repository.ub.ac.id

PERBANDINGAN WEBSOCKET PADA KOMUNIKASI APLIKASI PERPESANAN BERBASIS ANDROID MENGGUNAKAN *LIBRARY* ANDROIDASYNC, JAVA WEBSOCKET, DAN NV WEBSOCKET CLIENT

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Muhammad Harisuddin Thohir
NIM: 125150200111103

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

UNIVERSITAS BRAWIJAYA

PENGESAHAN

PERBANDINGAN WEBSOCKET PADA KOMUNIKASI APLIKASI PERPESANAN BERBASIS ANDROID MENGGUNAKAN *LIBRARY* ANDROIDASYNC, JAVA WEBSOCKET, DAN NV WEBSOCKET CLIENT

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Muhammad Harisuddin Thohir
NIM: 125150200111103

Skripsi ini telah diuji dan dinyatakan lulus pada
18 Mei 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Aryo Pinandito, S.T, M.MT	Lutfi Fanani, S.Kom., M.T., M.Sc.
NIP: 19830519 2014041 001	NIK: 201607 8902171 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 2003121 001

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 18 Mei 2018

Muhammad Harisuddin Thohir
NIM: 125150200111103

UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Puji syukur atas kehadirat Allah S.W.T., karena atas segala rahmat dan hindayah-Nya, penulis dapat menyelesaikan tugas akhir yang berjudul "Perbandingan Websocket Pada Komunikasi Aplikasi Perpesanan Berbasis Android Menggunakan *Library* Androidasync, Java Websocekt, Dan Nv Websocket Client".

Dalam pelaksanaannya penulis mendapatkan bantuan dari berbagai pihak. Dalam kesempatan ini, penulis mengucapkan banyak terima kasih kepada:

1. Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D, selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
2. Tri Astoto Kurniawan, S.T, M.T, Ph.D, selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
3. Agus Wahyu Widodo, S.T, M.Cs, selaku Ketua Program Studi Teknik Informatika.
4. Aryo Pinandito, S.T, M.MT, selaku Dosen Pembimbing pertama yang telah membimbing dan memberikan banyak saran kepada penulis sehingga dapat menyelesaikan skripsi ini.
5. Lutfi Fanani, S.Kom., M.T., M.Sc., selaku Dosen Pembimbing kedua yang telah membimbing dan memberikan banyak saran kepada penulis sehingga dapat menyelesaikan skripsi ini.
6. Kedua Orang Tua Penulis serta keluarga besar atas segala doa, nasihat, dukungan baik moril maupun materiil dalam melancarkan skripsi ini.
7. Teman-teman atas bantuan, dukungan, motivasi dan berbagi informasi serta pihak-pihak lain yang tidak dapat penulis sebut satu persatu, yang turut membantu peyelesain skripsi baik secara langsung maupun tidak langsung.

Dengan segala kerendahan hati, penulis menyadari bahwa skripsi ini masih memiliki banyak kekurangan. Oleh karena itu kritik dan saran yang bersifat konstruktif sangat dibutuhkan sebagai pedoman untuk menyempurnakan skripsi ini agar lebih baik. Penulis berharap semoga skripsi ini dapat bermanfaat bagi diri sendiri maupun bagi semua pihak.

Malang, 18 Mei 2018

Penulis

me@harisuddin.com

repository.ub.ac.id

ABSTRAK

Semakin banyaknya aplikasi *mobile* yang menambahkan fitur perpesanan atau bahkan menjadikannya sebagai fitur utama menunjukkan bahwa fitur ini merupakan salah satu fitur yang sangat penting. Salah satu protokol yang memungkinkan untuk digunakan sebagai protokol komunikasi dua arah yaitu Websocket. Dalam mengembangkan aplikasi pada *platform* Android untuk mengimplementasikan komunikasi data menggunakan Websocket terdapat berbagai *library* Websocket *client* yang dapat digunakan. *Platform* yang dipilih adalah Android karena dari segi pengguna jumlahnya lebih banyak, yaitu sekitar 86,8%. Untuk dapat memilih *library* yang tepat maka diperlukan adanya uji coba terhadap *library* tersebut. Untuk *library* yang akan diuji yaitu AndroidAsync, Java Websocket, dan Nv Websocket Client. Parameter yang akan diuji ada dua yaitu penggunaan daya dan kecepatan pengiriman data. Pengujian dilakukan dengan cara terlebih dahulu menentukan *library* yang akan diuji kemudian ukuran pesan yang akan dikirimkan, setelah itu menjalankan aplikasi Trepn Profiler kemudian menjalankan aplikasi perpesanan yang akan diuji. Ketika aplikasi berjalan maka secara otomatis akan mengirimkan pesan ke pengguna lain sebanyak 50 kali, proses ini diulang sebanyak 30 kali pada masing-masing ukuran pesan dan *library* yang diuji. Total pengujian yang dilakukan sebanyak 450 kali atau sebanyak 22.500 pesan yang dikirimkan. Hasil pengujian menunjukkan bahwa semakin besar pesan yang dikirimkan maka kecepatannya juga akan semakin tinggi. Sedangkan untuk hasil kecepatan rata-rata pada masing-masing *library* menunjukkan bahwa Nv Websocket Client lebih unggul dalam hal kecepatan pengiriman data. Di sisi lain dalam hal konsumsi daya hasil rata-rata pada masing-masing *library* menunjukkan bahwa AndroidAsync merupakan *library* yang lebih hemat dalam penggunaan daya jika dibandingkan dengan *library* lain.

Kata kunci: Android, Daya, Kecepatan, Performansi, Websocket

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

The growing number of mobile apps that add messaging features or even make it a main feature indicate that this feature is one of the most important features. One of the protocols that allows to be used as a two-way communication protocol is Websocket. In developing applications on the Android platform to implement data communications using Websocket there are various Websocket client libraries that can be used. The chosen platform is Android because in terms of users the number is more than other mobile platform, which is about 86.8%. To choose the right library then required some testing of each libraries. For the libraries to be tested are AndroidAsync, Java Websocket, and Nv Websocket Client. Parameters to be tested there are 2 that is power consumtion and speed of data transmission. Testing is done by first determining the library to be tested then the size of the message to be sent, after that run Trepn Profiler then run the messaging application to be tested. When the application runs it will automatically send a message to another user 50 times, this process is repeated 30 times on each size of the message and the library being tested. Total testing conducted as much as 450 times or as many as 22,500 messages sent. Test results show that the larger the message is sent then the speed will also be faster. As for the results of the average speed in each libraries shows that Nv Websocket Client is superior in terms of data transmission speed. On the other hand in terms of average power consumption results in each libraries shows that AndroidAsync is a library that is more efficient in the use of power when compared with other libraries.

Keywords: Android, Performace, Power, Speed, Websocket

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Menurut data *Top Free in Android Apps* yang diperoleh dari toko aplikasi Android (Google Play) yang diakses melalui jaringan internet Indonesia menunjukkan 6 dari 10 aplikasi teratas dari Google Play adalah aplikasi yang memiliki fitur perpesanan dan 4 dari 6 aplikasi tersebut merupakan aplikasi yang fitur utamanya adalah perpesanan, 10 aplikasi tersebut yaitu WhatsApp Messenger, BBM, Facebook Messenger, Instagram, SHAREit, Facebook, Facebook Lite, UC Browser, Line, dan UC News (Google Play, 2017). Bahkan aplikasi taksi *online* Grab juga mengadopsi fitur perpesanan untuk memudahkan penumpang dan *driver* dalam berkomunikasi (Pratama, 2016).

WhatsApp merupakan aplikasi perpesanan paling populer di seluruh dunia, bahkan dari seluruh pengguna *instant messaging* 58% diantaranya menggunakan WhatsApp (Meola, 2016). Aplikasi perpesanan ini paling banyak digunakan di Indonesia alasan utamanya adalah ringan. Selain itu aplikasi ini sederhana dan mudah untuk digunakan (Techinasia, 2015). Untuk menunjang komunikasi antar penggunanya WhatsApp menggunakan protokol yang mendukung untuk melakukan *real time chatting* yaitu XMPP atau lebih tepatnya menggunakan Ejabberd, yang merupakan XMPP (Jabber) *server* yang dikembangkan dengan bahasa pemrograman Erlang selain itu juga bersifat *open source*, tetapi tidak sebatas itu WhatsApp memodifikasinya untuk mengurangi ukuran pesan yang dikirimkan (High Scalability, 2014).

Januari 1999 Jeremie Miller mengumumkan keberadaan sebuah teknologi terbuka untuk *instant messaging* yang diberi nama Jabber atau yang sekarang lebih dikenal dengan nama Extensible Messaging and Presence Protocol (XMPP), teknologi ini memungkinkan untuk melakukan *real time* XML *streaming,* Sekarang ini XMPP banyak digunakan untuk kebutuhan *real time communication* baik digunakan untuk aplikasi Instant Messaging seperti Whatsapp, Nimbuzz, dan Kontalk. Bahkan digunakan untuk teknologi *push notification* oleh Apple, Google, dan Catpush (xmpp.org). XMPP pada awalnya menggunakan metode *polling* untuk pengiriman dan menerimaan pesan melalui protokol HTTP (XMPP, 2009), yang kemudian digantikan dengan metode yang lebih efisien yaitu menggunakan metode *long polling* dengan nama Bidirectional-streams Over Synchronous HTTP (BOSH) yang digunakan mulai tahun 2004 (XMPP, 2016). Metode *long polling* ini mengalami *transport overhead* yang tinggi jika dibandingkan menggunakan *native* TCP, dengan adanya Websocket yang merupakan sebuah protokol yang dapat melakukan komunikasi dua arah antara *client* dan *server* sehingga memiliki performa lebih baik jika dibandingkan dengan BOSH (Ed dkk., 2014).

Websocket merupakan sebuah protokol yang berjalan di atas protokol TCP yang memungkinkan untuk menjalankan komunikasi dua arah antar *client* untuk mengirimkan pesan yang tidak terpercaya dalam sebuah *environment* yang

terkontrol ke *remote host* yang telah ikut serta dalam komunikasi (jaringan) tersebut (Fette, 2011). Whatsapp sendiri menggunakan ejabberd untuk melakukan pertukaran pesan (High Scalability, 2014), yang mana ejabberd sendiri juga telah menggunakan Websocket untuk protokol *transport*-nya yang lebih efisien dan *scalable* jika dibandingkan dengan BOSH (Ejabberd Docs). Jadi dari sini dapat disimpulkan bahwa untuk melakukan pertukaran pesan Whatsapp menggunakan Websocket untuk protokol transportnya. Inilah yang melatarbelakangi penelitian ini untuk memberikan informasi perbandingan *library* Websocket *client* pada Android, yang mana android saat ini masih menguasai pasar dengan persentase 86,8%, sedangkan untuk *library* yang dipilih untuk diuji yaitu Nv Websocket Client, AndroidAsync, dan Java Websocket yang mana ketiganya masih aktif dikembangkan oleh pengembangnya, diimplementasikan dalam aplikasi perpesanan karena merupakan salah satu aplikasi yang membutuhkan komunikasi secara *real time*. Perangkat *mobile* memiliki kapasitas daya yang terbatas, situasi ini dapat membatasi kemampuan untuk menikmati penggunaan perangkat secara penuh (Bedregal, dkk., 2013).

1.2 Rumusan Masalah

Penelitian ini merumuskan beberapa permasalahan yang akan dibahas meliputi:

1. Bagaimana hasil perbandingan konsumsi daya dan kecepatan pengiriman data pada aplikasi perpesanan dengan Websocket antara *library* AndroidAsync, Java Websocket, dan Nv Websocket Client pada Android?
2. *Library* manakah yang paling baik untuk aplikasi perpesanan?

1.3 Tujuan

1. Menganailisis hasil perbandingan kecepatan pengiriman data dan konsumsi daya pada aplikasi perpesanan dengan Websocket antara *library* AndroidAsync, Java Websocket, dan Nv Websocket Client berbasis Android.
2. Mengetahui *library* mana yang paling baik untuk aplikasi perpesanan.

1.4 Manfaat

Adapun manfaat yang diharapkan dalam dokumentasi penelitian ini adalah sebagai berikut:

1. Memberikan informasi kepada para pengembang aplikasi Android tentang konsumsi daya dan kecepatan pengiriman data *library* Websocket pada aplikasi perpesanan berbasis Android.

