontroler. Data yang berasal dari mikrokontroler seperti halnya suhu maupun status piranti elektronik perlu dikirimkan secara langsung dari *web API* menuju *web browser* melalui koneksi TCP yang telah terbuka (**Notasi 6abcd menuju 8a/8b/8c/8d**). Untuk memisahkan permintaan data suhu dan data status piranti elektronik (**Notasi 3a/3b/3c/3d**) agar tidak saling bercampur digunakanlah *path* tertentu pada URL yang digunakan.

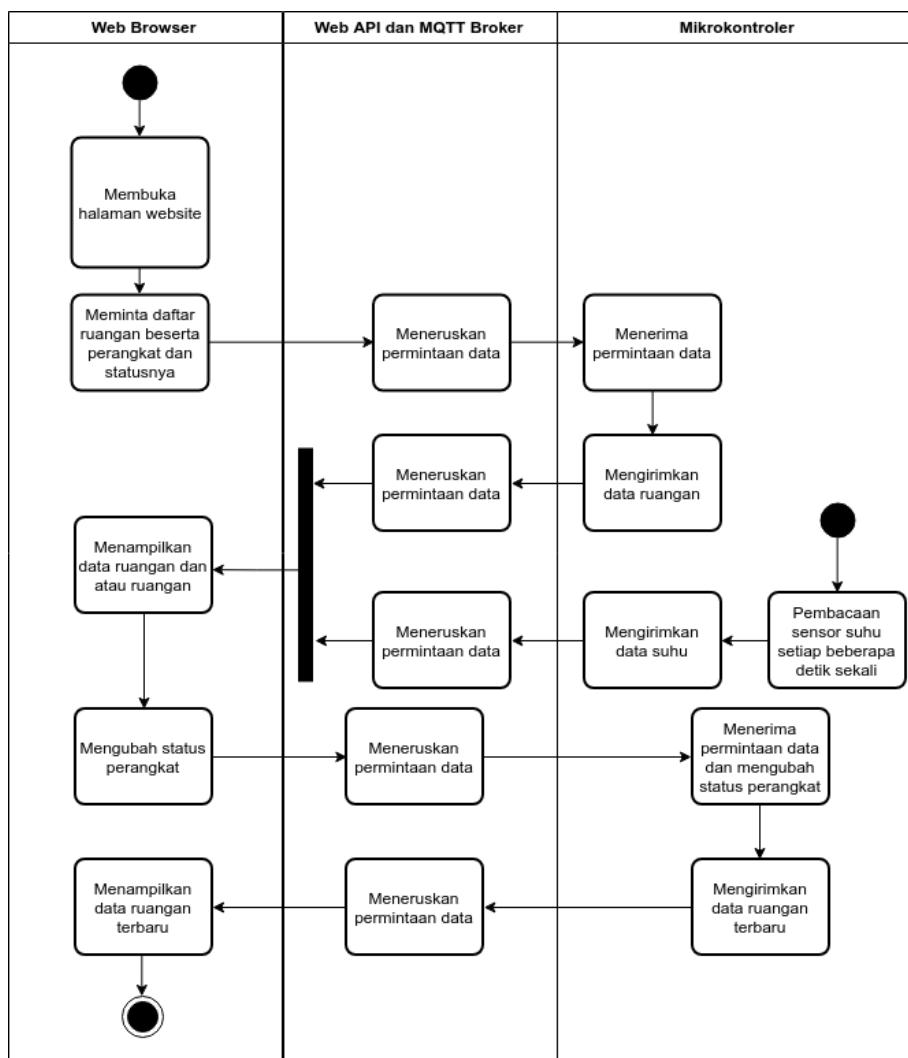
Jumlah *path* yang dapat digunakan sesuai dengan jumlah koneksi terbuka yang dibutuhkan. Koneksi terbuka yang dimaksud adalah koneksi yang memungkinkan *web API* untuk mengirimkan data menuju *web browser* selama hubungan antar keduanya masih terjalin. Pada penelitian ini dibutuhkan tiga koneksi terbuka yakni untuk melewatkkan data status piranti elektronik, data sensor suhu dari ESP32 serta data sensor suhu dari ESP 8266. Untuk melakukan permintaan data status piranti elektronik yang terhubung pada seluruh mikrokontroler *path* yang digunakan adalah ‘/0’, untuk permintaan data sensor suhu pada ESP32 adalah ‘/1’ sedangkan untuk permintaan data sensor pada ESP8266 adalah ‘/2’.

### 3.4.2. Perancangan Model

Perancangan model dibuat untuk menggambarkan alur pemakaian sistem oleh penghuni rumah. Dalam pembuatannya, perancangan model dibuat menggunakan *Unified Modeling Language* (UML) Diagram yang mana memudahkan pengembang maupun orang lain untuk menyampaikan alur kerja suatu sistem yang telah dibuat. UML Diagram yang digunakan dalam perancangan model adalah *use case diagram* dan *activity diagram*.

a. *Activity Diagram*

*Activity Diagram* digunakan untuk menggambarkan aliran kerja atau langkah-langkah yang ditempuh selama sistem berjalan. Dalam perancangan *activity diagram*, keseluruhan *activity* yang terdapat di dalam sistem ditampilkan sesuai dengan tiga komponen utama yang terdapat pada aplikasi. Pada Gambar 3.3 terlihat *activity diagram* dari sistem yang akan dibuat.

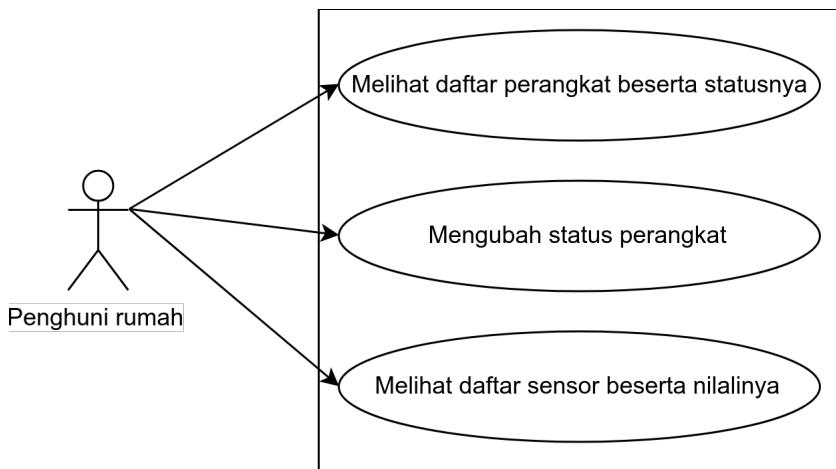


Gambar 3.3 *Activity Diagram* sistem

b. *Use Case Diagram*

*Use Case Diagram* adalah diagram UML yang ditujukan untuk menggambarkan skenario fungsi dari sistem yang dikembangkan. Tujuan utama penggunaan *use case diagram* adalah untuk membuat visualisasi dari fungsi yang dibuat dalam suatu sistem.

Gambar 3.4 merupakan *use case diagram* dari *web server* yang akan dibuat.



**Gambar 3.4 Use case diagram web server**

Untuk tampilan *web application* yang digunakan pada sistem kendali rumah pintar ini dapat dilihat pada Lampiran 4.

### 3.5 Pembuatan *Web API* serta *Web Server*

Proses pengiriman data menggunakan metode *WebSocket* berbeda dengan ketika menggunakan *Server-Sent Events* baik di sisi *server* maupun di sisi *web browser*. Hal ini dikarenakan *WebSocket* mampu untuk melakukan pengiriman data dari dua arah (*bidirectional*), namun tidak dengan *Server-Sent Events*. Selain itu untuk mengirim data

menggunakan protokol *WebSocket* di *Javascript* diperlukan *WebSocket API* sedangkan untuk metode *Server-Sent Events* membutuhkan *EventSource API*.

### **3.5.1 Penerapan Node.js serta Vue.js**

Sebelum menerapkan *WebSocket* maupun *Server-Sent Events* di sisi server, perlulah dibuat proyek Node.js baru. Node.js sebagai *Javascript Framework* diperlukan untuk mempermudah pembuatan *Web API* maupun *web server*. Proses instalasi Node.js dapat dilihat selengkapnya pada Lampiran 1. Untuk membuat proyek baru yang akan digunakan sebagai *Web API* dimasukkan perintah :

```
$ mkdir webapi && cd webapi
$ touch index.js
$ npm init
```

Setelah itu, masukkan nama beserta *metadata* npm lainnya. *Metadata* npm masih dapat diubah setelah proyek terbentuk dengan cara mengubah lalu konten yang ada pada “*package.json*”.

Setelah *Web API* selesai dibuat, maka proyek baru yang akan digunakan sebagai *web server* dapat dibuat dengan menggunakan perintah :

```
$ vue create webapp
```

Kemudian pilihlah pilihan “*Manually select features*” dan dilanjutkan dengan memilih fitur Babel dan Router. Babel merupakan *javascript compiler* yang memungkinkan berbagai *web browser* untuk menggunakan ECMAScript2015 keatas, mengubah ekstensi vue

maupun jsx sehingga mampu untuk dibaca oleh web browser serta melakukan *polyfill*.

Untuk fitur Router berguna untuk memudahkan pemetaan alamat (URL) pada “Vue.js”.

