BAB 4 IMPLEMENTASI

Pada bab ini berisi penjelasan tentang proses implementasi dari rancangan penelitian. Implementasi yang dilakukan adalah implementasi protokol Websocket dan SSE pada server, implementasi *client* pada perangkat Android, dan implementasi *tools* untuk mendapatkan *resource* kinerja pengiriman masing-masing protokol pada sisi server.

4.1 Implementasi Server

Dalam mekanisme pengiriman notifikasi, *server* berperan untuk mengatur dan mengirimkan notifikasi kepada *client*. Pada penelitian ini *server* juga berperan sebagai *broker* sehingga koneksi dari *client* ke *server* bersifat langsung atau *direct*. Agar *server* dari masing-masing protokol dapat bekerja mengirimkan notifikasi, dibutuhkan *framework, library,* dan konfigurasi.

4.1.1 Implementasi Flask

Flask adalah *framework* yang diterapkan menggunakan bahasa python. Fungsi dari *framework* ini adalah untuk membantu *web developer* maupun *developer* lainnya untuk membuat sebuah aplikasi atau layanan berbasis *web*. Pada penelitian ini flask digunakan untuk membuat *push service* yang akan diimplementasikan pada *server* karena flask juga berfungsi untuk mengirimkan notifikasi secara *broadcast*. Langkah pertama implementasi flask adalah melakukan instalasi Flask kedalam *server*. Instalasi flask pada *server* dilakukan dengan mengetikkan perintah sepeti pada tabel 4.1 pada terminal Ubuntu.

Tabel 4.1 Tabel Instalasi Flask

```
1  $ sudo pip install flask
```

Perintah instalasi flask berlaku untuk kedua protokol karena protokol Websocket dan protokol SSE juga menggunakan framework flask untuk mengirimkan notifikasi. Setelah melakukan instalasi flask, selanjutnya adalah melakukan instalasi *library* Socket.IO.

4.1.2 Implementasi Socket.IO

Socket.IO adalah *library* yang berfungsi untuk membuat sebuah *real time web application*. Sehingga sebuah *web application* atau *web service* untuk dapat mengirimkan informasi secara cepat jika menggunakan Socket.IO memungkinkan. Baik protokol Websocket maupun SSE juga menggunakan Socket.IO agar kedua *server* dapat mengirimkan notifikasi ke *client*. Untuk melakukan instalasi Socket.IO menggunakan perintah pada tabel 4.2 terminal Linux.

Tabel 4.2 Tabel Instalasi Socket.IO

```
1  $ sudo pip install flask-socket.io
```

Library ini dipanggil atau di-*import* pada python yang terinstall pada sistem operasi Linux menggunakan perintah pip.

4.1.3 Implementasi Push Service Websocket

Setelah instalasi Flask-SocketIO selesai, langkah selanjutnya adalah membuka *file* app.py yang ada di dalam folder Flask-Socketio/example untuk mekanisme pengiriman menggunakan protokol Websocket. Pada penelitian nama folder diganti menjadi Flask-tes dan melakukan backup dengan menggandakan *file* app.py menjadi app2.py. *File* app2.py adalah *file push service* dan berisi *source code* yang berfungsi untuk melakukan pengiriman notifikasi. Pada *file* tersebut dilakukan modifikasi untuk disesuaikan dengan skenario pengujian yang akan dilakukan.

Tabel 4.3 *Source Code Binding* IP *Address* pada Websocket

```
1  socketio.run(app, host='0.0.0.0', port=5000)
```

Source code pada tabel 4.3 adalah konfigurasi yang dilakukan oleh penulis untuk mengikat alamat *server* agar ketika berpindah jaringan lokal tetap dapat menggunakan alamat IP *server* sesuai dengan alamat IP pada jaringan yang bersangkutan untuk pengiriman notifikasi menggunakan protokol Websocket.

4.1.4 Implementasi Push Service SSE

Setelah implementasi *push service* pada protokol Websocket telah dilakukan, langkah berikutnya adalah implementasi *push service* pada protokol SSE. Untuk mekanisme pengiriman notifikasi menggunakan protokol SSE, *file* sse.py merupakan *file push service* yang terdapat pada folder Flask-SSE/ . sama halnya dengan *file push service* pada protokol Websocket, modifikasi *source code* juga dilakukan pada *file push service* protokol SSE untuk disesuaikan dengan skenario pengujian yang dilakukan.

Tabel 4.4 *Source Code* Pengiriman Notifikasi pada Protokol SSE

```
1  emit('my response', {'data': 'Message: '+a, 'count': i, 'time':
   time.time()}, broadcast=True
```

Source code pada tabel 4.4 adalah perintah yang berfungsi mengirimkan pesan notifikasi dari *server* kepada *client*. Parameter message merupakan parameter yang berisi pesan notifikasi yang akan dikirimkan ke *client*. Sedangkan parameter time berfungsi untuk menampilkan delay dari proses pengiriman notifikasi ke *client* sehingga pada *browser client* akan dicetak nilai *delay* pengiriman pesan menggunakan protokol SSE.