1.5 Batasan Masalah

Penelitian yang dilakukan memiliki batasan–batasan masalah sebagai berikut:

1. Studi kasus yang digunakan adalah aplikasi perpesanan berbasis Android yang menggunakan protokol Websocket untuk saling berkomunikasi

2. *Platform* yang digunakan dalam penelitian ini dibatasi hanya menggunakan Android sebagai Websocket *client*
3. Pendekatan pengembangan aplikasi menggunakan *native mobile application development*
4. Pengujian kinerja difokuskan kepada ukuran data yang dikirimkan
5. Aspek keamanan dalam pertukaran data tidak diterapkan dalam penelitian ini
6. Pengujian aplikasi dibatasi hanya pada proses pengiriman pesannya saja.

1.6 Sistematika Pembahasan

Sistematika dalam penulisan dokumentasi pada penelitian ini dapat diuraikan sebagai berikut:

BAB I : Pendahuluan

Bab ini menguraikan tentang latar belakang, rumusan masalah, batasan, tujuan, dan manfaat penelitian dalam Perbandingan Websocket Pada Komunikasi Aplikasi Perpesanan Berbasis Android Menggunakan *Library* AndroidAsync, Java Websocekt, dan Nv Websocket Client.

BAB II : Tinjauan Pustaka

Bab ini menguraikan mengenai kajian pustaka dan teori-teori pendukung yang akan digunakan dalam penelitian Perbandingan Websocket Pada Komunikasi Aplikasi Perpesanan Berbasis Android Menggunakan *Library* AndroidAsync, Java Websocekt, dan Nv Websocket Client. Teori–teori yang berhubungan terdiri dari Websocket, Pendekatan Pengembangan Aplikasi *Mobile*, dan Pengujian Perangkat Lunak.

BAB III : Metode Penelitian

Metode penelitian merupakan bagian yang menerangkan mengenai langkah kerja dan metode yang diterapkan dalam Perbandingan Websocket Pada Komunikasi Aplikasi Perpesanan Berbasis Android Menggunakan *Library* AndroidAsync, Java Websocekt, dan Nv Websocket Client.

BAB IV : Analisis Kebutuhan dan Perancangan

Menganalisis kebutuhan dalam Perbandingan Websocket Pada Komunikasi Aplikasi Perpesanan Berbasis Android Menggunakan *Library* AndroidAsync, Java Websocekt, dan Nv Websocket Client serta melakukan perancangan sistem.

BAB V : Implementasi

Menjelaskan tahap–tahap implementasi dalam Perbandingan Websocket Pada Komunikasi Aplikasi Perpesanan Berbasis Android Menggunakan *Library* AndroidAsync, Java Websocekt, Dan Nv Websocket Client.

BAB VI : Pengujian dan Analisis

Memaparkan mengenai tahap–tahap pengujian yang dilakukan terhadap aplikasi dan melakukan analisis terhadap hasil yang diberikan oleh aplikasi.

BAB VII: Penutup

Memberikan kesimpulan dari penelitian yang telah dilakukan dan saran yang dapat digunakan oleh peneliti selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini akan menjelaskan teori-teori dasar untuk menunjang penelitian analisis performansi sebagai berikut:

2.1 Kajian Pustaka

Pada bagian kajian pustaka ini membahas mengenai beberapa penelitian sebelumnya yang dijadikan referensi untuk melakukan penelitian. Beberapa penelitian tersebut antara lain penelitian yang berjudul *"Aplikasi Grupchat Di Android Menggunakan Websocket"* yang dilakukan oleh Fitri Hardianto pada tahun 2015. Penelitian ini mengembangkan sebuah aplikasi android yang mampu melakukan komunikasi/*chat* antar penggunanya secara *public/group* dan penggunanya mampu bertukar pesan dalam bentuk teks dengan pengguna yang lain, aplikasi ini memanfaatkan Websocket sebagai protokol komunikasinya.

Pada penelitian kedua dengan judul *"Pengembangan Push Notification Menggunakan Websocket"* yang dilakukan oleh Andrias Yudianto P. yang dilakukan pada 2017. Penelitian ini mengembangkan sebuah sistem yang mampu mengirimkan *push notification* ke perangkat yang terhubung menggunakan sebuah sistem operasi Ubuntu sebagai *server* dan perangkat android serta browser pada Windows dan Linux sebagai client. Pendekatan yang dilakukan untuk mengembangkan aplikasi Android pada penelitian ini menggunakan pendekatan pengembangan Hybrid dengan menggunakan *framework* Cordova.

Pada penelitian ketiga dengan judul *"Communicating and Displaying Real-Time Data with WebSocket"* yang dilakukan oleh Victoria Pimentel dan Bradford G. Nickerson pada tahun 2012. Penelitian ini membandingkan *latency* antara penggunaan HTTP Polling, HTTP Long Polling dan Webscoket dengan cara mengirimkan 1200 pesan dalam waktu kiri-kira 15 menit, pengujian dilakukan sebanyak tiga kali dengan waktu yang berbeda-beda yaitu pukul 8 pagi (*traffic* tidak sibuk), pukul 1 siang (*traffic* normal) dan pukul 8 malam (*traffic* sibuk). Hasil menunjukkan bahwa *latency* rata-rata dari HTTP Polling secara signifikan lebih tinggi (antara 2,3 dan 4,5 kali lebih tinggi) dari Websocket maupun HTTP Long Polling. Protokol websocket dapat memiliki *latency* yang lebih rendah atau lebih tinggi dari *latency* rata-rata HTTP Long Polling. Untuk jarak yang lebih jauh secara signifikan protokol Websocket memiliki *latency* rata-rata yang lebih rendah (antara 3,8 dan 4,0 kali) dari HTTP Long Polling.

Pada penelitian keempat dengan judul "*Optimizing Energy Consumption per Application in Mobile Devices*" yang dilakukan oleh Jose Carlos Valdivia Bedregal, Eveling Gloria Castro Gutierrez dan Robert E. Arisaca M. pada tahun 2013. Penelitian ini manganalisis konsumsi baterai dari masing-masing aplikasi yang berjalan pada Android, aplikasi yang digunakan untuk melakukan analisis adalah Power Tutor yang mana menyediakan informasi tentang komponen dari telepon seluler yang mengonsumsi energi tertinggi pada setiap aplikasi, kemudian hasilnya dijadikan acuan untuk melakukan analisis. Power Tutor sendiri adalah

aplikasi Android yang menampilkan jumlah energi yang dikonsumsi oleh komponen sistem seperti CPU, *network interface*, *display*, GPS dan aplikasi lainnya. Hasil dari penelitian ini menunjukkan adanya aplikasi yang mengonsumsi baterai secara berlebih, situasi ini dapat ditangani dengan cara mengeluarkan aplikasi tersebut sehingga baterai bisa mengalami penghematan sebesar 10%.

Penelitian kelima dengan judul "*The Comparison Of Impacts To Android Phone Battery Between Polling Data And Pushing Data*" yang dilakukan oleh Pham Cao Dinh dan Sirapat Boonkrong pada tahun 2013. Penelitian ini membandingkan konsumsi energi dari dua teknik yang berbeda untuk mendapatkan data dari server yaitu *Polling* dan *Pushing* data Menggunakan Power Tutor. Untuk mencari teknik mana yang lebih efisien dalam hal konsumsi daya yang digunakan. Hasil dari penelitian ini menunjukkan bahwa aplikasi yang menggunakan teknik *pushing* data mampu menghemat baterai lebih dari aplikasi yang menggunakan teknik *polling*.

2.2 Websocket

Websocket merupakan sebuah *channel* komunikasi *asyncronous event-driven*, *full-duplex* untuk *web application*. Websocket mempunyai kemampuan untuk memberikan *update* secara *real-time* yang sebelumnya menggunakan metode *long polling*. Keuntungan utama menggunakan Websocket adalah mengurangi kebutuhan sumber daya baik di sisi klien maupun *server.* Websocket menggunakan HTTP sebagai mekanisme *transport*, komunikasi tidak seketika berakhir setelah respon diterima oleh *client*, melainkan selama koneksi masih terbuka klien dan *server* dapat salaing mengirim pesan secara *asynchronous* (Lombardi, 2015).

Pada penelitian ini Websocket digunakan sebagai *transport protocol* untuk melakukan pertukaran pesan antara client di mana pesan yang dikirimkan dalam bentuk JSON.

2.3 *Library*

Library yang digunakan untuk koneksi websocket pada Android di antaranya ialah:

2.3.1 Nv Websocket Client

Nv Websocket Client Android merupakan sebuah *open source library* untuk Java/Android yang mengimplementasikan protokol Websocket untuk membuat *native mobile* WebSocket *clients*. Keunggulan library ini diantaranya (Nv Websocket Client, 2017):

- Sesuai dengan RFC6455 (The WebSocket Protocol),
- Mendukung Java SE 1.5+ dan Android,
- Mendukung semua tipe *frame* (*continuation, binary, text, close, ping* dan *pong*),

- Mendukung metode Basic Authentication,
- Mendukung HTTP *proxy,* khususnya *secure* Websocket (wss) melalui *secure proxy* (https),
- Dan juga mendukung RFC7692 (Compression Extensions for WebSocket).

Pada penelitian penggunaan AndroidAsync sebagai Websocket *client* untuk melakukan pertukaran pesan dengan *client* lain dalam bentuk JSON yang berisi teks atau gambar yang di-*encode* ke dalam bentuk Base64. Karena file yang di-*encode* menggunakan Base64 ukurannya akan naik menjadi 137% dari ukuran semula (Cisco, 2014), maka ukuran file yang akan dikirimkan nantinya akan dikirimkan diturunkan dulu menjadi 73% dari ukuran aslinya agar nanti file yang dikirimkan bisa sesuai.

2.3.2 AndroidAsync

AndroidAsync merupakan sebuah *library* protokol jaringan tingkat rendah yang dapat digunakan sebagai Socket *client/server* HTTP *client/server*, Websocket *client/server*, dan Socket.io *client* (AndroidAsync, 2016).

Pada penelitian penggunaan AndroidAsync sebagai Websocket *client* untuk melakukan pertukaran pesan dengan client lain dalam bentuk JSON yang berisi teks atau gambar yang di-*encode* ke dalam bentuk Base64.

2.3.3 Java Websocket

Java Websocket merupakan *library* Websocket *client/server* yang ditulis dengan bahasa pemrograman Java dan diimplementasikan menggunakan kelas ServerSocketChannel dan SocketChannel pada Java. Library ini websocket yang mendukung RFC 6455 (Java-Websocket, 2014).

Pada penelitian penggunaan AndroidAsync sebagai Websocket *client* untuk melakukan pertukaran pesan dengan client lain dalam bentuk JSON yang berisi teks atau gambar yang di-*encode* ke dalam bentuk Base64.

2.4 Javascript Object Notation (JSON)

JavaScript Object Notation (JSON) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman *JavaScript*, Standar *ECMA-262* Edisi ke-3 Desember 1999 (JSON, 2016).

JSON terbuat dari dua struktur (JSON, 2016):

- Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai object, record, struct, dictionary, hash table, keyed list, atau associative array.

- Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai array, vector, list, atau sequence.

Pada penelitian ini JSON digunakan sebagai format data yang dikirimkan baik untuk pengiriman data menggunakan Websocket maupun HTTP.

2.5 Bottle

Bottle merupakan sebuah Web Server Gateway Interface (WSGI) *micro web-framework* untuk python yang cepat, simpel dan ringan. Bottle didistribusikan sebagai satu *file* modul dan tidak memiliki ketergantungan selain standar *library* Python (Bottle, 2017).

Bottle memiliki beberapa fitur, di antaranya:

1. *Routing*

   Request ke pemetaan pemanggilan fungsi dengan menggunakan URL yang bersih dan dinamis.

2. *Template*

   Built-in template engine yang cepat dan *pythonic.* Selain itu juga mendukung *template engine* lain seperti mako, jinja2, dan cheetah.

3. *Utilities*

   Mengakses data, unggah *file*, *cookies, headers,* dan *metadata* lainnya terkait HTTP.

4. *Server*

   Memiliki *built-in* HTTP *development server* dan mendukung *server libray* lain seperti paste, fapws3, bjoern, Google App Engine, cherrypy atau WSGI HTTP *server* lain (Bottle, 2017).

Pada penelitian ini Bottle digunakan sebagai *micro web-framework* untuk menjalankan Web API yang akan melayani request http dari *client*.