### **3.5.2 Penerapan *WebSocket* dan *Server-Sent Events***

Untuk menggunakan *WebSocket API* di *web browser* dibutuhkanlah *server* yang mampu digunakan sebagai *WebSocket server*. Node.js sebagai *web API* telah mampu untuk dijadikan *WebSocket server* menggunakan *module (library)* tambahan yang bernama ‘*WebSocket*’. Module ini dapat diunduh melalui npm setelah Web API dibuat dengan memasukkan perintah :

```
$ cd webapi  
$ npm install WebSocket
```

Penggunaan *module* ‘*WebSocket*’ dapat dilihat pada dokumentasinya ataupun pada halaman Lampiran 2. Berbeda dengan di *Web API*, penerapan *WebSocket* di *web server* dapat menggunakan *WebSocket API* yang telah disediakan oleh HTML5, sehingga tidak diperlukan *module* tambahan.

Sama halnya dengan *WebSocket*, dalam penerapan *Server-Sent Events* pada sisi *server* dibutuhkan *module* tambahan yakni ‘*express*’ dan ‘*http2*’. *Module* ‘*express*’ berguna untuk mempermudah pembuatan *REST API* melalui HTTP/1.1 di dalam Node.js, sedangkan *module* ‘*http2*’ berguna untuk pembuatan *REST API* melalui HTTP/2. Untuk penerapan *Server-Sent Events* di dalam *web server* menggunakan *EventSource API* yang telah dimiliki oleh sebagian besar *web browser*.

### 3.5.3 Penerapan TLS

*Transport Layer Security* (TLS) merupakan protokol kriptografi yang berfungsi untuk mengamankan data. Di dalam *OSI Layer*, TLS berada di *layer session*. TLS merupakan contoh dari penerapan kriptografi asimetris. Kriptografi asimetris adalah sistem kriptografi yang menggunakan sepasang kunci, *public key* dan *private key*. Dalam penerapannya pada *web application*, *public key* dapat dimiliki oleh siapa saja yang mengakses halaman *web application* sedangkan *private key* hanya dimiliki oleh pemilik *web application*. Public key berguna untuk mengenkripsi data – data yang dimiliki oleh pengakses halaman agar hanya bisa dibaca oleh pemilik *private key*.

Tanpa menggunakan TLS, HTTP/2 dan HTTPS tidak dapat digunakan. TLS dapat dibuat menggunakan bantuan OpenSSL. Untuk meng-generate *self-signed certificate* di dalam sistem operasi Linux Mint, dimasukkan perintah sebagai berikut :

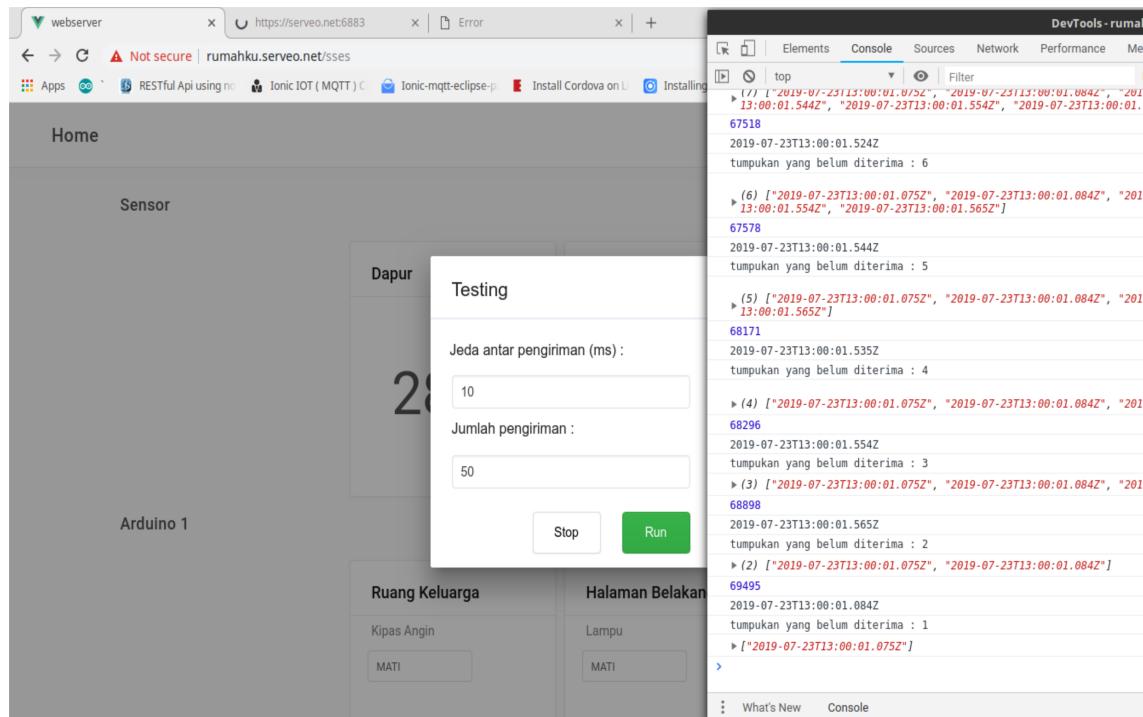
```
$ openssl req -x509 -newkey rsa:4096 -keyout  
secret.key -out secret.crt -days 365
```

## 3.6 Metode Pengambilan Data

Tujuan dari penelitian ini dilakukan adalah untuk membandingkan beberapa metode pengiriman data di dalam kasus rumah pintar. Metode pengiriman data yang dipilih adalah HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE serta WebSocket. Keempat metode dibandingkan berdasarkan nilai *response time* serta presentase penggunaan CPU yang didapatkan setelah proses pengambilan data dilakukan.

### 3.6.1 Response Time

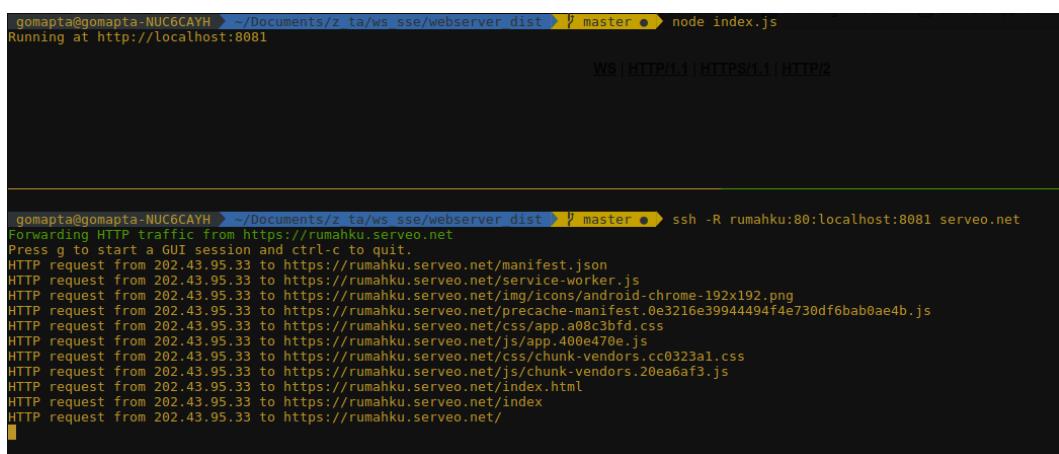
Pengambilan data *response time* untuk keempat metode dilakukan dengan cara menghitung selisih waktu dari lima puluh perintah yang dikirimkan dari *web browser* sampai mendapatkan balasan dari mikrokontroler. Proses ini ditunjukkan dengan notasi 3a/3b/3c/3d menuju 8a/8b/8c/8d pada Gambar 3.3. Kelimapuluhan perintah dikirimkan dengan jeda antar waktu pengiriman yang bervariasi serta memiliki ukuran paket yang sama. Perhitungan waktu dilakukan dengan menandai waktu awal pengiriman perintah serta waktu akhir ketika pesan balasan diterima dari sisi *web browser*. Hasil perhitungan akan ditampilkan ke dalam *console* pada suatu *web browser* seperti yang terlihat pada Gambar 3.5.



Gambar 3.5 Tampilan pengujian response time melalui Serveo

Pengambilan *response time* dilakukan dengan menggunakan salah satu fitur yang telah dimiliki oleh *web application*. Pada fitur tersebut dapat diatur jeda antar pengiriman serta jumlah data yang akan dikirimkan. Setiap kali data pengujian diterima oleh *web browser* maka pada *console* akan ditampilkan hasil *response time*, data yang diterima serta tumpukan data yang belum diterima seperti yang berada di dalam persegi panjang hijau pada Gambar 3.5.

Proses pengambilan *response time* dilakukan dalam dua skenario yang berbeda, yakni di dalam jaringan lokal yang sama serta melalui jaringan internet. Untuk skenario pengujian melalui jaringan internet dibutuhkan aplikasi pihak ketiga, yakni Serveo. Serveo adalah aplikasi yang berguna untuk melakukan *SSH Tunneling*, yang mana memungkinkan seseorang untuk mengakses komputer yang berbeda jaringan melalui port tertentu. Untuk penggunaan Serveo dapat dilihat pada Gambar 3.6



```
gomapta@gomapta-NUC6CAYH:~/Documents/z_tasw_sse/webserver_dist$ ./master node index.js
Running at http://localhost:8081
WS | HTTP/1.1 | HTTPS/1.1 | HTTP/2

gomapta@gomapta-NUC6CAYH:~/Documents/z_tasw_sse/webserver_dist$ ./master ssh -R rumahku:80:localhost:8081 serveo.net
Forwarding HTTP traffic from https://rumahku.serveo.net
Press g to start a GUI session and ctrl-c to quit.
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/manifest.json
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/service-worker.js
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/img/icons/android-chrome-192x192.png
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/precache-manifest.0e3216e39944494f4e730df6bab0ae4b.js
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/css/app.a08c3bfd.css
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/js/app.400e470e.js
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/css/chunk-vendors.cc0323a1.css
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/js/chunk-vendors.20ea6af3.js
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/index.html
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/index
HTTP request from 202.43.95.33 to https://rumahku.serveo.net/
```