Tabel 4.5 Perintah *binding* alamat IP pada Protokol SSE

```
1  $ gunicorn SSE:app –worker-class gevent –bind 0.0.0.0
```

S*ource code* pada tabel 4.5 digunakan untuk implementasi *binding* alamat IP menjadi *localhost*. Terdapat perbedaan metode *binding* alamat IP jika dibandingkan dengan protokol Websocket. *Binding* alamat IP pada protokol SSE menggunakan perintah diatas pada terminal Linux. Namun untuk *binding* alamat IP *server* tetap menggunakan alamat 0.0.0.0 agar alamat IP *server* sesuai dengan alamat IP ketika berpindah jaringan.

4.2 Implementasi Client

Tahap implementasi pada *client* dibagi menjadi dua yaitu implementasi *client* untuk pengiriman notifikasi menggunakan protokol Websocket dan implementasi *client* untuk pengiriman notifikasi menggunakan protokol SSE. *Client* menggunakan *browser* pada masing-masing perangkat uji untuk mengakses antarmuka.

4.2.1 Implementasi Client Websocket

Untuk implementasi *client* pengiriman notifikasi menggunakan Websocket menggunakan *browser* sebagai antar muka untuk menerima notifikasi yang dikirimkan oleh *server*. Tabel 4.6 adalah *source code* yang diimplementasikan pada sisi *client* dengan *menggunakan* Javascript pada pengiriman notifikasi menggunakan protokol Websocket.

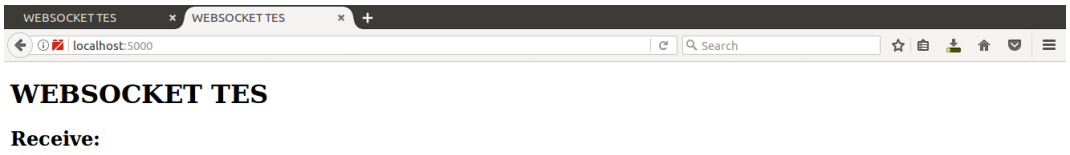
Tabel 4.6 *Source Code* Halaman Antar-Muka Websocket

```
1.  <!DOCTYPE HTML>
2.  <html>
3.  <head>
4.  <title>WEBSOCKET TES</title>
5.  <script          type="text/javascript"          src="/static/jquery-
6.  1.4.2.min.js"></script>
7.  <script                                        type="text/javascript"
8.  src="/static/socket.io.min.js"></script>
9.  <script type="text/javascript" charset="utf-8">
10.         $(document).ready(function(){
11. var path = window.location.host;
12.             var socket = io.connect('ws://'+ path + namespace);
13.
14.             socket.on('connect', function() {
15.                 socket.emit('my event', {data: 'Hello!'});
16.             });
17.            var sum = 0;
18.
```

```
19.             var n = 0;
20.  socket.on('my response', function(msg,msg2) {
21.                 var sec = new Date().getTime() / 1000;
22.                 var delay = sec - msg.time;
23.                 sum = sum + Math.abs(delay);
24.                 console.log(sum);
25.                 n = n + 1;
26.                 var avg = sum / n;
27.                    $('#log').append(msg.data + ' data ke - ' + n +'
28.  delay = ' + delay + ' avg = ' + avg +'</br>');
29.                });
30.
31.           });
32.  </script>
33.  </head>
34.  <body>
35.  <h1>WEBSOCKET TES</h1>
36.  <h2>Receive:</h2>
37.  <div id="log"></div>
38.  </body>
39.  </html>
40.
```

Source code pada tabel 4.6 apabila diakses dari sisi *client* maka berbentuk halaman situs. Di dalamnya berisi *code* yang berfungsi untuk melakukan koneksi dengan *server* pada baris 11 dan 12.Baris 14 dan 15 merupakan *code* untuk menampilkan notifikasi yang dikirimkan oleh *server*. Baris 17 dan 18 berisi variabel yang digunakan pada baris 19 sampai 28. Baris 19 sampai baris 28 merupakan *code* untuk menghitung *delay* menggunakan *timestamp*. Setelah proses perhitungan *delay* selesai, notifikasi dan *delay* akan ditampilkan pada halaman antar-muka.

Untuk mengakses halaman antar-muka *client*, pada *address bar browser client* diketikkan alamat IP dan port dari *server* sesuai dengan alamat perangkat laptop pada jaringan yang terhubung. Misalnya: 10.0.0.1:5000. Setelah diakses maka *push service* akan mengirimkan referensi halaman yang ditunjuk kepada *client* berdasarkan alamat IP dan port. Halaman situs untuk *client* Websocketdapat dilihat pada gambar 4.1.