2.6 Websocket *Server*

Python Websocket Server merupakan sebuah *library* Websocket yang dapat berjalan baik di Python2 maupun Python3 yang tidak membutuhkan *library* eksternal. Kelebihan dari *library* ini di antaranya mampu berjalan pada Python2 dan Python3, API yang simpel, mendukung *multiple clients*, tidak ada ketergantungan dengan *library* lain. Akan tetapi untuk sekarang *library* ini masih belum mendukung fitur SSL (Pithikos, 2017). Pada penelitian ini Python Websocket Server digunakan sebagai Websocket *server* untuk melayani pertukaran pesan antar klien.

2.7 MongoDB

MongoDB merupakan sebuah basis data yang relatif masih baru yang tidak memiliki konsep tabel, skema, SQL, atau baris. MongoDB tidak memiliki

transaksi, ACID *compliance, joins, foreign keys,* atau hal lain yang ada di database pada umumnya. Singkatnya, MongoDB merupakan sebuah basis data yang sangat berbeda dari basis data yang biasanya digunakan, khususnya berbeda dengan Relational Database Management System (RDBMS) (Hows, 2015). MongoDB merupakan sebuah *document database* dengan skalabilitas dan fleksibilitas dengan *query* dan *indexing* yang dibutuhkan. MongoDB menyimpan data dengan fleksibel, mirip dengan JSON, yang berarti *field* dari masing-masing dokumen dapat berbeda satu dengan yang lainnya dan struktur datanya dapat berubah-ubah (MongoDB, 2017).

Pada penelitian ini MongoDB digunakan sebagai basis data *server* untuk menyimpan data-data pengguna, pesan dan data-data lain yang dibutuhkan, yang diakses baik oleh HTTP *server* ataupun Websocket *server*.

2.8 Realm

Realm Mobile Database dibangun untuk berjalan pada perangkat *mobile*. Tidak seperti basis data tradisional, *objects* pada Realm merupakan *native objects*. *Object* tidak perlu disalin keluar dari basis data, memodifikasinya, kemudian menyimpannya kembali karena selalu bekerja dengan "*live*", *real object.* Jika satu *thread* atau proses memodifikasi sebuah *object*, maka *thread* atau proses lain akan segera diberitahu. Realm Mobile Database bersifat *open source* dan *cross-platform* dengan *library* yang tersedia untuk Android, iOS, Xamarin (.NET) dan React Native (Realm, 2017).

Pada penelitian ini Realm digunakan sebagai basis data pada aplikasi klien untuk menyimpan data-data seperti data pengguna, daftar kontak, autentikasi pengguna, pesan dan juga pengaturan.

2.9 Vagrant

Vagrant merupakan sebuah alat untuk membangun dan mengelola *virtual mechine environment* dalam satu *workflow*. Dengan alur kerja yang mudah digunakan dan fokus pada otomatisasi, Vagrant mengurangi waktu *setup development environment,* dan meningkatkan produktifitas (Vagrant, 2017).

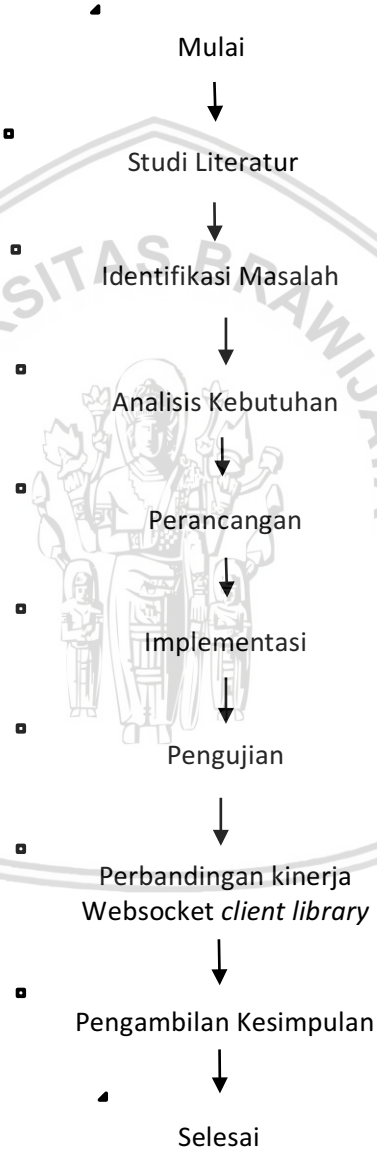
Pada penelitian ini Vagrant digunakan sebagai *environtment* untuk menjalankan server baik Websocket maupun HTTP agar tidak terpengaruh dengan aplikasi yang sedang berjalan pada komputer.

2.10 Trepn Profiler

Trepn™ Profiler merupakan sebuah aplikasi untuk perangkat *mobile* yang digunakan untuk mengukur daya dan kinerja dari aplikasi yang berjalan. Salah satu produk Qualcomm ini mampu berjalan pada sebagian besar perangkat Android, terdapat fitur tambahan apabila perangkat yang diuji menggunakan prosesor Qualcomm® Snapdragon™ (Qualcomm, 2017). Pada penelitian ini Trepn digunakan sebagai *tool* untuk mengetahui jumlah konsumsi daya rata-rata yang digunakan oleh masing-masing *library* Websocket yang akan diuji.

BAB 3 METODOLOGI

Di dalam bab ini menjelaskan tentang tahapan yang akan dilakukan dalam penelitian analisis konsumsi daya dan kecepatan pengiriman data menggunakan Websocket pada Android. Diagram alir pengerjaan tugas akhir ini ditunjukkan dalam Gambar 3.1.



Gambar 3.1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Metode ini dipergunakan untuk memperoleh dasar teori sebagai referensi untuk penyusunan tugas akhir dan pengembangan aplikasi. Dasar teori dan pustaka yang berhubungan dalam penyusunan tugas akhir antara lain:

1. Kajian pustaka dari penelitian sebelumnya sebagai referensi untuk melakukan penelitian.
2. Websocket *server* digunakan sebagai *server* yang mengontrol semua komunikasi yang dilakukan antar klien yang menggunakan protokol Websocket.
3. Android Websocket *client* yang terdiri dari 3 *library* yang akan diuji, bertindak sebagai klien yang terhubung dengan Websocket *server* untuk mentransmisikan pesan melalui protokol Websocket.
4. JSON digunakan sebagai format data yang ditransmisikan melalui protokol Websocket.
5. Web Service digunakan untuk melakukan autentikasi pengguna.
6. Vagrant sebagai media yang mengatur virtualisasi sistem operasi yang digunakan untuk menjalankan Websocket *server*.
7. Trepn Profiler digunakan sebagai alat ukur untuk melihat penggunaan daya ketika aplikasi dijalankan.

3.2 Identifikasi Masalah

Dimulai dengan melakukan identifikasi permasalahan yang ada yaitu bagaimana cara memperoleh nilai konsumsi daya yang dihabiskan oleh aplikasi *chat* dengan menggunakan Websocket dan juga bagaimana cara memperoleh kecepatan pengiriman pesan yang dikirimkan melalui protokol Websocket. Dari permasalahan tersebut maka diperlukan suatu *tool* baik berupa aplikasi ataupun metode tersendiri yang dimasukkan ke dalam aplikasi untuk memperoleh data berupa konsumsi daya yang digunakan dan juga kecepatan pengiriman pesan.

3.3 Analisis Kebutuhan

Supaya aplikasi dapat saling bertukar pesan menggunakan protokol Websocket maka beberapa hal yang diperlukan di antaranya Websocket *server* untuk mentransmisikan pesan melalui protokol Websocket serta REST API untuk keperluan autentikasi yang berjalan pada suatu *server* tersendiri atau dalam mesin virtual agar sumber daya yang digunakan tidak terganggu oleh aplikasi lain yang berjalan. Kemudian untuk menyimpan data pesan yang telah dikirimkan maka memerlukan basis data untuk menyimpannya baik di pada *server* maupun klien. Selain itu juga diperlukan aplikasi *chat* yang mengimplementasikan protokol Websocket sebagai media komunikasinya.

3.4 Perancangan

Tahap perancangan penggunaan dan pengujian Websocket *client library* pada Android di antaranya terdapat beberapa komponen perancangan yaitu perancangan basis data, baik di sisi klien maupun *server* sebagai media penyimpanan data berupa pesan atau gambar yang ditransmisikan melalui protokol Websocket, selain itu juga untuk menyimpan data pengguna, selanjutnya perancangan Web API yang menggunakan protokol HTTP untuk melakukan validasi pengguna yang masuk dan juga untuk mendapatkan token yang digunakan untuk berkomunikasi menggunakan websocket, selanjutnya agar Websocket *client* dapat berkomunikasi maka diperlukan minimal sebuah *host* (server) yang menjalankan *service* Websocket server, berikutnya agar komunikasi bisa berjalan maka aplikasi klien harus mengimplementasikan tiga Websocket *client library* yang akan dibandingkan, dan yang terakhir agar desain antarmuka yang dihasilkan sesuai dengan kebutuhan maka perlu dilakukan perancangan antarmuka yang sesuai dengan kebutuhan fungsional aplikasi.

3.4.1 Perancangan Basis data

Perancangan basis data bertujuan untuk merancang susunan data yang akan digunakan dalam pembuatan aplikasi baik di sisi *client* ataupun *server*. Hasil perancangan tersebut akan diimplementasikan menggunakan MongoDB pada sisi *server*, sedangkan pada sisi *client* (Android) basis data yang digunakan adalah Realm.

3.4.2 Perancangan Web API

Setelah melakukan perancangan basis data maka dilakukan perancangan Web API. Dilakukan perancangan Web API agar *client* dan *server* dapat berkomunikasi seperti *register, login, update profile* dan lainnya di luar pengiriman pesan, karena untuk pengiriman pesan akan ditangani sendiri oleh Websocket. Perancangan dimulai dengan menentukan *request* apa saja yang ingin diperlukan kemudian parameter apa saja yang akan digunakan ketika mengirimkan *request* dan menerima *response*. Setelah itu mengimplementasikannya dengan membuat *url pattern* dari masing-request yang akan mengarahkannya ke sebuah fungsi tertentu sehingga pengguna mendapatkan *response* yang sesuai dengan *request* yang dikirimkannya.

3.4.3 Perancangan Webscoket *Server*

Setelah melakukan perancangan Web API maka yang selanjutnya dilakukan adalah melakukan perancangan Websocket server agar dapat meneruskan pesan dari satu *client* ke *client* lain yang dituju. Perancangan ini dimulai dengan merancang keamanan mulai dari autentifikasi saat pengguna pertama kali terhubung dengan Websocket *server*, kemudian merancang enkripsi yang akan digunakan untuk melakukan pengiriman pesan, setelah itu merancang format pengiriman pesan yang akan digunakan.

3.4.4 Perancangan Penggunaan *Library*

Perancangan penggunaan *library* bertujuan untuk merancang tahapan yang digunakan pada implementasi sistem. Ketika aplikasi berjalan maka *library* akan terhubung ke Websocket *server*, jika koneksi sudah terautentifikasi maka aplikasi siap menerima dan mengirim pesan.

Masing-masing *library* akan dijalankan sebagai *service*. Ketika pengguna mengirimkan pesan maka *service* akan menerima pesan yang kemudian dikirimkan ke Websocket server untuk diteruskan kepada pengguna lain yang dituju. Proses pengiriman pesan juga disertai dengan menyimpan pesan ke dalam basis data. Pesan akan diterima oleh penerima pesan jika *service* dari *library* Websocket *client* berjalan, kemudian pesan tersebut disimpan ke dalam basis data. Pada halaman Chat menggunakan sebuah Listener untuk mengetahui perubahan pada basis data, untuk itu secara otomatis pesan akan tampil pada halaman Chat. Selama proses pengiriman dan penerimaan data dilakukan pengamatan jumlah konsumsi daya yang digunakan dan juga kecepatan pengiriman pesan.

3.4.5 Perancangan Antarmuka

Perancangan antarmuka bertujuan untuk merancang antarmuka yang dapat memenuhi kebutuhan fungsional aplikasi. Untuk membantu dalam penelitian ini, maka dibutuhkan antarmuka aplikasi yang dapat menampilkan kecepatan pengiriman pesan saja karena untuk mengamati konsumsi daya menggunakan aplikasi lain yaitu Power Tutor.