**Gambar 3.6 Pengaktifan Serveo melalui Raspberry Pi**

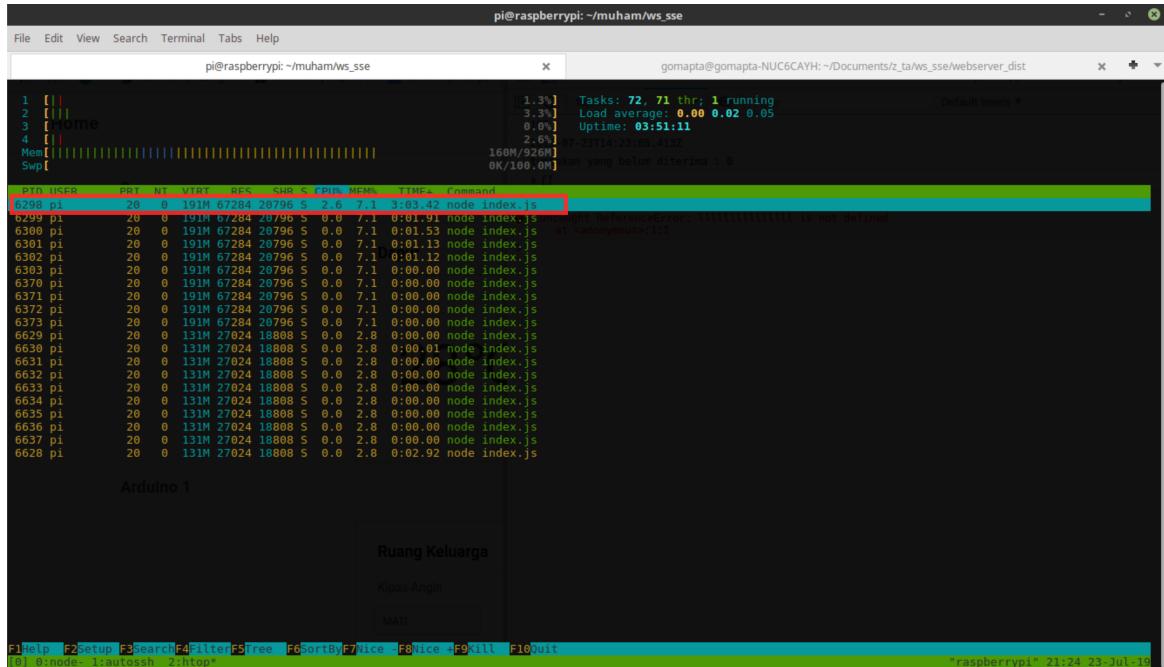
Untuk menggunakan Serveo dibutuhkan SSH Client. Pengaktifan Serveo dapat dilakukan dengan menjalankan perintah berikut di dalam terminal Linux.

```
$ ssh -R rumahku:80:localhost:8081 serveo.net
```

Setelah perintah dijalankan maka hasil dapat dilihat melalui *web browser* dengan memasukkan URL “<http://rumahku.serveo.net>” dan didapatkan hasil seperti yang terlihat pada Gambar 3.5.

### 3.6.2 Penggunaan CPU

Pengambilan data presentase penggunaan CPU dilakukan dengan menggunakan bantuan aplikasi ‘htop’ dan ‘top’ yang terdapat di dalam Raspberry Pi. Aplikasi ‘htop’ digunakan untuk mengecek aplikasi node mana yang digunakan oleh *web API* dengan cara memilih aplikasi yang hanya bernama “node”. Proses pemilihan dapat dilihat pada Gambar 3.7.



Gambar 3.7 Pemilihan aplikasi node di dalam htop

Setelah didapatkan hasil seperti gambar di atas, dipilihlah node dengan nilai presentase penggunaan CPU yang memiliki nilai berganti-ganti ketika dilakukan proses pengiriman data dari *web API*. Dari node tersebut diambil nilai pid semisal ‘6298’ dan kemudian nilai pid tersebut dimasukkan ke dalam perintah berikut.

```
$ top | grep 6298
```

Hasil yang akan ditampilkan setelah perintah dijalankan dapat dilihat pada Gambar 3.8.

```
pi@raspberrypi:~/muham/ws_sse/webserver/dist/js $ top | grep 6298
6298 pi      20   0 196100  68072  20796 S  1,3  7,2  3:04.75 node
6298 pi      20   0 196100  68072  20796 S  1,3  7,2  3:04.79 node
6298 pi      20   0 196100  68336  20796 S  1,3  7,2  3:04.83 node
6298 pi      20   0 196100  67516  20796 S  2,0  7,1  3:04.89 node
6298 pi      20   0 196100  67516  20796 S  1,6  7,1  3:04.94 node
6298 pi      20   0 196100  67776  20796 S  1,3  7,1  3:04.98 node
6298 pi      20   0 196100  67776  20796 S  1,3  7,1  3:05.02 node
```

**Gambar 3.8 Hasil perintah top dengan pemilihan pid**

Dikarenakan dengan menggunakan ‘grep’ data yang muncul hanya baris yang terdapat di dalamnya teks ‘6298’, maka untuk mengetahui *header table* presentase penggunaan CPU dapat dijalankan aplikasi ‘top’ secara normal.

```
pi@raspberrypi:~/muham/ws_sse/webserver/dist/js $ top
top - 21:25:46 up  3:52, 10 users,  load average: 0,00, 0,01, 0,04
Tasks: 148 total,   1 running,  99 sleeping,   0 stopped,   1 zombie
%Cpu(s): 4,0 us, 0,3 sy, 0,0 ni, 95,7 id, 0,1 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 948304 total, 353812 free, 151520 used, 442972 buff/cache
KiB Swap: 102396 total, 102396 free,     0 used. 727376 avail Mem
          PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
          6666 pi      20   0   8684   3232   2736 R 10.5  0,3  0:00.08 top
```

**Gambar 3.9 Hasil perintah top secara normal**

Skenario pengambilan data presentase penggunaan CPU dilakukan dengan cara mengirimkan perintah setiap detiknya dari beberapa *web browser* menuju mikrokontroler secara bersamaan. Setiap *web browser* diibaratkan sebagai satu penghuni rumah. Setiap perintah yang dikirimkan dari *web browser* akan dibalas oleh mikrokontroler. Perintah

yang dikirimkan dari *web browser* memiliki ukuran paket yang sama, begitu juga dengan perintah yang dikirimkan dari mikrokontroler. Selain menerima balasan dari mikrokontroler, *web browser* juga menerima dua data suhu yang dikirimkan melalui *stream* yang berbeda antar keduanya maupun dengan *stream* yang digunakan untuk menerima pesan dari mikrokontroler. Untuk jumlah *web browser* yang digunakan akan bervariasi selama proses pengambilan data.

Dua *web browser* yang sama ketika dibuka pada waktu yang bersamaan akan menggunakan *session*, *history*, maupun *cache* yang sama. Hal ini tidak mengganggu proses pengambilan data WebSocket maupun HTTP/2 SSE yang memiliki karakteristik paralel, namun tidak dengan HTTP/1.1 SSE maupun HTTPS SSE yang hanya mampu membuka enam TCP *Connection*. Untuk mengatasi hal tersebut dibuat *session* yang berbeda untuk setiap membuka *web browser* dengan menjalankan *script* berikut di terminal Linux.

```
#!/bin/bash

RND_DIR="chromee-$RANDOM"

echo $RND_DIR

cd /tmp

mkdir $RND_DIR

google-chrome --user-data-dir=/tmp/$RND_DIR

--incognito

rm -R $RND_DIR
```

## **BAB IV**

### **HASIL PENELITIAN DAN PEMBAHASAN**

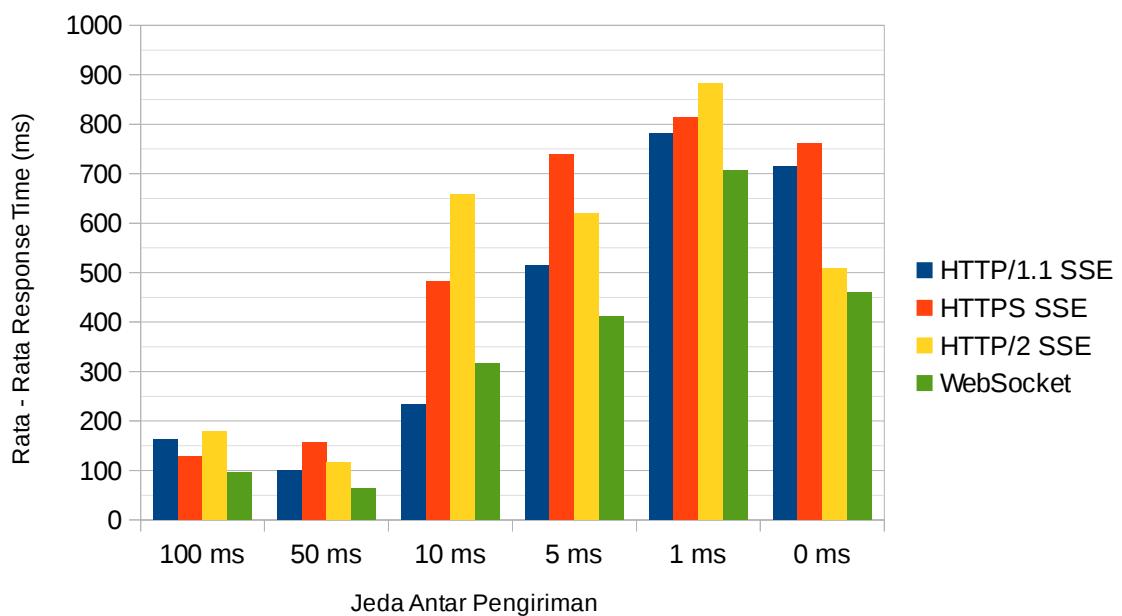
Pada bab ini akan menjelaskan mengenai hasil perbandingan antara HTTP/1.1 *Server-Sent Events*, HTTP/2 *Server-Sent Events*, HTTPS *Server-Sent Events* serta dengan WebSocket sebagai program pengirim yang digunakan pada sistem kendali rumah pintar. Proses pengambilan data dari keempat program pengirim yang digunakan pada penelitian ini menggunakan jaringan seluler pada waktu siang hari. Untuk parameter yang dibandingkan berupa *response time* serta presentase penggunaan CPU.