Gambar 4.1 Antarmuka *Client* Protokol Websocket

4.2.2 Implementasi Client SSE

Seperti halnya implementasi *client* pengiriman notifikasi menggunakan Websocket, implementasi *client* pengiriman notifikasi menggunakan SSE juga diterapkan pada *browser* dengan menggunakan Javascript. Tabel 4.7 adalah *source code* yang diimplementasikan pada sisi *client* dengan menggunakan Javascript pada pengiriman notifikasi menggunakan protokol SSE.

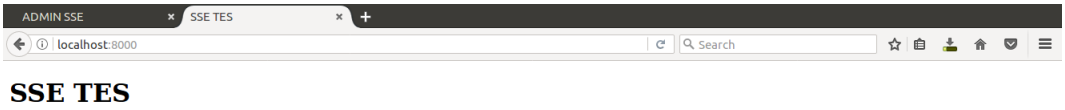
Tabel 4.7 *Source Code* Halaman Antar-Muka SSE

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <title>SSE TES</title>
5.  </head>
6.  <body>
7.  <h1>SSE TES</h1>
8.        <div id="list"></div>
9.
10.
11. <script>
12.     var source = new EventSource("{{ url_for('SSE.stream') }}");
13.     var count = 0;
14.     var sum = 0;
15.     var n = 0; //variabel buat hitung
16.     source.addEventListener('greeting', function(event) {
17.        var data = JSON.parse(event.data);
```

```
18.         var list = document.createElement("LI");
19.         var sec = new Date().getTime() / 1000;
20.         var delay = sec - data.time;
21.         sum = sum + Math.abs(delay); //nilai delay diabsolut
22.         n = n + 1;
23.         var avg = sum / n;
24.         var textnode = document.createTextNode(data.message +' data
25. ke - ' + n +' delay = ' + delay + ' avg = ' + avg);
26.         list.appendChild(textnode);
27.         document.getElementById("list").appendChild(list);
28.         var title = document.title;
29.         count++;
30.                 var newTitle = '(' + count +')' + 'SSE Test';
31.                 document.title = newTitle;
32.         var docBody = document.getElementById('site body');
33.         docBody.onload = newUpdate;
34.     }, false);
35.     source.addEventListener('error', function(event) {
36.     }, false);
37.
38.
39. </script>
40. </body>
41. </html>
```

Source code pada tabel 4.7 akan menampilkan halaman antar-muka *client* yang berupa halaman situs. Di dalamnya terdapat *code* yang berfungsi untuk melakukan koneksi dengan *server* pada baris 12. Baris 22 sampai 28 merupakan *code* untuk menghitung *delay* dari notifikasi yang dikirimkan dengan menggunakan *timestamp*. Baris ke 29 merupakan *code* untuk menampilkan notifikasi dan hasil perhitungan *delay* yang diperoleh.

Untuk mengakses halaman antar-muka *client*, pada *address bar browser client* diketikkan alamat IP dan port dari *server* sesuai dengan alamat perangkat laptop pada jaringan yang terhubung. Akan tetapi alamat port diganti menjadi 8000. Misalnya: 192.168.43.110:8000. Setelah diakses maka halaman yang akan ditampilkan dapat dilihat pada gambar 4.2.



Gambar 4.2 Antarmuka *Client* Protokol SSE

4.3 Implementasi Tools Pengujian

Implementasi *tools pengujian* pada penelitian ini dibagi menjadi 2 yaitu menggunakan modul *timestamp* untuk pengujian mencari *delay* dan mencari seberapa banyak penggunaan CPU yang dipakai oleh masing-masing *server* ketika mengirimkan notifikasi menggunakan modul *psutil* berdasarkan skenario pengujian yang dilakukan.

Untuk pengujian mencari *delay*, pada *file push service* kedua protokol dimasukkan modul *time* dari python untuk menampilkan *delay* menggunakan *timestamp* pada sisi *client*. Tabel 4.8 berisi *source code* untuk menampilkan notifikasi dan hasil perhitungan *delay*.

Tabel 4.8 *Source Code* untuk menampilkan *delay*Websocket

```
1  emit('my response', {'data': 'Message: '+a, 'count': i, 'time': time.time()}, broadcast=True)
```

Untuk menampilkan *delay* pada *client* maka dimasukkan fungsi operasi untuk menghitung *delay* berdasarkan waktu notifikasi diterima *client* dikurangi waktu notifikasi dikirim server. Tabel 4.9 berisi *source code* untuk menampilkan *delay* dan rata-rata *delay* pengiriman notifikasi pada *server* Websocket.

Tabel 4.9 *Source Code* untuk menghitung *delayWebsocket*