3.5 Implementasi

Berdasarkan perancangan yang telah dijelaskan sebelumnya maka implementasi dilakukan dengan membuat purwarupa berupa aplikasi *native* Android. Pendekatan aplikasi *native* dilakukan untuk mendapatkan kinerja aplikasi yang optimal.

Library yang akan digunakan pada sistem ini yaitu Nv Websocket Client, AndroidAsync, dan Java Websocket. Untuk menunjang penelitian ini, peneliti menggunakan *smartphone* dengan sistem operasi Android untuk menjalankan aplikasinya.

3.6 Pengujian

Proses pengujian konsumsi daya dilakukan dengan cara menjalankan aplikasi Power Tutor kemudian menjalankan aplikasi kemudian melakukan pengiriman pesan dengan ukuran (Byte) tertentu selama waktu yang ditentukan secara berulang-ulang. Pengujian ini dilakukan sebanyak 30 kali untuk mendapatkan variasi nilai. Data akan ditampilkan dalam bentuk tabel maupun grafik.

Sedangkan untuk pengujian kecepatan dilakukan bersamaan dengan pengujian konsumsi daya, pada setiap pesan yang dikirmkan pengirim akan mendapatkan pesan dari penerima bahwa pesan yang dikirim telah sampai,

kemudian pengirim menghitung selisih antara waktu pengiriman dan waktu pesan terkirim sehingga diperoleh waktu yang diperlukan untuk pengiriman pesan. Dari sini kemudian bisa dihitung kecepatan pengiriman datanya dengan cara membagi ukuran data yang dikirimkan dengan waktu yang diperlukan untuk pengiriman pesan.

3.7 Perbandingan kinerja Websocket *client library*

Untuk membandingkan masing-masing *library* maka pada setiap pengujian dilakukan dengan cara yang sama dengan menggunakan masing-masing *library* secara bergantian. Dari sini akan didapatkan hasil jumlah konsumsi daya dan kecepatan rata-rata dari masing-masing *library* untuk dibandingkan satu sama lain.

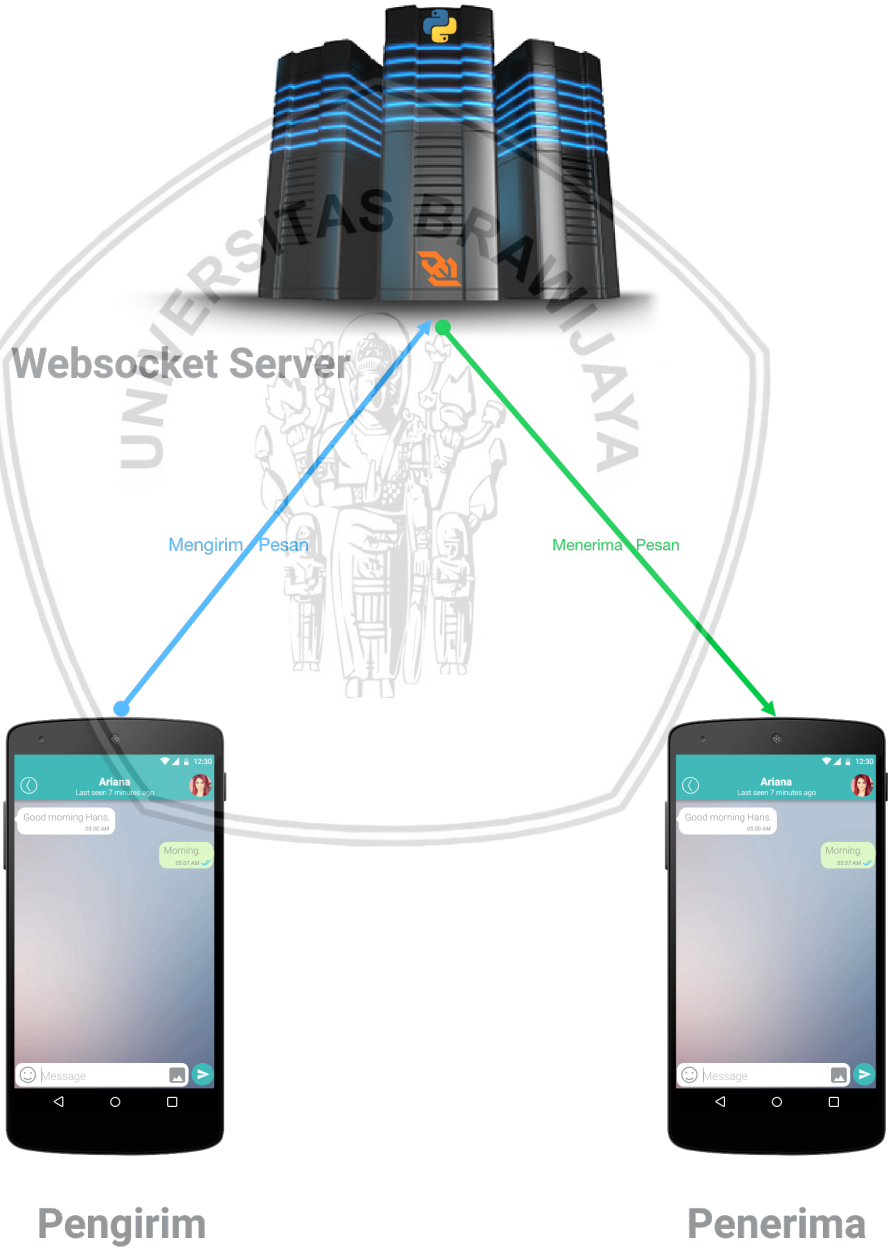
3.8 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua tahapan selesai dilakukan. Mulai dari tahapan studi literatur, identifikasi masalah, perancangan, implementasi, pengujian, hingga analisis. Kesimpulan dibuat untuk menjawab rumusan masalah yaitu bagaimana perbandingan konsumsi daya dan kecepatan pengiriman data pada aplikasi perpesanan dengan Websocket antara *library* AndroidAsync, Java Websocket, dan Nv Websocket Client berbasis Android dan juga untuk mengetahui manakah *library* yang paling baik untuk aplikasi pengiriman pesan. Selain kesimpulan, saran juga dibuat untuk memberikan pertimbangan maupun acuan terhadap penelitian selanjutnya khususnya pada *platform* Android.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

4.1 Deskripsi Umum Sistem

Prototipe analisis perbandingan penggunaan *library* Websocket ini merupakan aplikasi *chat* yang memanfaatkan Websocket sebagai protokol komunikasinya yang menggunakan beberapa *library* klien untuk dibandingkan satu sama lain. Pada aplikasi tersebut terdapat menu pengaturan yang dapat digunakan untuk mengganti *library* yang digunakan untuk berkomunikasi. Pertukaran pesan dapat digambarkan dalam Gambar 4.1.



Gambar 4.1 Skenario pengiriman pesan

1. Pengirim yang sudah terhubung ke *server* Websocket dapat mengirimkan pesan ke pengguna lain (Penerima). Pesan dikirimkan ke *server* yang sudah terhubung, pesan tersebut dikirimkan dalam format JSON, paramater pesan yang dikirimkan berupa id, type, message, time, from dan to.
2. *Server* menerima pesan dari pengirim pesan kemudian mengirimkannya kepada Penerima.
3. Setelah Penerima menerima pesan, maka Penerima mengirimkan pesan ke Pengirim pesan (sebelumnya) bahwa pesan dengan id yang telah diterima oleh Penerima telah diterima, pesan ini dikirimkan ke server dan oleh server dikirimkan ke Pengirim (sebelumnya). Dari skenario ke-3 ini maka akan diperoleh waktu penerimaan pesan oleh Penerima pesan, sehingga bisa diperoleh berapa lama waktu yang diperlukan untuk mengirimkan pesan tersebut.

4.2 Kebutuhan Sistem

Bertujuan untuk mengetahui kebutuhan yang diperlukan dalam membangun sistem ini. Kebutuhan sistem pada penelitian ini terbagi menjadi dua yaitu kebutuhan fungsional dan non fungsional.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional berisi tentang proses-proses apa saja yang harus dapat dilakukan oleh sistem. Kebutuhan fungsional pada sistem ini terdiri dari:

1. Sistem dapat mengirimkan pesan berupa teks atau gambar melalui protokol Websocket serta mengetahui kecepatan pengiriman pesan
2. Sistem dapat menerima pesan berupa teks atau gambar melalui protokol Websocket serta mengetahui kecepatan pengiriman pesan.

4.2.2 Kebutuhan Non Fungsional

Kebutuhan non fungsional dari sistem yang akan dibangun yaitu:

1. *Usability*

   Tampilan aplikasi dibuat mirip dengan aplikasi *chat* pada umumnya agar lebih mudah digunakan

2. *Portability*

   Karena sistem yang akan dibuat bertujuan untuk dilakukan pengujian maka sistem hanya mampu berjalan pada jaringan lokal

3. *Reliability*

   Protokol yang digunakan untuk berkomunikasi pada sistem ini menggunakan Websocket untuk itu diharapkan pesan yang dikirimkan akan secepatnya sampai ke penerima. Untuk masalah keamanan sistem hanya

akan menangani masalah keamanan untuk autentikasi pengguna, sedangkan masalah keamanan (enkripsi) pesan yang ditransmisikan tidak diperhatikan

4. *Supportability*

   Aplikasi dapat dijalankan pada sistem operasi Android minimal versi 4.1 Jelly Bean (API 16).

4.3 Analisis Kebutuhan Sistem

Analisis kebutuhan bertujuan untuk menentukan semua kebutuhan yang diperlukan untuk membangun sustem perangkat lunak. Berikut ini dijelaskan tentang Analisis Kebutuhan Sistem yang terdiri dari Identifikasi Aktor, Daftar Kebutuhan Sistem, Diagram *Use Case*, Deskripsi *Use Case* dan Diagram *Activity*.

4.3.1 Identifikasi Aktor

Tahap ini bertujuan untuk melakukan identifikasi terhadap pengguna yang terhubung langsung dengan sistem yang akan dibuat yang terdiri dari aktor pengirim pesan dan aktor penerima pesan. Penjelasan dari masing-masing pengguna dapat dilihat pada Tabel 4.1.

Tabel 4.1 Tabel Identifikasi Aktor

Aktor	Deskripsi
Pengirim Pesan	Pengguna (pengirim pesan) yang sudah terdaftar dalam sistem dapat melakukan *login* ke aplikasi Chat dengan memasukan alamat *email* dan juga *password.* Jika pengguna dapat melakukan *login,* Pengguna dapat mengirimkan pesan ke Pengguna lain yang sudah terdaftar pada sistem.
Penerima Pesan	Pengguna (penerima pesan) yang sudah terdaftar dalam sistem dapat melakukan *login* ke aplikasi Chat dengan memasukan alamat *email* dan juga *password.* Jika pengguna dapat melakukan *login,* Pengguna dapat menerima pesan dari Pengguna lain yang sudah terdaftar pada sistem.

4.3.2 Daftar Kebutuhan Sistem

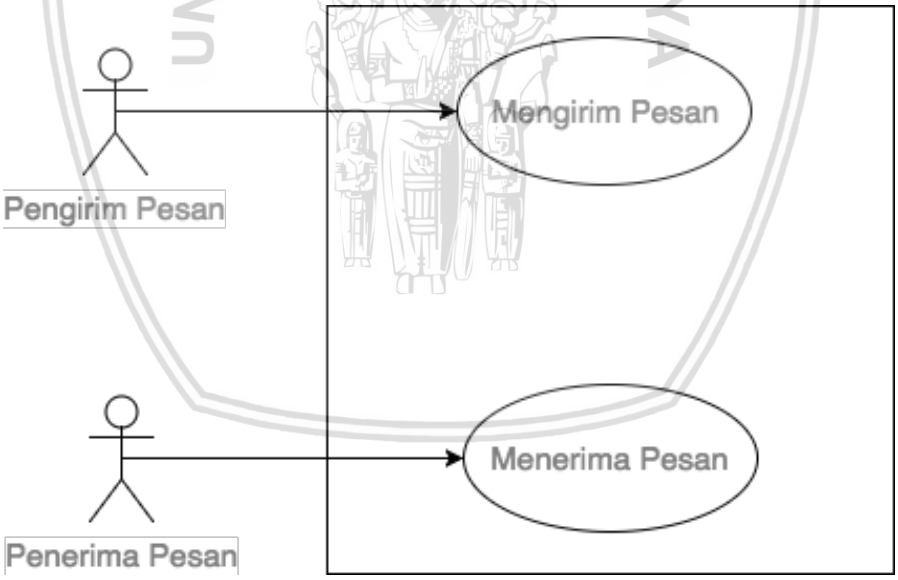
Daftar kebutuhan fungsional sistem terdiri dari dua kebutuhan fungsional yaitu mengirim dan menerima pesan. Kebutuhan sistem dimodelkan ke dalam bentuk *use case,* dan setiap kebutuhan fungsional harus dilakukan pengujian terlebih dahulu sebelum digunakan oleh aktor yang bersangkutan. Kebutuhan fungsional sistem memuat apa saja yang perlu dilakukan oleh masing-masing *use case*. Kebutuhan fungsional sistem ini dapat dilihat pada Tabel 4.2.