#### **4.1 Hasil Perbandingan *Response Time***

*Response time* merupakan lama waktu yang dibutuhkan oleh *web browser* untuk mendapatkan balasan dari mikrokontroler setelah *web browser* melakukan pengiriman perintah menuju mikrokontroler. Hal ini ditunjukkan Gambar 3.3 dengan notasi dari 3a/3b/3c/3d menuju 8a/8b/8c/8d. Data yang didapatkan oleh *web browser* berupa status piranti elektronik yang terhubung dengan mikrokontroler. Dalam perjalannya, data – data tersebut melewati MQTT *Broker* serta *web API*. *Web API* tidak harus berada di dalam jaringan lokal yang sama sengan *web browser*. Dengan bertambahnya nilai *response time*, tingkat variasi data yang didapatkan juga semakin beragam. Oleh karena itu, dibuatlah dua skenario pengambilan data *response time* yakni ketika *web browser* berada di dalam jaringan lokal yang sama dengan *web API* maupun ketika *web browser* terhubung dengan *web API* melalui jaringan internet.

#### 4.1.1 Skenario Jaringan Lokal

Pada skenario ini dilakukan pengiriman lima puluh pesan dari *web browser* menuju mikrokontroler dengan kondisi *web browser* berada di dalam jaringan lokal yang sama dengan *web API*. Hasil yang didapatkan dari pengiriman kelimpuluh pesan telah dirangkum pada Gambar 4.1.



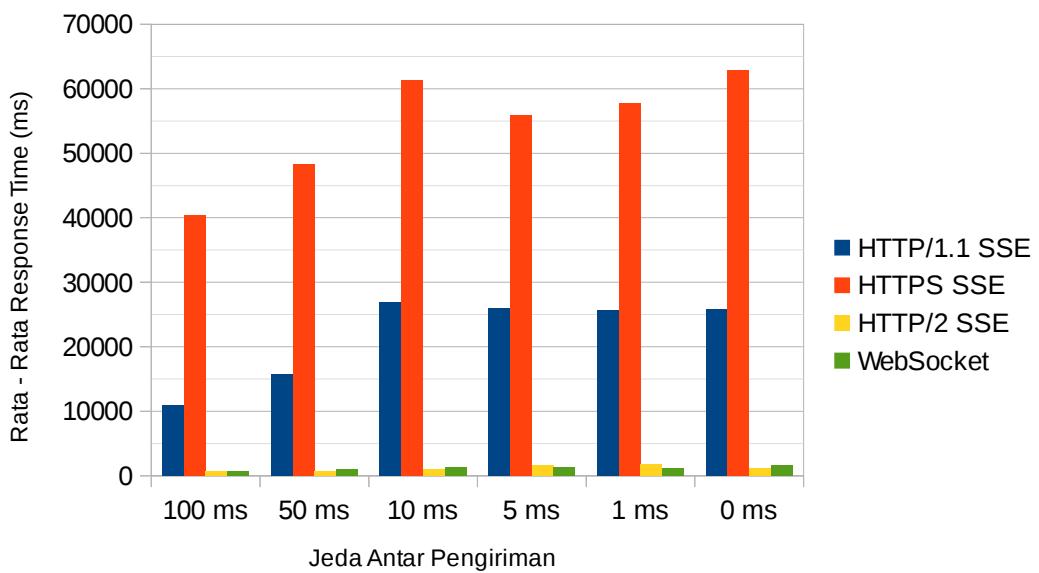
**Gambar 4.1 Grafik rata - rata *response time* pada skenario jaringan lokal**

Berdasarkan hasil yang didapatkan pada percobaan ini dapat dinyatakan bahwa WebSocket memiliki nilai rata – rata *response time* terkecil. Selain itu, jeda antar pengiriman juga memengaruhi nilai rata – rata *response time* untuk setiap metode pengiriman. Semakin kecil nilai jeda antar pengiriman, semakin besar pula nilai rata – rata *response time* yang dibutuhkan. Untuk lebih detailnya, dapat dilihat pada Tabel 4.1

**Tabel 4.1 Rata - rata response time pada skenario lokal**

Metode Pengiriman	Response Time (ms) dengan jeda antar pengiriman						Nilai Rata - rata (ms)
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms	
HTTP/1 SSE	162.7	100.0	233.4	515.3	781.2	714.0	417.8
HTTP/2 SSE	179.1	116.9	658.9	620.1	882.6	509.5	494.5
HTTPS SSE	128.7	155.8	482.2	739.8	814.2	761.3	513.7
WebSocket	95.3	63.2	317.0	411.5	706.8	459.8	342.3

#### 4.1.2 Skenario Jaringan Internet

**Gambar 4.2 Grafik rata - rata response time pada skenario jaringan internet**

Proses pengambilan data pada skenario ini menggunakan cara yang sama dengan skenario jaringan lokal, namun dengan kondisi *web browser* terhubung dengan *web API*

melalui jaringan internet dengan bantuan *tunneling* oleh Serveo. Hasil yang didapatkan dari percobaan ini telah dirangkum pada Gambar 4.2. Variasi data yang didapatkan ketika melalui jaringan internet jauh berbeda dibandingkan dengan skenario jaringan lokal. Nilai rata - rata *response time* pada skenario jaringan internet lebih besar dibandingkan dengan skenario jaringan lokal. Untuk nilai perbandingan kepada keempat metode pengiriman, nilai rata – rata *response time* yang dimiliki oleh HTTP/2 SSE dan WebSocket relatif sama namun secara keseluruhan nilai rata – rata WebSocket lebih kecil dibandingkan dengan HTTP/2 SSE.

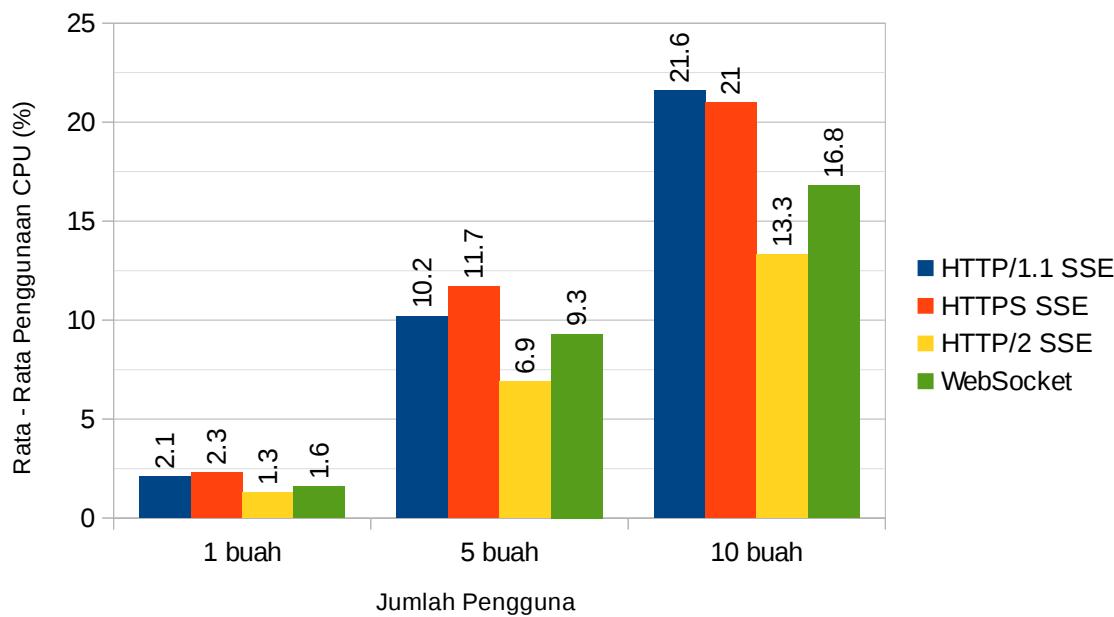
**Tabel 4.2 Rata - rata response time pada skenario internet**

Metode Pengiriman	<i>Response Time</i> (ms) dengan jeda antar pengiriman						Nilai Rata – rata (ms)
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms	
HTTP/1 SSE	10908.2	15793.3	26885.4	25979.3	25734.6	25880.3	21863.6
HTTP/2 SSE	696.5	722.1	1071.3	1661.3	1701.3	1142.5	1165.8
HTTPS SSE	40336.9	48274.9	61342.9	55929.0	57696.9	62834.5	54402.5
WebSocket	620.9	953.0	1368.1	1247.4	1156.4	1581.1	1154.5

Hasil lain yang didapatkan dari percobaan ini adalah ketika digunakannya Serveo ada **kemungkinan hilangnya suatu pesan yang dikirimkan oleh mikrokontroler maupun web browser** ketika pengiriman dilakukan melalui HTTP/1.1 maupun HTTP/1.1 over TLS. Untuk lebih detail hasil percobaan dapat dilihat pada Lampiran 3.