```
1  var sec = new Date().getTime() / 1000;
2              var delay = sec - msg.time;
3              sum = sum + Math.abs(delay);
4              console.log(sum);
5              n = n + 1;
```

```
6               var avg = sum / n;
7                  $('#log').append(msg.data + ' data ke - ' + n +'
delay = ' + delay + ' avg = ' + avg +'</br>');
8
9             });
```

Baris 1 merupakan operasi hitung untuk mengubah *delay* yang sebelumnya dalam satuan *millisecond* menjadi *second*. Baris 2 merupakan operasi hitung *delay* dengan cara waktu notifikasi diterima pada variabel *sec* dikurangi waktu notifikasi dikirim pada variabel *msg*.*time()*. Baris 3 merupakan operasi penjumlahan *delay* notifikasi sesudah dengan *delay* notifikasi sebelumnya. Baris 6 merupakan operasi hitung rata-rata *delay* dari perhitungan yang diperoleh dengan membagi nilai rata-rata total dengan *delay* yang diperoleh pada setiap notifikasi yang diterima.

Tabel 4.10 berisi *source code* untuk menampilkam pengiriman notifikasidan hasil perhitungan *delay*menggunakan protokol SSE.

Tabel 4.10 *Source Code* untuk menampilkan *delay* SSE

```
1  SSE.publish({"message": text, "time": time.time()}, type='greeting')
```

Source code operasi perhitungan *delay* dan rata-rata *delay* pada *client* pengiriman notifikasi menggunakan SSE dapat dilihat pada tabel 4.11.

Tabel 4.11 *Source Code* untuk menghitung *delay* SSE

```
1  var sec = new Date().getTime() / 1000;
2        var delay = sec - data.time;
3        sum = sum + Math.abs(delay);
4        n = n + 1;
5        var avg = sum / n;
6        var textnode = document.createTextNode(data.message +' data ke
- ' + n +' delay = ' + delay + ' avg = ' + avg);
7
```

Baris 1 merupakan operasi hitung untuk mengubah *delay* yang sebelumnya dalam satuan *millisecond* menjadi *second*. Baris 2 merupakan operasi hitung *delay* dengan cara waktu notifikasi diterima pada variabel *sec* dikurangi waktu notifikasi dikirim pada variabel *data*.*time()*. Baris 3 merupakan operasi penjumlahan *delay* notifikasi sesudah dengan *delay* notifikasi sebelumnya Baris 5 merupakan operasi hitung rata-rata *delay* dari perhitungan yang diperoleh pada baris 2. Baris 6 berfungsi untuk menampilkan notifikasi, *delay* serta rata-rata *delay* yang diterima *client*.

Implementasi pengujian berikutnya adalah mencari nilai *resource* yang terpakai, dalam hal ini SSE, ketika *server* mengirimkan notifikasi ke *client* berdasarkan skenario yang telah ditentukan dengan menggunakan modul *psutil*.

Untuk menampilkan nilai CPU yang digunakan *server* pada terminal Linux maka akan dimasukkan *source code* untuk memanggil modul *psutil* pada *file server push* masing-masing protokol. Tabel 4.12 adalah *source code* yang dimasukkan pada *file server push* protokol Websocket.

Tabel 4.12 *Source Code* Psutil pada Websocket

```
1  def server_check():
2         while 1:
3              cpu = psutil.cpu_percent()
4              mem = psutil.virtual_memory().percent
5              print "Debug CPU = " +str(cpu)+ " MEM = " +str(mem)
6              time.sleep(2)
7  t = Thread(target = server_check, args = ())
8  t.start()
9  #thread.start_new_thread(server_check,())
```

Baris 3 dan 4 berfungsi untuk menampilkan CPU dan *memory* yang digunakan dalam persen. Baris 5 berfungsi untuk menampilkan nilai CPU dan *memory* yang digunakan oleh *server* pada *console* Ubuntu. Baris 6 berfungsi untuk mencetak *code* pada baris 5 setiap 2 detik.

Sedangkan *file server push* protokol SSE menggunakan *source code* pada tabel 4.13.

Tabel 4.13 *Source Code* Psutilpada SSE

```
1  def server_check():
2         while 1:
3              cpu = psutil.cpu_percent()
4              mem = psutil.virtual_memory().percent
5              print "Debug CPU = " +str(cpu)+ " MEM = " +str(mem)
6              time.sleep(2)
7  thread.start_new_thread(server_check,())
```

Baris 3 dan 4 berfungsi untuk menampilkan CPU dan *memory* dalam persen. Baris 5 berfungsi untuk menampilkan nilai CPU dan *memory* yang digunakan oleh *server* pada *console* Ubuntu. Baris 6 berfungsi untuk mencetak *code* pada baris 5 setiap 2 detik.