Tabel 4.2 Tabel Kebutuhan Fungsional

Identifikasi	Kebutuhan	*Use Case*
FU-01	Sistem mampu mengirimkan pesan berupa teks atau gambar melalui protokol Websocket serta menampilkan kecepatan pengiriman pesan	Mengirimkan Pesan
FU-02	Sistem mampu menerima pesan berupa teks atau gambar serta menampilkan kecepatan pengiriman pesan melalui protokol Websocket	Menerima Pesan

4.3.3 Diagram *Use Case*

Pemodelan diagram *use case* sistem diperoleh dari kebutuhan fungsional pada Tabel 4.2. Diagram *use case* dibuar sesuai dengan kebutuhan fungsional agar semua kebutuhan aktor dapat terpenuhi. Gambar 4.2 merupakan diagram *use case* sistem yang terdiri dua aktor dengan dua *use case*.



Gambar 4.2 Diagram *Use Case* Mengirim Pesan

4.3.4 Deskripsi *Use Case*

Bagian deskripsi *use case* berisi nama *use case*, aktor yang menjalankan *use case*, tujuan dari *use case,* deskripsi mengenai *use case*, kondisi awal yang harus dipenuhi dan kondisi akhir yang diharapkan setelah berjalannya fungsional *use case*.

Tabel 4.3 menjelaskan tentang deskripsi *use case* mengirim pesan. Deskripsi *use case* ini bertujuan untuk mengirimkan pesan dari Pengirim Pesan kepada

Penerima Pesan baik pesan berupa gambar ataupun teks. Kondisi yang harus dipenuhi sebelum *use case* ini dijalankan adalah aktor sudah berhasil melakukan *login*. Sedangkan kondisi akhir yang harus terpenuhi adalah pesan diterima oleh Penerima Pesan yang dituju dan pengirim mendapatkan pemberitahuan melalui tanda centang ganda berwarna biru.

Tabel 4.3 Deskripsi *Use Case* Mengirim Pesan

Identifikasi	
Nama	Mengirim Pesan
Deskripsi	*Use Case* ini menjelaskan aktor dapat mengirim pesan berupa teks atau gambar ke akun lain
Aktor	Pengirim Pesan
Kondisi Awal	Aktor telah berhasil melakukan *login* dan sistem telah menampilkan halaman *chat*, kemudian *library* Websocket klien terhubung dengan Websocket *server*
Kondisi Akhir	Sistem mengirimkan pesan kepada Penerima Pesan sesuai dengan yang dituju oleh Aktor
Normal Flow	*Service library* terhubung dengan Websocket *server,* Aktor pemilih gambar atau menulis pesan yang akan dikirimkan, setelah menekan tombol kirim maka pesan akan dikirimkan menuju Websocket *server* yang kemudian akan diteruskan kepada Penerima Pesan yang dituju
Alternative Flow	Jika aktor sedang tidak terhubung dengan Websocket *server* maka pesan tidak akan dikirimkan, menunggu sampai terhubung dengan Websocket *server*

Tabel 4.4 menjelaskan tentang deskripsi *use case* menerima pesan. Deskripsi *use case* ini bertujuan untuk menerima pesan dari Pengirim Pesan baik pesan berupa gambar ataupun teks. Kondisi yang harus dipenuhi sebelum *use case* ini dijalankan adalah aktor sudah berhasil melakukan *login*. Sedangkan kondisi akhir yang harus terpenuhi adalah pesan diterima oleh aktor kemudian mengirimkan pesan notifikasi bahwa pesan yang dikirimkan oleh Pengirim Pesan sudah diterima.

Tabel 4.4 Deskripsi *Use Case* Menerima Pesan

Identifikasi	
Nama	Menerima Pesan
Deskripsi	*Use Case* ini menjelaskan aktor dapat menerima pesan berupa teks atau gambar dari akun lain
Aktor	Penerima Pesan

Identifikasi	
Kondisi Awal	Pengguna telah berhasil melakukan *login* dan aplikasi dalam kondisi terbuka, kemudian *library* Websocket klien terhubung dengan Websocket *server*
Kondisi Akhir	Sistem menerima pesan dari Pengirin Pesan
Normal Flow	*Service library* terhubung dengan Websocket *server,* Aktor menerima pesan yang dikirimkan melalui Websocket *server*, jika pesan yang diterima bertipe teks maka pesan disimpan ke dalam basis data kemudian ditampilkan, jika pesan bertipe gambar maka dilakukan proses *decode* terhadap isi pesan untuk menghasilkan gambar, kemudian gambar disimpan pada *directory* tertentu untuk selanjutnya ditampilkan pada aplikasi
Alternative Flow	Jika aktor sedang tidak terhubung dengan Websocket *server* maka pengguna tidak dapat menerima pesan, jika aplikasi sudah terhubung dengan Websocket *server* maka pesan-pesan yang sebelumnya dikirimkan oleh pengirim pesan akan diterima oleh aktor

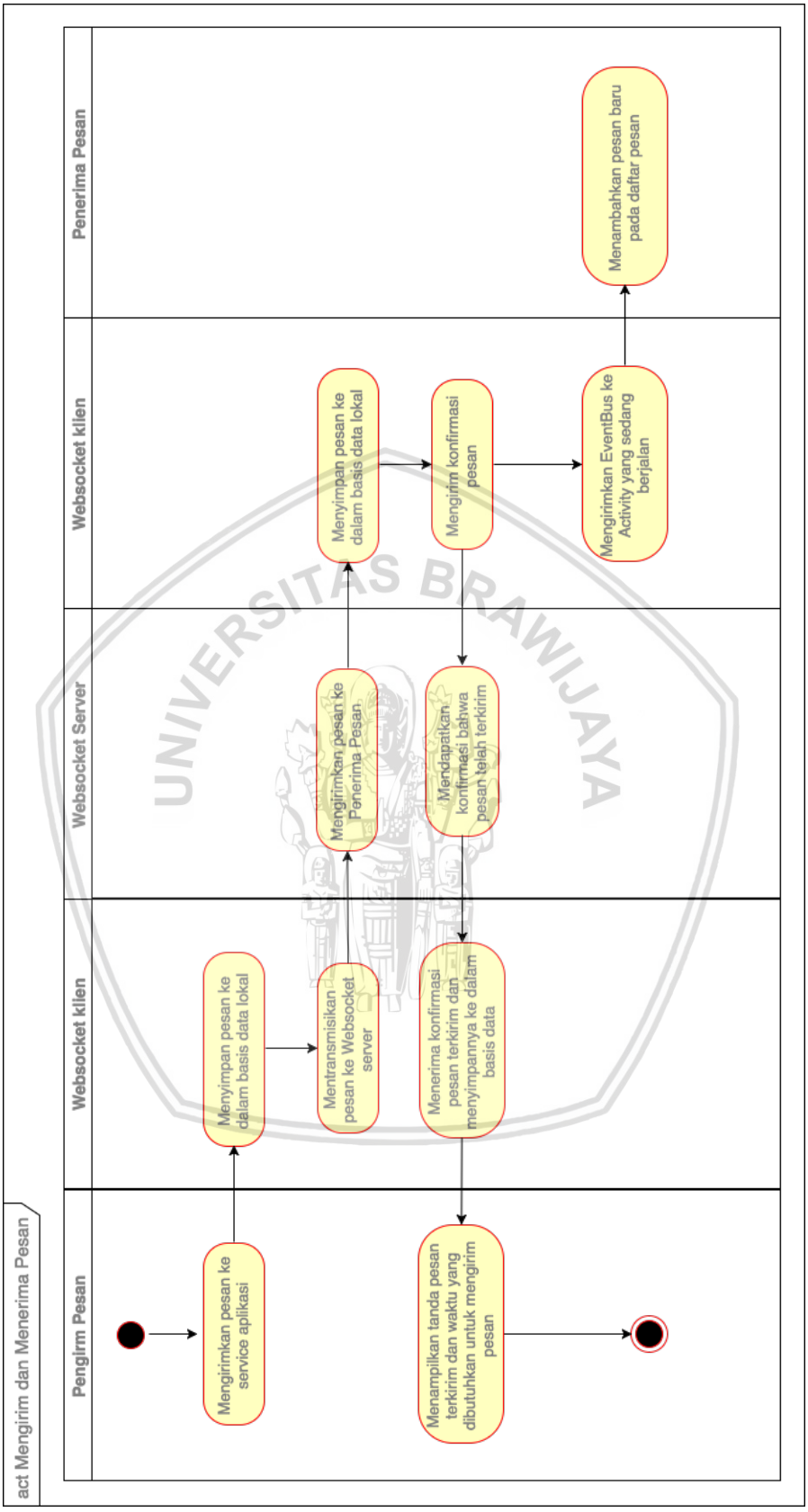
4.3.5 Diagram *Activity*

Diagram *Activity* memodelkan antara pengguna dan sistem yang berjalan sesuai deskripsi *use case* yang telah dibuat sebelumnya. Adapun diagram *activity* yang akan dibuat adalah *activity* mengirim pesan dan menerima pesan. Diagram *activity* dari sistem ini terdapat dalam Gambar 4.3.

Berdasarkan Deskripsi *Use Case* yang telah dijabarkan pada Tabel 4.3 dapat dibuat Diagram *Activity* Mengirim dan Menerima Pesan. Gambar 4.3 menjelaskan mengenai bagaimana pesan dikirimkan oleh pengirim pesan kepada Websocket *server* ketika sudah terhubung dengan Websocket *server* yang selanjutnya diteruskan kepada penerima pesan, ketika pesan telah diterima oleh penerima pesan maka penerima pesan akan mengirimkan pesan notifikasi ke Websocket *server* bahwa pesan telah diterima, kemudian pesan itu diteruskan kepada pengirim pesan. Kondisi awal yang harus terpenuhi adalah masing-masing penerima dan pengirim pesan telah terhubung dengan Websocket server.

4.3.6 Diagram *Sequence*

Diagram *sequence* digunakan untuk menggambarkan interaksi antara aktor dan sistem yang disusun dalam urutan waktu. Diagram *sequence* berupa deskripsi atau rangkaian langkah-langkah yang dilakukan dari sebuah kejadian untuk menghasilkan keluaran tertentu. Diagram *sequence* dari sistem ini dapat dilihat dalam Gambar 4.4.



Gambar 4.3 Diagram *Activity* Mengirim Pesan

Berdasarkan Diagram *Activity* dalam Gambar 4.3 maka dapat dirancang Diagram *Sequence* untuk mengirim pesan. Dalam Gambar 4.4 menjelaskan proses pengiriman pesan yang dimula dengan terlebih dahulu menghubungkan Websocket klien dengan Webocket *server* agar pengirim pesan dapat mengirimkan pesan ke penerima pesan dengan memasukkan pesan berupa teks atau gambar yang kemudian disimpan pada basis data lokal dan sekaligus dikirimkan ke Websocket *server* untuk diteruskan kepada penerima pesan, jika pesan telah diterima maka penerima pesan akan membalas dengan pesan notifikasi bahwa pesan telah berhasil dikirimkan, ketika mendapatkan notifikasi bahwa pesan telah berhasil dikirimkan maka data pesan pada basis data lokal akan diperbaharui untuk menambahkan waktu penerimaan pesan yang secara otomatis akan memperbaharui tampilan pesan dengan indikasi tanda cawang ganda berwarna biru. Kondisi awal yang harus terpenuhi adalah masing-masing penerima dan pengirim pesan telah terhubung dengan Websocket server.

4.3.7 Diagram *Class*

Diagram *class* merupakan sebuah gambaran tentang sistem dan relasi-relasi yang terdapat di dalamnya. Diagram *class* dari sistem ini terdapat dalam Gambar 4.5.