## 4.2 Hasil Perbandingan Penggunaan CPU

Di dalam proses pengambilan data presentase penggunaan CPU, sebuah *web browser* diibaratkan sebagai seorang penghuni rumah yang sedang mengakses *web application*. Setiap *web browser* mengirimkan perintah dari *web application* menuju mikrokontroler dengan jeda antar pengiriman sebesar satu detik secara terus-menerus. Selama itu pula dilakukan pengamatan terhadap perubahan nilai presentase penggunaan CPU dengan memvariasikan jumlah penghuni rumah yang mengakses *web application*. Hasil yang didapatkan dari percobaan ini telah dirangkum pada Gambar 4.3.



Gambar 4.3 Grafik rata - rata penggunaan CPU

Berdasarkan hasil yang diperoleh pada percobaan ini dapat dinyatakan bahwa HTTP/2 SSE memiliki nilai rata - rata presentase penggunaan CPU terkecil. Selain itu,

nilai rata – rata presentase penggunaan CPU dari keempat metode cenderung memiliki urutan yang sama. Urutan dapat dimulai dari pemilik rata - rata presentase penggunaan CPU terkecil yakni HTTP/2 SSE, WebSocket, HTTP/1.1 SSE sampai dengan HTTPS SSE. Selain itu, jumlah pengguna web application juga memengaruhi nilai presentase penggunaan CPU. Semakin banyak jumlah pengguna maka semakin besar pula nilai presentase penggunaan CPU.

## **BAB V**

### **PENUTUP**

#### **5.1 Kesimpulan**

Berdasarkan data yang diperoleh serta analisis yang sudah dilakukan, maka dapat ditarik beberapa kesimpulan dari penelitian tersebut, diantaranya :

1. WebSocket memiliki nilai *response time* terkecil di antara ketiga metode lainnya.
2. HTTP/2 *Server-Sent Events* memiliki nilai presentase penggunaan CPU terkecil di antara ketiga metode lainnya.
3. HTTPS *Server-Sent Events* cenderung memiliki nilai terbesar baik dalam pengujian *response time* maupun presentase penggunaan CPU.
4. Semakin kecil jeda antar pengiriman dapat menyebabkan bertambahnya nilai *response time*. Hal ini berlaku untuk keempat metode pengiriman.
5. Semakin banyak jumlah pengguna *web application* dapat meningkatkan nilai presentase penggunaan CPU. Hal ini berlaku untuk keempat metode pengiriman.

#### **5.2 Saran**

Pada pengembangan selanjutnya untuk memperbaiki proyek akhir ini dapat dilakukan beberapa pertimbangan yaitu :

1. Untuk mendapatkan data *response time* yang lebih akurat, dapat dilakukan beberapa pengujian *response time* pada saat yang berbeda-beda seperti halnya ketika pagi, siang, sore dan malam.
2. Perlunya notifikasi di dalam *web application* ketika ada perubahan status piranti elektronik yang ada di rumah.

## DAFTAR PUSTAKA

- Cirani, Simone, Gianluigi Ferrari, Marco Picone, and Luca Veltri. 2018. *Internet of Things : Architectures, Protocols and Standards*. New Jersey: John Wiley & Sons, Inc.
- Hasibuan, Zainal A. 2007. *Metodologi Penelitian Pada Bidang Ilmu Komputer Dan Teknologi Informasi, Konsep, Metode Teknik, Dan Aplikasi*. 2011.
- Hillar, Gastón C. 2017. *MQTT Essentials : A Lightweight IoT Protocol : The Preferred IoT Publish-Subscribe Lightweight Messaging Protocol*. Birmingham: Packt.
- Ably. 2018. “Long Polling - Concepts and Considerations.” 2018. <https://www.ably.io/concepts/long-polling>.
- Briere, Danny, and Pat Hurley. 2011. *Smart Homes For Dummies*. New Jersey: John Wiley & Sons.
- Cirani, Simone, Gianluigi Ferrari, Marco Picone, and Luca Veltri. 2018. *Internet of Things : Architectures, Protocols and Standards*. New Jersey: John Wiley & Sons, Inc.
- Elman, Julia, and Mark Lavin. 2014. *Lightweight Django: Using REST, WebSockets, and Backbone - Julia Elman, Mark Lavin*. California: O'Reilly Media, Inc.
- Estep, Eliot. 2013. “Mobile HTML5: Efficiency and Performance of WebSockets and Server-Sent Events.” Aalto University.
- Hillar, Gastón C. 2017. *MQTT Essentials : A Lightweight IoT Protocol : The Preferred IoT Publish-Subscribe Lightweight Messaging Protocol*. Birmingham: Packt.
- Ibrahim, Dogan. 2014. “A New Approach for Teaching Microcontroller Courses to Undergraduate Students.” *Procedia - Social and Behavioral Sciences* 131 (May): 411–14.
- Kayal, Paridhika, and Harry Perros. 2017. “A Comparison of IoT Application Layer Protocols through a Smart Parking Implementation.” *Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks, ICIN 2017*, no. January: 331–36.
- Kwan, Joel, Yassine Gangat, Denis Payet, and Remy Courdier. 2016. “An Agentified Use of the Internet of Things.” In *2016 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 311–16. IEEE.

- Leach, Paul J., Tim Berners-Lee, Jeffrey C. Mogul, Larry Masinter, Roy T. Fielding, and James Gettys. 1999. "Hypertext Transfer Protocol -- HTTP/1.1."
- Ludin, Stephen, and Javier Garza. 2017. *Learning HTTP/2: A Practical Guide for Beginners - Stephen Ludin, Javier Garza*. Sebastopol: O'Reilly Media, Inc.
- MDN. 2019. "Protocol Upgrade Mechanism - HTTP | MDN." 2019. [https://developer.mozilla.org/en-US/docs/Web/HTTP/Protocol\\_upgrade\\_mechanism](https://developer.mozilla.org/en-US/docs/Web/HTTP/Protocol_upgrade_mechanism).
- Muhammad, Panser Brigade, Widhi Yahya, and Achmad Basuki. 2018. "Analisis Perbandingan Kinerja Protokol Websocket Dengan Protokol SSE Pada Teknologi Push Notification." *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer (JPTIIK) Universitas Brawijaya* 2 (6): 2235–42.
- Örnmyr, Oliver, and Rasmus Appelqvist. 2017. "Performance Comparison of XHR Polling , Long Polling , Server Sent Events and Websockets." Blekinge Institute of Technology.
- Peon, R., and H. Ruellan. 2015. "HPACK: Header Compression for HTTP/2." <https://doi.org/10.17487/RFC7541>.
- Ramli, Kalamullah, Asril Jarin, and Suryadi Suryadi. 2018. "A Real-Time Application Framework for Web-Based Speech Recognition Using HTTP/2 and SSE." *Indonesian Journal of Electrical Engineering and Computer Science* 12 (3): 1230.
- Rhee, Kyung-Hyune, and Jeong Hyun Yi. 2015. *Information Security Applications: 15th International Workshop, WISA 2014*. Berlin: Springer.
- Rochman, Hudan Abdur, Rakhmadhany Primananda, and Heru Nurwasito. 2017. "Sistem Kendali Berbasis Mikrokontroler Menggunakan Protokol MQTT Pada Smarthome." *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer* 1 (6): 445–55.
- Saito, Nobuo, and David Menga. 2015. *Ecological Design of Smart Home Networks : Technologies, Social Impact and Sustainability*. Cambridge: Woodhead Publishing.
- Souders, Steve. 2009. *Even Faster Web Sites: Performance Best Practices for Web Developers*. California: O'Reilly Media.
- Udayashankara, V, and M S Mallikarjunaswamy. 2009. *Microcontroller*. New Delhi: Tata McGraw-Hill Education.

- Vasseur, Jean-Philippe, Adam Dunkels, Jean-Philippe Vasseur, and Adam Dunkels. 2010.  
“What Are Smart Objects?” *Interconnecting Smart Objects with IP*, January, 3–20.
- Wang, Vanessa., Frank. Salim, and Peter. Moskovits. 2013. *The Definitive Guide to HTML5 WebSocket*. Apress

## LAMPIRAN

### LAMPIRAN 1 – Instalasi

#### 1. Instalasi Mosquitto

Instalasi Mosquitto dapat dilakukan dengan cara mengunduh langsung melalui *default repository* Raspberry Pi 3.

```
$ sudo apt update  
$ sudo apt install -y mosquitto mosquitto-clients
```

Pada penginstalan di atas, “mosquitto” adalah nama paket *broker* untuk protokol MQTT sedangkan “mosquitto-clients” adalah paket pendukung untuk melakukan percobaan penggunaan protokol MQTT sebagai *client*. Setelah instalasi dilakukan, cek status Mosquitto dengan perintah:

```
$ service mosquitto status
```

Untuk menjalankan Mosquitto secara otomatis setelah Raspberry Pi 3 dihidupkan adalah dengan memasukkan perintah berikut.

```
$ sudo systemctl enable mosquitto.service
```

Setelah Mosquitto aktif, lakukan percobaan menggunakan mosquitto-client. Bukalah terminal baru dan masukkan perintah berikut untuk menjalankan *client* sebagai *subscriber* :

```
$ mosquitto_sub -h localhost -t testing
```

Kemudian buka terminal baru lagi dan masukkan perintah berikut untuk menjalankan *client* sebagai *publisher* :

```
$ mosquitto_pub -h localhost -t testing -m hai
```

Pada kedua perintah di atas, “testing” adalah *topic* yang digunakan untuk oleh protokol MQTT sedangkan “hai” adalah pesan yang dikirimkan oleh *publisher* yang akan diterima oleh *subscriber* dengan *topic* yang sama.