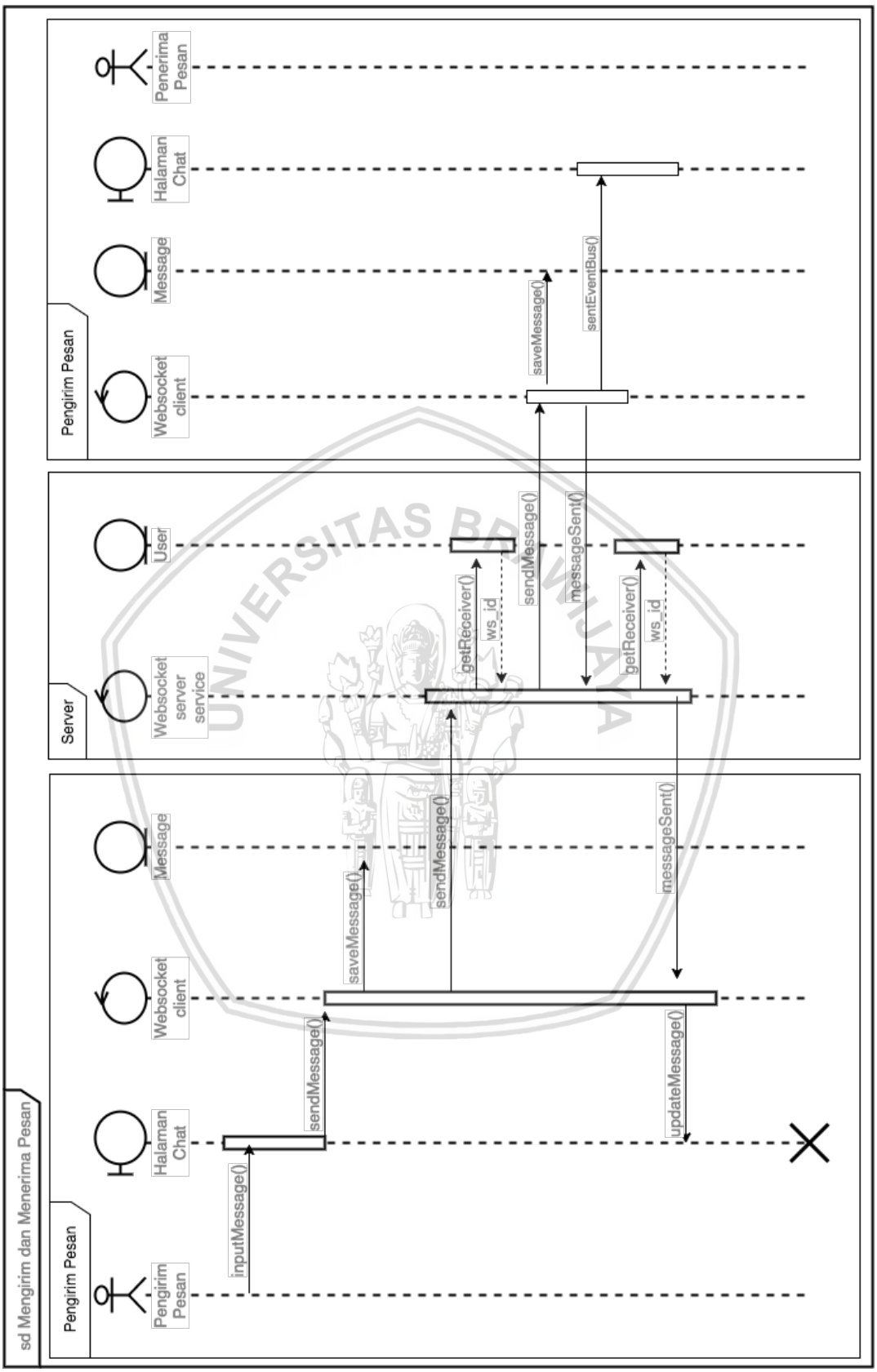
Dalam Gambar 4.5 menjelaskan diagram *class* dari proses mengirim pesan. Setelah koneksi antara Websocket klien dan Websocket *server* terbentuk maka pengirim pesan dapat mengirimkan pesan melalui Websocket klien yang kemudian dikrimkan kepada menerima pesan melalui Websocket *server*, ketika pesan sudah diterima maka pengirim pesan akan mendapatkan pesan notifikasi bahwa pesan terlah diterima oleh penerima pesan, dari sini maka bisa diperoleh berapa lama proses pengiriman pesan.

4.4 Perancangan Sistem

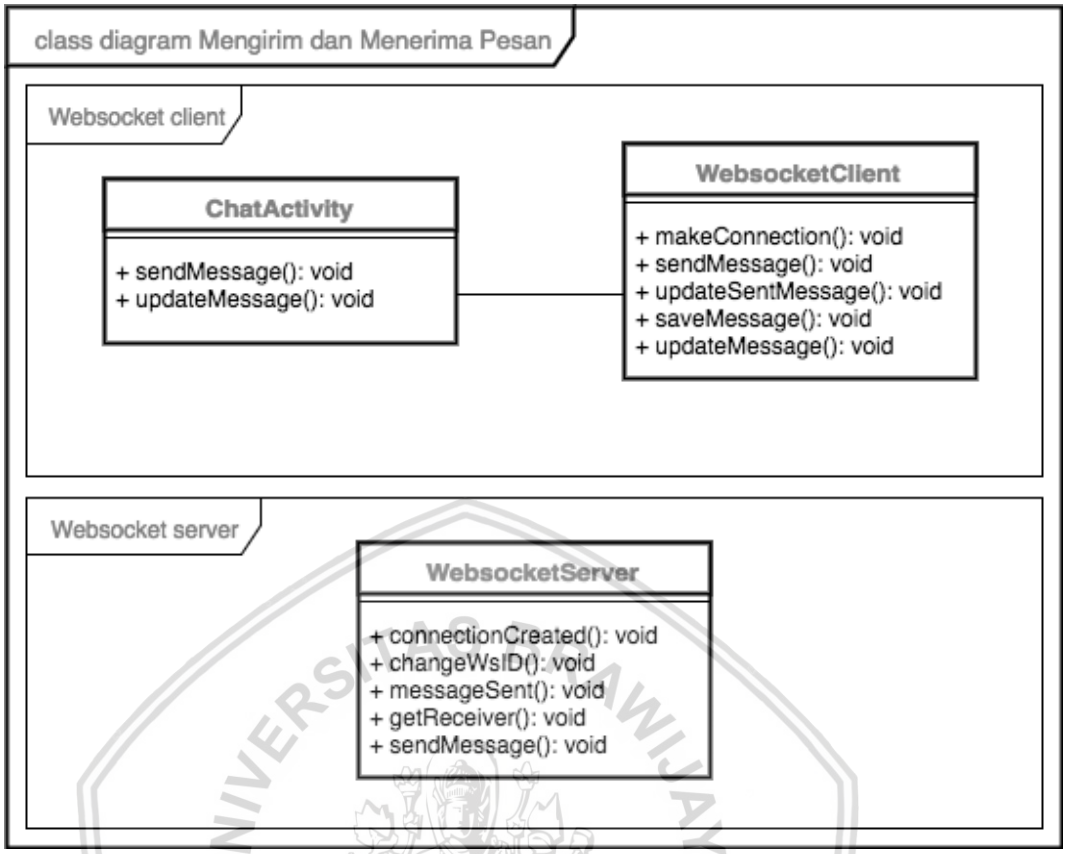
Perancangan pada subbab ini dibagi dalam 5 bagian yaitu perancangan basis data baik di sisi klien maupun *server*, perancangan Web API, perancangan *library* Websocket (Nv Websocket Client, AndroidAsync, dan Java Websocket), perancangan antarmuka dan perancangan skenario pengujian. Setelah perancangan selesai maka selanjutnya akan dilakukan implementasi berdasarkan perancangan yang telah dibuat.

4.4.1 Perancangan Basis Data

Aplikasi pada penelitian ini menggunakan basis data baik dari sisi *server* maupun sisi klien. Basis data ini digunakan untuk menyimpan data-data pengguna seperti data profil, pesan-pesan yang telah dikirimkan maupun diterima oleh pengguna dan juga notifikasi bahwa pesan telah diterima. Pada sisi *server* basis data yang digunakan adalah mongoDB, sendangkan di sisi klien basis data yang digunakan adalah Realm.



Gambar 4.4 Diagram *Sequence* Mengirim Pesan



Gambar 4.5 Diagram *Class* Mengirim Pesan

4.4.2 Basis Data Pada Sisi *Server*

Pada sisi *server* basis data yang digunakan merupakan salah satu dari basis data NoSQL yaitu mongoDB, di mana basis data ini menyimpan data dalam bentuk dokumen yang mirip dengan JSON. Pada sisi *server* terdapat dua *collection* yaitu user dan message. Pada *collection* user menyimpan data pengguna berupa id, username, name, email, phone, password, image (url) dan ws_id. Sedangkan pada *collection* message menyimpan data pesan berupa id, message, type, time, from dan to.

4.4.3 Basis Data Pada Sisi *client*

Pada sisi *client* basis data yang digunakan adalah Realm dengan beberapa *model classes (object)* di antaranya auth, user, contact, message dan settings. Dari masing-masing *object* di dalamnya memiliki variabel yang berbeda-beda yaitu:

1. Auth : email, password
2. User : id, name, username, email, image, phone
3. Contact : id, name, username, email, phone, image
4. Message : id, type, message, send_time, arrival_time, from, to

4.4.4 Perancangan Web API

Bahasa yang akan digunakan Web API adalah Python dengan menggunakan sebuah *micro framework* yaitu Bottle dengan menggunakan sebuah *server library* yang mendukung *multi-threading*, stabil dan sudah teruji yaitu Paste. Beberapa *endpoint* yang dibutuhkan di antaranya login, contact, dan logout.

4.4.5 Perancangan Websocket *Server*

Bahasa yang akan digunakan sebagai Websocket *server* adalah Python dengan menggunakan sebuah *library* yaitu websocket_server yang mampu mentransmisikan pesan melalui protokol Websocket dengan struktur pesan yang dikirimkan yang dijelaskan pada tabel 4.5.

Tabel 4.5 Struktur Pesan

No	Parameter	Keterangan
1	id	Berupa angka acak berjumlah 8 digit untuk membedakan pesan yang satu dengan yang lain
2	type	Tipe pesan yang dikirimkan dapat berupa text atau image
3	message	Isi pesan yang dikirimkan berupa teks, jika tipe merupakan gambar maka parameter ini berisi teks hasil *encode* base64 dari gambar yang dipilih
4	send_time	Waktu pengiriman pesan mengacu pada saat pesan dikirimkan dari pengirim pesan
5	arrival_time	Waktu penerimaan pesan mengacu pada saat pesan diterima oleh penerima pesan
6	from	Id dari pengirim pesan
7	to	Id dari penerima pesan
8	status	Staus pesan yang menunjukan pesan tersebut sudah sampai pada tahap apa seperti sudah terkirim (ke *sever*), gagal terkirim, atau sudah sampai pada penerima

4.4.6 Perancangan *Library* Websocket

Terdapat 3 *library* yang akan digunakan pada sistem yang akan dibuat yaitu Nv Websocket Client, AndroidAsync dan Java Websocket. Masing-masing *library* akan dijalankan sebagai *service* yang berjalan secara bergantian dan dapat diatur oleh pengguna, pengaturan penggunaan *library* dapat dilakukan dengan cara mengetikkan perintah misal "/set library 1" untuk menggunakan *library* AndroidAsync, kemudian menekan tombol untuk mengirim pesan.

4.4.6.1 Nv Websocket Client

Untuk menggunakan *library* Nv Websocket Client langkah pertama yang perlu dilakukan adalah menambahkan *libary* pada *project* yang dibuat dengan cara menambahkan library pada *gradle dependencies*. *Library* Websocket klien akan dijalankan sebagai sebuah *service* ketika aplikasi terbuka. Jika pengguna pengirimkan pesan maka pesan akan dikirimkan kepada *service* Websocket klien yang sedang berjalan kemudan pesan tersebut akan dikirimkan kepada Websocket *server* untuk diteruskan kepada penerima pesan. Ketika pesan sudah terkirim maka *service* tersebut akan mendapatkan notifikasi bahwa pesan telah diterima yang selanjutnya akan dilakukan pembaharuan pada pesan yang telah dikirimkan tersebut dengan menambahkan waktu penerimaan pesan yang secara otomatis akan memperbaharui tampilan pesan dengan menambahkan tanda cawang ganda berwarna biru pada pesan tersebut.