## 2. Instalasi Node.js

Sebelum menginstall Node.js, periksa terlebih dahulu *processor* yang digunakan oleh Raspberry Pi dengan perintah :

```
$ uname -m
```

Sebagai catatan, seluruh Raspberry Pi 3 memiliki processor dengan model ARMv8, sehingga pada penelitian ini diunduh Node.js untuk Linux Binaries (ARM) dengan opsi ARMv8 dari halaman resminya (<https://nodejs.org/en/download/>). Setelah Node.js berhasil diunduh, masuk pada folder Downloads dan ekstraklah hasil unduhan dengan perintah :

```
$ cd ~/Downloads
$ tar -xzf node-v10.15.3-linux-arm64.tar.xz
```

Kemudian salinlah hasil ekstrak Node.js ke *directory* /usr/local/

```
$ cd node-v10.15.3-linux-arm64
$ sudo cp -R * /usr/local/
```

Setelah itu, untuk cek apakah instalasi berhasil dengan cara mengecek versi npm dan node yang telah di-*install*,

```
$ npm -v
```

```
$ node -v
```

Setelah node dan npm berhasil ter-*install*, *install* modul Vuejs-cli dengan memasukkan perintah:

```
$ npm install -g @vue-cli
```

## LAMPIRAN 2 – Source Code

### A. *Source Code* untuk ***Web Browser***

#### 1. *Source Code* untuk **keempat program pengirim**

##### a. Untuk menerima data dari *Web API*

```
client.addEventListener("message", e =>
{ console.log(e) //e adalah pesan yang diterima
})
```

- b. Apabila koneksi antara *web browser* dengan *web API* telah terhubung maka akan menjalankan perintah tertentu.

```
client.onopen = () => {
console.log("koneksi telah terjalin")
}
```

#### 2. *Source Code* untuk **WebSocket** pada sisi *web browser*

##### a. Untuk membuka koneksi dengan *Web API*

```
client = new WebSocket(
"ws://<Web API Domain / IP Address>" + "/" + <path>);
```

##### b. Untuk mengirim data menuju *Web API*

```
client.send(<message>);
```

3. Source Code untuk **HTTP/1.1 SSE** pada sisi *web browser*

a. Untuk membuka koneksi dengan *Web API*

```
client = new EventSource(
"http://<Web API Domain / IP Address>" + "/" +
<path>);
```

b. Untuk mengirim data menuju *Web API*

```
fetch("https://<Web API Domain / IP Address>", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: <message>
}).then(response => {
  console.log(response);
});
```

4. Source Code untuk **HTTPS SSE** pada sisi *web browser*

a. Untuk membuka koneksi dengan *Web API*

```
client = new EventSource(
"https://<Web API Domain / IP Address>" + "/" +
<path>);
```

b. Untuk mengirim data menuju *Web API*

```
fetch("https://<Web API Domain / IP Address>", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: <message>
}).then(response => {
  console.log(response);
});
```

5. Source Code untuk **HTTP/2 SSE** pada sisi *web browser*

a. Untuk membuka koneksi dengan *Web API*

```
client = new EventSource(
  "https://<Web API Domain / IP Address>" + "/" +
  <path>);
```

b. Untuk mengirim data menuju *Web API*

```
fetch("https://<Web API Domain / IP Address>", {
  method: "POST",
  body: <message>
}).then(response => {
  console.log(response);
});
```

B. *Source Code* untuk *Web API*

1. *Source Code* untuk **MQTT**

a. Untuk membuka koneksi antara Web API dengan MQTT Broker

```
const mqtt = require('mqtt')
var client = mqtt.connect(
  'mqtt://<MQTT Broker Domain / IP Address>')
```

b. Untuk menerima data dari MQTT Broker

```
client.subscribe(<topic>);
client.on('message', function (topic, message) {
  // message dalam bentuk buffer
  console.log(message.toString('utf-8'))
});
```

c. Untuk mengirim data menuju MQTT Broker

```
client.publish(<topic>, <message>)
```

2. *Source Code* untuk **WebSocket**

a. Membuat *Server API*

```

var WebSocketServer = require('websocket').server;
var http = require('http');

var server = http.createServer(function (request,
response) {
    console.log(request.headers);
    response.writeHead(404);
    response.end();
});

server.listen(1337, function () {
    console.log(`websocket on port 1337!`);
});

var wsServer = new WebSocketServer({
    httpServer: server
});

```

b. Menangkap permintaan data dari *web browser*

```

wsServer.on('request', function (request) {

    var connection = request.accept()

    //mengirim pesan menuju web browser
    connection.sendUTF(<message>);

    //menerima pesan dari web browser
    connection.on('message', function (message) {
        console.log(JSON.parse(message.utf8Data))
    });

    //ketika koneksi antara web browser
    //dengan web API terputus
    connection.on('close', function (reasonCode,
description) {
        console.log(connection.remoteAddress + ' '
disconnected.');
    });
});

```

**3. Source Code untuk HTTP/1.1 SSE**

a. Membuat *Server API*

```
const express = require('express')
var cors = require('cors');
const app = express()

app.use(cors())
app.use(express.static('public'))
app.use(express.json());

app.listen(3000, function () { console.log(`http/1.1
on port 3000!`) }) ;
```

b. Menangkap permintaan data dari *web browser*

```
app.get('/index', (req, res) => {
    res.writeHead(200, {
        'Content-Type': 'text/event-stream',
        'Cache-Control': 'no-cache',
        'Connection': 'keep-alive',
    });
    res.write('\n');
    res.write(
        "event: message\n" +
        "data: " + <message> + "\n\n"
    );
}

app.post('/', (req, res) => {
    console.log(req.body)
    res.send('OK');
});
```

4. *Source Code* untuk **HTTPS SSE**

a. Membuat *Server API*

```
const express = require('express')
var cors = require('cors');
const app = express()
var https = require('https')

app.use(cors())
app.use(express.static('public'))
```

```
app.use(express.json());

https.createServer({
    key: fs.readFileSync('./cert/server.key'),
    cert: fs.readFileSync('./cert/server.crt')
}, app)
.listen(3002, function () {
    console.log(`https 1.1 on port 3002!`)
})
```

b. Menangkap permintaan data dari *web browser*

```
app.get('/index', (req, res) => {
    res.writeHead(200, {
        'Content-Type': 'text/event-stream',
        'Cache-Control': 'no-cache',
        'Connection': 'keep-alive',
    });
    res.write('\n');
    res.write(
        "event: message\n" +
        "data: " + <message> + "\n\n"
    );
}

app.post('/', (req, res) => {
    console.log(req.body)
    res.send('OK');
});
```

4. *Source Code* untuk **HTTP/2 SSE**

a. Membuat *Server API*

```
const express = require('express')
var cors = require('cors');
const app = express()
var https = require('https')

app.use(cors())
app.use(express.static('public'))
app.use(express.json())

https.createServer({
```

```

    key: fs.readFileSync('./cert/server.key'),
    cert: fs.readFileSync('./cert/server.crt')
}, app)
.listen(3002, function () {
  console.log(`https 1.1 on port 3002!`)
})

```

b. Menangkap permintaan data dari *web browser*

```

const http2 = require('http2');
http2.createSecureServer({
  key: fs.readFileSync('./cert/server.key'),
  cert: fs.readFileSync('./cert/server.crt')
},
(req, res) => {
  if (req.headers[':method'] == 'POST') {
    let body = '';
    req.setEncoding('utf8');
    req.on('data', (chunk) => {
      body += chunk
    })
    req.on('end', () => {
      try{
        //menerima pesan dari web browser
        console.log(body);
        res.end()
      }catch(err){
        res.statusCode = 400;
        res.end(err)
      }
    })
    res.setHeader(
      'Access-Control-Allow-Origin', '*')
    res.write('OK')

  } else
  if (req.headers[':method'] == 'GET') {
    new Promise((resolve) => {
      res.statusCode = 200;
      res.setHeader(
        'Content-Type', 'text/event-stream');
      res.setHeader(

```

```
'Cache-Control', 'no-cache')
    res.setHeader(
'Access-Control-Allow-Origin', '*')
    res.write(`\n`)
    resolve(null)
}).then(() => {
    //mengirim pesan menuju web browser
    res.write(
"event: message\ndata: "+msg+"\n\n")
    res.end()
})
})}
}).listen(
3001, () => {
    console.log("http/2 on port 3001");
}
)
```

### LAMPIRAN 3 – Hasil Penelitian

Selama pengujian berlangsung, terdapat beberapa nilai response time yang hilang terutama ketika menggunakan Serveo. Nilai yang hilang ini ditandai dengan tanda ( - ).