4.4.6.2 AndroidAsync

Pada subbab ini proses yang dilakukan sama seperti pada subbab 4.4.6.1, yang membedakan adalah pada subbab ini menggunakan *library* AndroidAsync, untuk menggunakan library ini pengguna memasukkan perintah "/set library 1" kemudian menekan tombol kirim.

4.4.6.3 Java Websocket

Pada subbab ini proses yang dilakukan sama seperti pada subbab 4.4.6.1, yang membedakan adalah pada subbab ini menggunakan *library* Java Websocket, untuk menggunakan library ini pengguna memasukkan perintah "/set library 2" kemudian menekan tombol kirim.

4.4.7 Perancangan Antarmuka

Perancangan antarmuka aplikasi pada penelitian ini dibuat sebagai sketsa yang menggambarkan kebutuhan aplikasi. Berikut ini adalah rancangan antarmuka dari aplikasi:

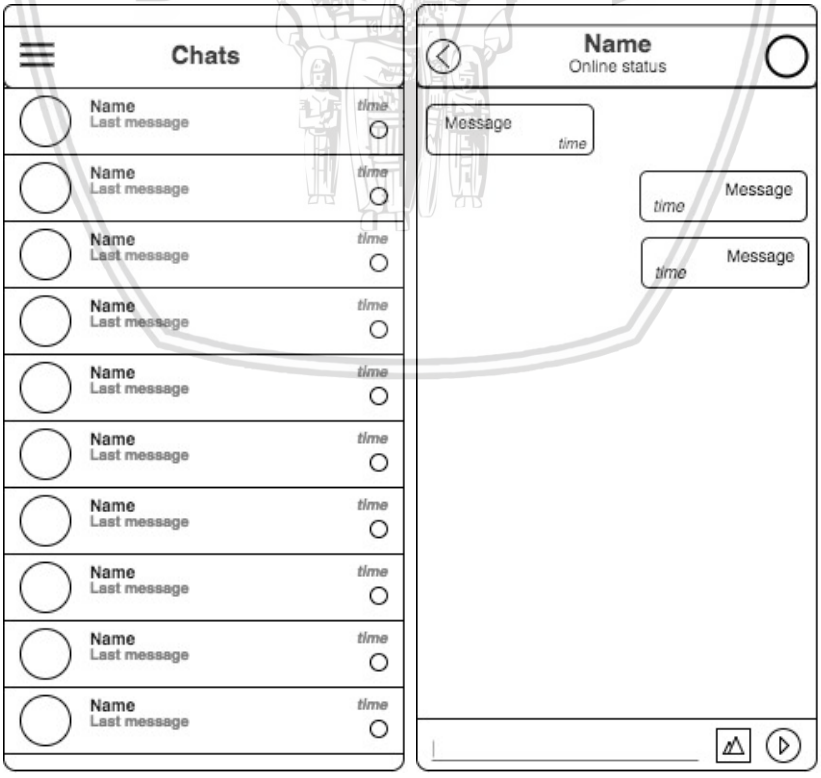
4.4.7.1 Desain halaman login.

Pada halaman login terdapat beberapa komponen yang perlu ditampilkan yaitu input berupa email dan password untuk validasi pengguna serta tombol yang berfungsi untuk mencocokkan apakah email dan password sesuai atau tidak, jika sudah sesuai maka pengguna akan diarahkan ke halaman utama. Perancangan antarmuka halaman login terdapat dalam Gambar 4.6.

4.4.7.2 Desain halaman utama.

Halaman utama menampilkan daftar obrolan dengan pengguna lain, masing-masing daftar berisi foto, nama, isi pesan terakhir (dikirim/diterima), waktu penerimaan/pengiriman pesan yang terakhir, serta jumlah pesan yang belum terbaca jika ada. Perancangan antarmuka halaman utama terdapat dalam Gambar 4.7.

Gambar 4.6 Desain halaman login



Gambar 4.7 Desain halaman utama dan halaman chat

4.4.7.3 Desain halaman chat.

Halaman chat menampilkan daftar percakapan yang telah dilakukan masing-masing daftar percakapan berisi isi pesan, waktu pengiriman/penerimaan pesan dalam format jam, menit dan detik, dan waktu yang diperlukan untuk mengirimkan pesan dalam *millisecond*. Komponen lain yang nantinya terdapat pada halaman chat yaitu nama serta foto pengguna lain yang sedang diajak melakukan percakapan, kotak input untuk menuliskan pesan serta tombol untuk mengirimkan pesan. Perancangan antarmuka halaman chat terdapat dalam Gambar 4.7.

4.4.8 Perancangan Skenario Pengujian

Pengujian akan dilakukan dengan cara menjalankan *server* dan juga aplikasi *chat* klien yang dengan menentukan *library* Websocket klien yang akan digunakan, kemudian masuk ke fitur *chat* dan memilih pesan (gambar) yang akan digunakan untuk pengujian, ukuran gambar yang akan digunakan pada pengujian ini yaitu 100KB, 200KB, 500KB, 1MB dan 5MB. Pengujian ini masing-masing akan dilakukan dengan mengirimkan pesan sebanyak 50 kali dengan masing-masing ukuran pesan yang berbeda, pengujian ini dilakukan sebanyak 30 kali dengan jeda masing-masing pesan selama 2 detik untuk mendapatkan variasi hasil yang nanti akan dicari hasil rata-ratanya. Pada setiap pengujian akan dicatat jumlah rata-rata penggunaan daya yang digunakan serta jumlah rata-rata waktu yang diperlukan untuk mengirimkan pesan pada masing-masing pengujian. Penerima pesan berupa *bot* yang dapat memberikan notifikasi otomatis ketika pesan telah terkirim.

BAB 5 IMPLEMENTASI

Bab ini membahas implementasi sistem berdasarkan perancangan perangkat lunak yang telah dibuat. Implementasi sistem pada pembahasan ini yang terdiri dari spesifikasi perangkat lunak, spesifikasi perangkat keras, batasan implementasi, implementasi basis data, impementasi *web* API, implementasi Websocket *server*, dan implementasi Websocket klien.

5.1 Spesifikasi Perangkat Lunak

Perangkat lunak yang akan digunakan dalam penelitian ini menggunakan spesifikasi yang dijelaskan pada tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Lunak *Smartphone*

No	Nama	Spesifikasi
1	Sistem Operasi	Android 6.0.1 Marshmallow
2	Kernel	3.10.84-perf-gf1cfa5f BuildUser@BuildHost #1
3	*Profiler app*	Trepn™ Profiler 6.2

5.2 Spesifikasi Perangkat Keras

Perangkat lunak yang akan digunakan dalam penelitian ini menggunakan spesifikasi yang dijelaskan pada tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Keras *Smartphone*

No	Nama	Spesifikasi
1	Prosesor	Octa-core (4x1.5 GHz Cortex-A53 & 4x2.0 GHz Cortex-A57)
2	*Chipset*	Qualcomm MSM8994 Snapdragon 810
3	Memori	3 GB

5.3 Batasan Implementasi

Batasan-batasan implementasi sistem ini sebagai berikut:

1. Pengujian dilakukan dengan menggunakan jaringan lokal guna mengurangi gangguan pada jaringan
2. Pengujian dilakukan dengan menjalankan *server* pada *virtualhost* guna mengurangi ganguan penggunaan sumber daya pada komputer
3. Pengujian dilakukan dengan menggunakan *bot* sebagai penerima pesannya.

5.4 Implementasi Basis Data

Implementasi basis data menggunakan realm dengan cara membuat *class* model Java turunan dari *class* RealmObject sebagai model untuk penyimpanan data. Subbab ini diimplementasikan berdaskarkan perancangan pada subbab 4.4.3. Kelas pertama yang dibuat adalah kelas User, kelas ini digunakan sebagai kelas model untuk menyimpan data pengguna, dalam kelas ini terdapat 7 parameter yang masing-masing bertipe data String, parameter pertama yaitu id yang juga sebagai *primary key*, kemudian username, name, email, phone, image dan token. Hasil implementasi kelas User dapat dilihat pada Tabel 5.3.

```
1.  public class User extends RealmObject implements Serializable
    {
2.      @PrimaryKey
3.      @Expose
4.      private String id;
5.      @Expose
6.      private String username;
7.      @Expose
8.      private String name;
9.      @Expose
10.     private String email;
11.     @Expose
12.     private String phone;
13.     @Expose
14.     private String image;
15.     @Expose
16.     private String token;
17. }
```

Tabel 5.3 Implementasi *Class* Model User

Kelas kedua yang dibuat adalah kelas Message, kelas ini dugunakan sebagai kelas model untuk menyimpan pesan baik yang dikirimkan atau yang diterima oleh pengguna, dalam kelas ini terdapat 7 parameter, parameter pertama yang juga merupakan *primary key* yaitu id dengan tipe data String, kemudian from, to, message, type yang juga bertipe data String, time dan sent_time yang bertipe long. Hasil implementasi kelas Message dapat dilihat pada Tabel 5.4.

```
1.   public class Message extends RealmObject {
2.       @PrimaryKey
3.       public String id;
4.       public String from;
5.       public String to;
6.       public String message;
7.       public String type;
8.       public long time;
9.       public long sent_time;
10.  }
```

Tabel 5.4 Implementasi *Class* Model Message

5.5 Implementasi Web API

Implementasi Web API menggunakan bahasa pemrograman Python dengan menggunakan *library* Bottle sebagai *microservice* untuk melayani *request* dari pengguna. Subbab ini diimplementasikan berdaskarkan perancangan pada subbab 4.4.4. Web API ini digunakan oleh pengguna untuk melakukan login

sehingga pengguna mendapatkan token yang akan digunakan untuk bertukar pesan menggunakan Websocket dengan pengguna yang lain. Alur dari algoritme pada fungsi login yaitu yang pertama melakukan pengecekan email dan password yang dikirimkan melalui *basic authentication*, pengecekan dilakukan dengan memeriksa apakah email dan password yang digunakan terdapat dalam basis data dan juga apakah email dan password sesuai, pengecekan dilakukan dengan memanggil fungsi check_pass. Jika email dan password sesuai maka fungsi login akan mengambil data pengguna berupa id, email, name, username, phone, dan image yang terdapat dalam basis data *server*, setelah itu membuat token dengan cara memanggil fungsi generateToken yang kemudian disimpan ke dalam variabel token dan selanjutnya dimasukkan ke dalam variabel cursor untuk nantinya dikirimkan berupa *response* JSON ke klien, selain itu juga untuk parameter image ditambah url yang sesuai agar gambar dapat diakses, selanjutnya melaukan *update* pada token yang terdapat dalam basis data pengguna, setelah semua data siap maka fungsi akan mengembalikan data dalam bentuk JSON yang berisi status dengan nilai true dan message dengan nilai data pengguna. Implementasi *endpoint* login dapat dilihat pada Tabel 5.5.

```
1.  @route('/login', method='GET')
2.  @auth_basic(check_pass)
3.  def login():
4.      email, password = request.auth
5.      cursor = db.user.find({'email': email},
6.                            {'id': 1, 'email': 1, 'name': 1,
    'username': 1, 'phone': 1, 'image': 1})
7.      cursor = cursor[0]
8.      response.content_type = 'application/json'
9.      del cursor['_id']
10.
11.     token = generateToken()
12.
13.     cursor['image'] = "/asset/img/user/%s" % (cursor['image'])
14.     cursor['token'] = token
15.
16.     db.user.update_one(
17.         {"username": cursor['username']},
18.         {
19.             "$set": {"token": token}
20.         }
21.     )
22.
23.     return dumps({'status': True, 'message': cursor})
```