#### A. Pengujian *response time* pada jaringan lokal

##### 1. HTTP/1.1 SSE

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	151	281	238	210	389	200
2	98	275	255	212	367	253
3	33	242	257	226	443	228
4	254	207	272	289	465	371
5	178	204	246	252	487	455
6	124	165	244	262	432	421
7	40	125	317	265	507	429
8	58	77	286	271	533	438
9	75	43	281	351	479	444
10	32	25	278	363	557	572
11	73	26	271	321	501	693
12	73	128	272	398	511	643
13	181	102	292	345	586	647
14	112	92	290	368	591	712
15	183	78	292	376	783	715
16	94	76	289	456	787	682

17	30	91	293	414	792	696
18	91	247	297	434	802	710
19	30	224	298	436	803	724
20	92	196	296	528	806	731
21	31	175	290	537	808	737
22	100	145	288	460	815	741
23	39	104	307	551	818	768
24	105	81	299	520	823	856
25	49	54	300	524	930	778
26	41	30	298	527	936	870
27	213	31	269	646	840	874
28	146	30	248	560	946	796
29	211	30	237	563	853	801
30	125	27	236	577	859	805
31	49	28	236	593	862	807
32	122	29	194	599	866	811
33	42	30	221	604	869	813
34	37	29	227	618	990	920
35	236	27	225	619	874	822
36	182	163	162	627	878	825
37	89	128	159	652	880	828
38	133	125	116	655	880	825
39	71	107	119	669	882	828
40	29	84	126	671	883	828
41	289	108	147	680	996	838
42	210	82	145	683	996	845

43	213	79	148	695	998	847
44	523	75	157	697	996	854
45	549	82	160	705	993	856
46	464	64	142	728	994	858
47	547	55	160	739	993	854
48	462	35	157	739	994	860
49	372	26	157	768	992	895
50	453	32	177	782	996	898
Rata-rata	162.68	99.98	233.42	515.3	781.22	714.04

## 2. HTTP/2 SSE

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	298	163	305	248	311	210
2	213	125	666	258	343	211
3	136	131	672	554	362	265
4	54	91	672	563	871	276
5	44	57	679	567	880	281
6	179	53	674	571	887	286
7	90	49	674	573	892	291
8	53	49	672	575	893	299
9	78	52	671	584	896	306
10	232	47	678	584	897	492
11	150	49	676	592	901	495
12	197	56	678	590	902	498

13	118	67	679	593	909	501
14	173	49	677	597	911	505
15	112	43	673	594	912	507
16	187	42	672	599	912	510
17	122	141	669	587	912	512
18	61	132	667	592	915	515
19	153	126	664	607	918	516
20	203	82	660	607	926	520
21	125	54	655	609	928	521
22	51	38	654	602	929	524
23	54	38	656	591	921	534
24	223	41	656	592	924	549
25	138	150	639	594	914	555
26	62	122	741	592	914	558
27	140	95	740	708	916	561
28	78	97	732	716	913	565
29	153	57	731	716	918	561
30	78	39	731	719	922	593
31	136	47	717	720	925	596
32	56	232	714	718	928	597
33	143	219	712	708	931	598
34	190	346	711	706	934	601
35	134	303	708	708	937	605
36	54	262	702	710	939	610
37	169	216	684	710	933	608
38	96	195	661	710	927	611

39	151	157	627	690	924	611
40	64	130	625	691	924	615
41	46	98	624	691	929	615
42	264	120	622	690	927	615
43	257	92	618	690	928	570
44	430	240	615	677	929	571
45	347	211	609	645	934	588
46	254	169	606	616	926	588
47	426	131	600	612	927	590
48	656	115	597	613	925	593
49	610	87	593	615	926	584
50	517	141	587	614	930	590
Rata-rata	179.1	116.92	658.9	620.16	882.64	509.46

### 3. HTTPS SSE

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	290	268	191	424	503	407
2	207	231	192	322	391	325
3	275	368	299	339	570	376
4	164	330	311	350	485	502
5	106	287	257	374	479	411
6	40	243	258	511	601	424
7	83	197	354	408	490	564
8	35	172	435	416	498	443

9	90	134	372	425	500	567
10	40	106	457	442	508	461
11	105	82	325	633	647	465
12	217	115	402	460	535	465
13	145	257	333	483	542	480
14	211	224	329	495	548	614
15	125	219	323	648	554	497
16	51	176	319	513	693	638
17	110	132	527	669	567	531
18	47	232	372	531	583	544
19	111	208	393	736	584	547
20	43	177	493	540	739	574
21	137	158	427	556	604	720
22	87	218	427	561	608	605
23	124	192	428	746	618	754
24	55	149	588	763	830	632
25	123	133	483	766	662	788
26	52	127	486	593	830	838
27	128	108	482	601	876	659
28	70	124	465	616	859	832
29	157	103	575	871	869	858
30	63	89	584	826	884	706
31	139	88	589	849	897	717
32	56	84	577	946	913	887
33	142	81	767	976	1019	975
34	55	49	582	932	973	981

35	41	192	587	934	986	945
36	269	167	589	938	1013	956
37	187	160	595	946	1031	961
38	256	241	592	958	1139	1073
39	169	227	579	963	1048	984
40	81	217	582	972	1077	1007
41	41	182	579	986	1080	1019
42	268	142	537	1093	1083	1041
43	181	95	544	1133	1106	1047
44	295	74	593	1024	1197	1063
45	215	42	577	1034	1126	1065
46	121	44	571	1040	1225	1204
47	104	38	570	1050	1274	1227
48	49	35	565	1162	1236	1237
49	179	39	559	1167	1302	1221
50	96	36	1087	1267	1329	1227
Rata-rata	128.7	155.84	482.16	739.76	814.22	761.28

#### 4. WebSocket

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	239	170	146	117	245	244
2	182	177	175	127	281	296
3	241	139	204	184	637	310
4	166	227	205	185	649	317

5	76	213	199	208	656	322
6	147	167	194	209	660	322
7	71	131	192	210	663	324
8	19	114	189	211	665	363
9	53	113	283	210	677	373
10	162	77	288	217	680	381
11	78	70	286	220	684	394
12	22	67	293	222	688	398
13	64	178	296	262	691	402
14	175	142	295	267	695	406
15	104	123	290	281	698	412
16	24	112	280	284	703	415
17	71	106	279	289	706	417
18	21	71	276	289	708	424
19	72	63	271	294	711	430
20	16	56	265	300	713	444
21	76	26	257	313	715	446
22	20	16	261	317	720	449
23	20	17	258	327	722	453
24	192	19	260	332	727	468
25	129	21	268	346	729	471
26	47	20	267	356	731	474
27	92	21	260	359	734	476
28	22	22	260	366	735	480
29	100	20	300	380	738	484
30	25	25	296	403	741	486
31	109	19	290	408	745	490

32	33	21	284	411	746	493
33	27	23	275	417	748	495
34	218	20	307	420	750	512
35	156	22	312	425	752	514
36	82	23	311	429	754	517
37	25	25	309	720	755	520
38	22	22	265	720	756	521
39	226	20	304	716	759	525
40	144	21	313	717	763	526
41	85	21	287	717	764	528
42	23	22	294	723	765	529
43	133	20	600	724	767	536
44	73	23	603	723	768	575
45	141	21	603	724	770	577
46	80	20	599	717	770	579
47	144	26	577	721	773	612
48	82	27	578	683	775	614
49	149	23	576	687	778	621
50	89	19	572	688	778	625
Rata-rata	95.34	63.22	317.04	411.5	706.76	459.8

### B. Pengujian *response time* pada jaringan internet

#### 1. HTTP/1.1 SSE

No	Response Time (ms) dengan jeda antar pengiriman

	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	-	-	-	-	-	-
2	4695	9214	19065	17519	17084	18320
3	4108	9269	19260	18137	17682	18542
4	4655	8833	19763	18166	17817	18561
5	5856	8899	19776	18726	18290	18575
6	4823	9292	20310	18771	18313	18708
7	5173	9260	20331	19351	18886	18782
8	5384	9659	20917	19365	18970	20031
9	5552	9771	20939	20302	19502	20071
10	7001	10322	21447	20600	19542	20695
11	6081	10288	21454	20959	20121	20772
12	6002	10957	22099	21040	20156	21351
13	8937	10924	22158	22381	21067	21367
14	7352	13036	22683	22400	21093	21959
15	7518	13210	22717	22405	21679	21973
16	7644	12077	23281	22618	21878	22558
17	9831	12210	23309	22975	23718	22577
18	8051	12615	23888	23002	23729	23577
19	8578	12737	23890	23601	24023	23593
20	8700	13301	25112	23615	24040	24141
21	9159	13270	25143	24131	24161	24164
22	9085	13421	25716	24150	24196	24793
23	9613	13514	25705	24788	24779	24801
24	11727	14129	26273	24805	24812	25336
25	10947	14982	26316	25392	25527	25602
26	10857	14954	26875	25414	25544	25956

27	10768	15100	26885	26406	26185	26204
28	13457	15567	27460	26430	26232	26556
29	11357	15528	27473	27025	26791	26812
30	12384	16084	28058	27056	26831	27315
31	12199	16797	28087	27654	27451	27523
32	12311	16715	28699	28378	27467	27973
33	15128	19364	28750	28390	28020	27981
34	12335	19427	29295	28675	28030	28598
35	13832	20006	29297	29047	29171	29345
36	13744	20042	29890	29277	29189	29388
37	13660	20450	29892	30401	29843	29421
38	16937	20514	31044	30400	29932	30010
39	13901	20955	31611	30419	31208	30037
40	14046	21066	31602	31046	31216	30616
41	17540	21518	32509	31148	31247	30645
42	14764	21507	32800	31670	31392	31472
43	14831	22049	33096	32212	31860	31463
44	15184	22031	33287	32412	32109	31873
45	18493	22567	33793	32422	32924	32060
46	15458	22558	33795	33119	32936	32580
47	15796	23053	34837	33172	33223	32579
48	15903	23117	35465	33624	33221	33279
49	16612	23626	35472	33690	33942	33796
50	16536	24087	35865	34302	33968	33803
Rata-rata	10908.3	15793.3	26885.5	25979.3	25734.6	25880.3

## 2. HTTP/2 SSE

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	633	760	760	792	796	682
2	675	722	762	1341	805	692
3	708	776	1082	1344	1378	787
4	618	733	1096	1347	1392	790
5	644	692	1103	1351	1397	794
6	656	647	1106	1352	1405	798
7	684	663	1110	1354	1410	802
8	597	621	1111	1354	1418	804
9	685	579	1109	1360	1422	808
10	597	598	1107	1361	1682	1089
11	581	606	1104	1363	1686	1093
12	762	570	1101	1365	1691	1098
13	683	547	1099	1367	1697	1103
14	596	703	1098	1577	1700	1107
15	589	671	1095	1579	1703	1112
16	788	642	1092	1581	1709	1168
17	709	625	1089	1584	1711	1171
18	1001	593	1085	1591	1712	1173
19	914	622	1081	1598	1714	1174
20	936	592	1077	1603	1716	1177
21	846	1149	1074	1603	1719	1179
22	759	1100	1070	1608	1722	1182

23	666	1064	1067	1609	1723	1183
24	561	1016	1066	1611	1726	1187
25	687	973	1062	1613	1728	1225
26	612	992	1112	1613	1729	1227
27	743	948	1109	1683	1732	1229
28	635	965	1108	1686	1736	1233
29	566	915	1106	1686	1737	1237
30	609	873	1109	1685	1738	1242
31	587	834	1107	1687	1741	1247
32	570	784	1106	1687	1744	1253
33	755	746	1101	1687	1746	1259
34	671	702	1096	1957	1894	1261
35	873	659	1093	1961	1894	1266
36	829	632	1091	1961	1896	1270
37	729	589	1087	1961	1899	1276
38	951	747	1083	1961	1900	1277
39	858	717	1079	1961	1902	1278
40	813	677	1076	1963	1905	1282
41	721	634	1070	1963	1906	1284
42	629	640	1065	1963	1906	1287
43	718	600	1060	1966	1909	1289
44	687	585	1057	1968	1910	1291
45	650	587	1052	1968	1911	1293
46	551	596	1047	1971	1912	1292
47	735	587	1042	1971	1912	1292
48	648	626	1040	1983	1914	1294

49	578	609	1034	1981	1915	1296
50	533	597	1029	1983	1915	1293
Rata-rata	696.52	722.1	1071.3	1661.28	1701.3	1142.52

### 3. HTTPS SSE

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	-	40587	-	-	-	-
2	33771	39885	52885	47931	49057	54320
3	32661	39931	53179	47953	49638	54829
4	32574	40476	54067	48432	50231	54843
5	33307	40539	54312	48511	50247	55847
6	33428	41108	54647	49119	50854	55860
7	33932	41073	55526	49343	50868	56321
8	33749	41637	55531	49712	51460	56563
9	34543	41611	55892	49724	51475	56957
10	34756	42177	55903	50345	52055	57464
11	35990	42894	56440	50370	52079	57973
12	36394	42816	56510	50915	52655	57987
13	36308	43089	57177	50929	52666	58349
14	39024	43370	57190	51967	54294	58605
15	38934	43341	57967	51990	54308	58955
16	36489	43886	58050	52678	54898	59796
17	36405	43848	58645	52688	54926	59817
18	36894	44994	58753	53338	55497	60405

19	36813	45057	59240	53365	55511	60602
20	37330	45462	59367	53936	56121	61197
21	37263	45420	59831	53946	56133	61251
22	37949	46072	60057	54572	56721	61819
23	38127	46198	60426	54585	56730	61833
24	38537	46577	60427	55607	57357	62396
25	38446	49140	61037	55615	57368	62523
26	42561	49469	61031	56413	57862	63955
27	38976	49430	61758	56423	57874	63969
28	39477	50324	61753	57118	58540	63975
29	43088	50286	62419	57138	58549	64146
30	39921	50923	62423	57791	59138	64642
31	40033	50954	63757	57813	59149	64722
32	40407	51417	63751	58402	59739	65366
33	40318	51482	63769	58411	59746	65564
34	40825	52098	63761	59010	60345	66083
35	40734	52261	64397	59016	60354	66091
36	41258	52944	64505	59603	61099	66277
37	45466	53004	65555	59605	61505	66291
38	45968	53711	65559	60232	62115	66880
39	46077	53679	65724	60239	62124	66890
40	46689	53707	65735	60821	62800	67381
41	46605	54261	66178	60827	62812	67393
42	46879	54191	66409	61598	63373	68060
43	47081	54818	66785	61861	63390	68289
44	47214	54728	67517	61866	63985	68667

45	48136	55823	67518	62467	63997	68679
46	48048	55851	67578	62473	64606	69293
47	47955	56037	68171	63272	64620	69309
48	48556	56343	68296	63281	65219	69877
49	48517	56575	68898	63630	65228	69886
50	52095	58235	69495	63639	65828	70695
Rata-rata	40336.9	48274.8	61342.9	55929	57696.9	62834.5

#### 4. WebSocket

No	Response Time (ms) dengan jeda antar pengiriman					
	100 ms	50 ms	10 ms	5 ms	1 ms	0 ms
1	587	692	580	698	654	979
2	525	662	598	708	678	991
3	641	631	644	893	678	1181
4	555	587	647	907	1016	1188
5	1000	1205	647	908	1022	1191
6	924	1167	655	912	1026	1194
7	832	1125	869	916	1028	1198
8	757	1312	909	925	1030	1200
9	843	1272	943	1065	1034	1204
10	748	1232	942	1068	1038	1207
11	655	1188	1553	1122	1043	1460
12	693	1146	1551	1124	1045	1463
13	610	1100	1549	1124	1048	1467

14	557	1066	1548	1127	1050	1474
15	616	1141	1545	1129	1167	1477
16	538	1116	1542	1130	1171	1479
17	616	1076	1540	1130	1175	1481
18	522	1032	1537	1130	1177	1484
19	614	998	1534	1325	1180	1487
20	533	957	1531	1325	1185	1629
21	616	911	1531	1326	1187	1631
22	530	868	1528	1327	1190	1632
23	640	825	1529	1329	1195	1636
24	548	781	1528	1333	1198	1639
25	517	735	1525	1333	1202	1640
26	784	865	1520	1335	1205	1641
27	691	826	1518	1336	1208	1643
28	597	783	1514	1339	1210	1647
29	519	738	1511	1339	1213	1648
30	640	694	1507	1344	1214	1652
31	650	651	1503	1344	1216	1654
32	565	686	1500	1348	1217	1657
33	672	656	1496	1349	1222	1658
34	576	615	1490	1354	1225	1658
35	658	647	1487	1353	1227	1660
36	581	602	1484	1354	1228	1662
37	499	559	1479	1355	1229	1868
38	563	790	1477	1357	1233	1868
39	501	720	1471	1356	1234	1869
40	593	757	1469	1356	1237	1872

41	514	716	1463	1481	1238	1871
42	506	704	1457	1483	1239	1873
43	671	1307	1454	1483	1241	1874
44	581	1264	1450	1483	1243	1876
45	509	1228	1445	1485	1349	1879
46	586	1181	1440	1485	1351	1881
47	511	1144	1699	1485	1354	1882
48	605	1084	1693	1485	1356	1881
49	514	1826	1690	1485	1358	1885
50	742	1784	1685	1484	1358	1885
Rata-rata	620.9	953.04	1368.14	1247.44	1156.44	1581.12

### C. Pengujian presentase penggunaan CPU

#### 1. HTTP/1.1 SSE

No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	2	10.9	26.2
2	2.3	11.9	23.8
3	2.3	10.2	21.5
4	2.6	10.8	22.8
5	2.6	12.3	25.2
Rata-rata	2.133	10.183	21.583

## 2. HTTP/2 SSE

No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	1.3	6.2	14.4
2	1.3	8.3	12.9
3	1.6	6.6	14.8
4	1.3	8.9	12.5
5	1.3	6.5	15.1
Rata-rata	1.3	6.9167	13.283

## 3. HTTPS SSE

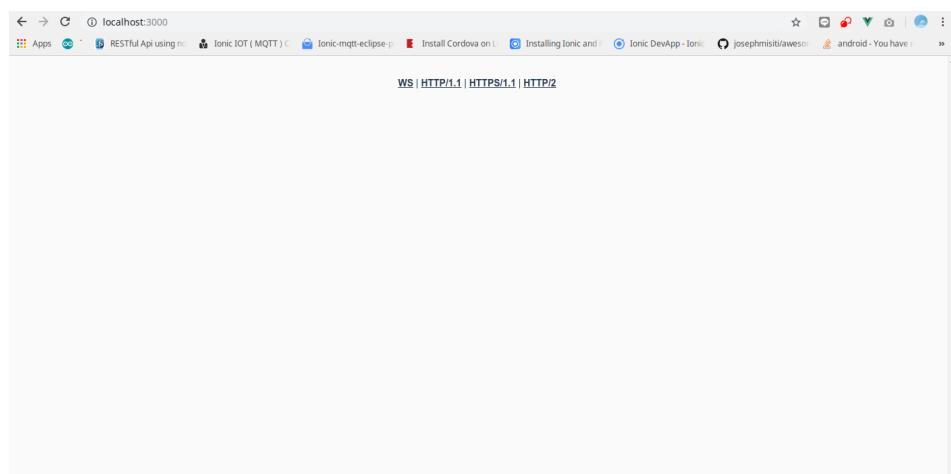
No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	2.6	14.4	21.8
2	2	12.8	22.8
3	3	12.9	24.1
4	2	11.8	22.2
5	3	13.4	25.1
Rata-rata	2.2667	11.7167	21

#### 4. WebSocket

No	Presentase penggunaan CPU (%)		
	1 pengguna	5 pengguna	10 pengguna
1	1.6	11.8	16.2
2	2.3	10.2	18.4
3	1.6	10.5	21.3
4	2	9.8	18.5
5	1.6	8.2	16.2
Rata-rata	1.683	9.25	16.767

#### LAMPIRAN 4 – Tampilan Website

Berikut merupakan tampilan awal website yang ditujukan untuk memilih metode pengiriman mana yang akan digunakan.



Berikutnya terdapat tampilan yang digunakan untuk mengendalikan perangkat dan mengamati suhu di rumah. Pada tampilan ini pula HTTP/1.1 SSE, HTTPS SSE, HTTP/2 SSE ataupun WebSocket diterapkan sesuai dengan yang telah dipilih pada tampilan sebelumnya.