Tabel 5.5 Implementasi Endpoint Login

5.6 Implementasi Websocket *Server*

Implementasi Websocket *Server* menggunakan bahasa pemrograman Python dengan menggunakan *library* WebsocketServer untuk melayani protokol Websocket. Subbab ini diimplementasikan berdasarkan perancangan pada subbab 4.4.5. Fungsi tersebut dijalankan ketika server menerima pesan dari *client* yang kemudian diidentifikasi jenis pesan dan juga pengguna yang dituju, jika user tujuannya adalah "0" maka server akan mengidentifikasi bahwa pesan tersebut adalah pesan pertama yang dikirimkan oleh pengguna untuk mengidentifikasi dan menyesuaikan database agar sesuai dengan Websocket id

dari pengguna. Jika pengguna (penerima pesan) yang dituju terdapat dalam database dan sedang aktif maka server akan langsung mengirimkan pesan ke pengguna (penerima pesan) tersebut. Implementasi Websocket *Server* dapat dilihat pada Tabel 5.6.

```
1.  def message_received(client, server, message):
2.   print "Message %s" % message
3.   message = loads(message, strict=False)
4.
5.   try:
6.           if message['to'] == 0:
7.               server.send_message(client, ("Your WS_ID : %i",
    client["id"]))
8.               db.user.update_one(
9.                   {"token": message["message"]},
10.                  {
11.                      "$set": {"ws_id": client["id"]}
12.                  }
13.              )
14.          else:
15.              message['from'] =
    get_sender_by_id(client['id'])["username"]
16.              server.send_message(get_receiver(message["to"]),
    dumps(message))
17.
18. except IOError:
19.          print "error send message 1"
20.          server.send_message(client, dumps({'id': message['id'],
    'message': 'failed to send message'}))
21.
22.      except Exception as ex:
23.          print "error send message => %s" % dumps({'id':
    message['id'], 'message': 'failed to send message'})
24.          server.send_message(client, dumps({'id': message['id'],
    'message': 'failed to send message', 'status': 'failed'}))
```

Tabel 5.6 Implementasi Websocket server

5.7 Implementasi Websocket Klien

Websocket klien diimplementasikan menggunakan tiga *library* Websocket klien yang berbeda yaitu Nv Websocket Client, AndroidAsync, dan Java Websocekt. Subbab ini diimplementasikan berdaskarkan perancangan pada subbab 4.4.6. Masing-masing library dijalankan pada *background thread* yang terdapat dalam sebuah Service.

5.7.1 Implementasi menggunakan Nv Websocket Client

Implementasi Websocket klien menggunakan Nv Websocket Client dilakukan berdasarkan perancangan pada subbab 4.4.6.1. Ketika fungsi connectSocket dipanggil maka Websocket klien akan menghubungkan ke Websocket server. Sedangkan dalam WebsocketListener terdapat beberapa fungsi di antaranya “onConnected” dijalankan ketika pertama kali terhubung dengan Websocket server, didalamnya dilakukan pengiriman pesan ke Websocket server untuk mengidentifikasi token pengguna. Sedangkan fungsi “onTextMessage” dijalankan ketika Websocket klien menerima pesan dari Websocket server, jika pesan yang diterima bertipe “text” atau “image” maka data tersebut akan disimpan dalam database, kemudan dilakuakn *broadcast* bahwa ada pesan baru yang masuk, dari

broadcast tersebut jika ActivityChat sedang berjalan maka akan menambahkan pesan yang baru masuk pada daftar pesan, sebenarnya Realm sudah menyediakan *listener* sendiri yang dapat berjalan ketika ada perubahan pada basis data, akan tetapi dalam beberapa percobaan terjadi beberapa kegagalan akhirnya untuk memperbarui pesan dilakukan *broadcast* menggunakan EventBus. Jika tipe pesan yang diterima adalah "notification" maka akan memperbarui "sent_time" pada pesan yang telah dikirimkan sebelumnya (bertipe text atau image) dengan id yang sesui dengan pesan tersebut. Hasil implementasi Websocket klien menggunakan Nv Websocket Client terdapat pada Tabel 5.7.

```
1. protected void connectSocket() {
2.     try {
3.         ws = new WebSocketFactory().createSocket(BASE_WS_URL);
4.         ws.addHeader("token", TOKEN);
5.         ws.addListener(new WebSocketListener() {
6.
7.             @Override
8.             public void onConnected(WebSocket websocket,
   Map<String, List<String>> headers) throws Exception {
9.                 Log.d("Websocket Service","Connected");
10.                ws.sendText("{\"to\": 0, \"message\": TOKEN,
   \"type\": \"token\", \"id\": \"-\", \"time\":
   "+System.currentTimeMillis()+"}");
11.            }
12.
13.            @Override
14.            public void onTextMessage(WebSocket websocket,
   final String text) throws Exception {
15.                socketResponse = text;
16.                Log.d("Websocket Service","Received => "+text);
17.
18.                new Handler(Looper.getMainLooper()).post(new
   Runnable() {
19.                    @Override
20.                    public void run() {
21.                        Realm realm =
   Realm.getDefaultInstance();
22.
23. realm.executeTransactionAsync(new Realm.Transaction(){
24.                            @Override
25.                            public void execute(Realm realm) {
26.                                Message message =
   realm.createObjectFromJson(Message.class, text);
27. }
28.
29.                        }, new Realm.Transaction.OnSuccess() {
30.                            @Override
31.                            public void onSuccess() {
32.                                Log.d("REALM", "Success");
33.                                EventBus.getDefault().post(new
   MessageEvent("you have new message. \n "+text));
34.
35.
36.                            }
37.                        }, new Realm.Transaction.OnError() {
38.                            @Override
39.                            public void onError(Throwable
   error) {
40.                                Log.d("REALM",
   "Failed\n"+error.getMessage());
```

```
41.
42.                                 }
43.                             });
44. }
45.                     });
46.                 }
47. });
48.         ws.connectAsynchronously();
49.
50.     } catch (IOException e) {
51.         e.printStackTrace();
52.     }
53.       }
```

Tabel 5.7 Fungsi connectSocket menggunakan Nv Websocket Client

5.7.2 Implementasi menggunakan AndroidAsync

Implementasi Websocket Klien menggunakan AndroidAsync dilakukan berdasarkan perancangan pada subbab 4.4.6.2. Yang membedakan dengan penggunakan pada *library* Nv Websocket Client adalah pengiriman pesan pertama kali ke server menggunakan fungsi "Util.writeAll" yang terdapat dalam fungsi "handleConnectCompleted", dan ketika menerima pesan fungsi yang dipanggil adalah "onDataAvailable". Hasil implementasi Websocket klien menggunakan AndroidAsync terdapat pada Tabel 5.8.

```
1. protected void connectSocket() {
2. AsyncServer.getDefault().connectSocket(new
   InetSocketAddress(BASE_IP, WS_PORT), new ConnectCallback() {
3.     @Override
4.     public void onConnectCompleted(Exception ex, final
   AsyncSocket socket) {
5.         mSocket = socket;
6.         handleConnectCompleted(ex, socket);
7.     }
8. });
9. }
10.
11. private void handleConnectCompleted(Exception ex, final
   AsyncSocket socket) {
12. if(ex != null) throw new RuntimeException(ex);
13.
14. Util.writeAll(socket, ("{\"to\": 0, \"message\": \""+TOKEN+"\",
   \"type\": \"token\", \"id\": \"-\", \"time\":
   "+System.currentTimeMillis()+"}").getBytes(), new
   CompletedCallback() {
15.     @Override
16.     public void onCompleted(Exception ex) {
17.         if (ex != null) throw new RuntimeException(ex);
18.         System.out.println("[Client] Successfully wrote
   message");
19.     }
20. });
21.
22. socket.setDataCallback(new DataCallback() {
23.     @Override
24.     public void onDataAvailable(DataEmitter emitter,
   ByteBufferList bb) {
25.         final String text =  new String(bb.getAllByteArray());
26.         System.out.println("[Client] Received Message " +
   text);
27.
28.         socketResponse = text;
```

```
29.         Log.d("Websocket Service","Received => "+text);
30.
31.         new Handler(Looper.getMainLooper()).post(new Runnable()
    {
32.             @Override
33.             public void run() {
34.
35.                 final Realm realm = Realm.getDefaultInstance();
36.                 final long now = System.currentTimeMillis();
37.
38.                 JSONObject object = null;
39.                 try {
40.                     object = new JSONObject(text);
41.                     final String msg_content =
    object.getString("message");
42.                     String msg_type = object.getString("type");
43.
44.                     if (msg_type.equals("notification")){
45.                         realm.executeTransactionAsync(new
    Realm.Transaction() {
46.                             @Override
47.                             public void execute(Realm realm) {
48.                                 Message message =
    realm.where(Message.class).equalTo("id",
    msg_content).findFirst();
49.                                 if (message != null){
50.                                     message.sent_time = now;
51.                                 }
52.                             }
53.                         }, new Realm.Transaction.OnSuccess() {
54.                             @Override
55.                             public void onSuccess() {
56.                                 EventBus.getDefault().post(new
    MessageEvent("you have new message. \n "+text));
57.                             }
58.                         });
59.                     }else{
60.                         realm.executeTransactionAsync(new
    Realm.Transaction(){
61.                             @Override
62.                             public void execute(Realm realm) {
63.                                 Message message =
    realm.createObjectFromJson(Message.class, text);
64.                                 message.sent_time = now;
65.                             }
66.
67.                         }, new Realm.Transaction.OnSuccess() {
68.                             @Override
69.                             public void onSuccess() {
70.                                 Log.d("REALM", "Success");
71.                                 EventBus.getDefault().post(new
    MessageEvent("you have new message. \n "+text));
72.                             }
73.                         }, new Realm.Transaction.OnError() {
74.                             @Override
75.                             public void onError(Throwable
    error) {
76.                                 Log.d("REALM",
    "Failed\n"+error.getMessage());
77.                             }
78.                         });
79.                     }
80.                 } catch (JSONException e) {
81.                     e.printStackTrace();
82.                 }
```

```
            }
        });
    }
});
}
```

Tabel 5.8 Fungsi connectSocket menggunakan AndroidAsync

5.7.3 Implementasi menggunakan Java Websocket

Implementasi Websocket Klien menggunakan Java Websocket dilakukan berdasarkan perancangan pada subbab 4.4.6.3. Yang membedakan dengan penggunakan pada *library* Nv Websocket Client adalah pengiriman pesan pertama kali ke server menggunakan fungsi "onOpen", dan ketika menerima pesan fungsi yang dipanggil adalah "onMessage". Hasil implementasi Websocket klien menggunakan Java Websocket terdapat pada Tabel 5.9.

```
private void connectSocket(){
try {
ws = new WebSocketClient( new URI( BASE_WS_URL )) {

@Override
public void onMessage(final String text ) {
    socketResponse = text;
    Log.d("Websocket Service","Received => "+text);

    new Handler(Looper.getMainLooper()).post(new Runnable() {
        @Override
        public void run() {

            final Realm realm = Realm.getDefaultInstance();
            final long now = System.currentTimeMillis();

            JSONObject object = null;
            try {
                object = new JSONObject(text);
                final String msg_content =
object.getString("message");
                String msg_type = object.getString("type");

                if (msg_type.equals("notification")){
                    realm.executeTransactionAsync(new
Realm.Transaction() {
                        @Override
                        public void execute(Realm realm) {
                            Message message =
realm.where(Message.class).equalTo("id",
msg_content).findFirst();
                            if (message != null){
                                message.sent_time = now;
                            }
                        }
                    }, new Realm.Transaction.OnSuccess() {
                        @Override
                        public void onSuccess() {
                            EventBus.getDefault().post(new
MessageEvent("you have new message. \n "+text));
                        }
                    });
                }else{
                    realm.executeTransactionAsync(new
Realm.Transaction(){
                        @Override
                        public void execute(Realm realm) {
```

```
                                Message message =
    realm.createObjectFromJson(Message.class, text);
                                message.sent_time = now;
                            }

                        }, new Realm.Transaction.OnSuccess() {
                            @Override
                            public void onSuccess() {
                                Log.d("REALM", "Success");
                                EventBus.getDefault().post(new
    MessageEvent("you have new message. \n "+text));
                            }
                        }, new Realm.Transaction.OnError() {
                            @Override
                            public void onError(Throwable error) {
                                Log.d("REALM",
    "Failed\n"+error.getMessage());

                            }
                        });
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        });
    }

    @Override
    public void onOpen( ServerHandshake handshake ) {
        Log.d("Websocket Service","Connected");
        ws.send("{\"to\": 0, \"message\": \""+TOKEN+"\", \"type\":
    \"token\", \"id\": \"-\", \"time\":
    "+System.currentTimeMillis()+"}");
    }

    };
    ws.connect();
    } catch ( URISyntaxException ex ) {

    }
    }
```

Tabel 5.9 Fungsi connectSocket menggunakan Java Websocket

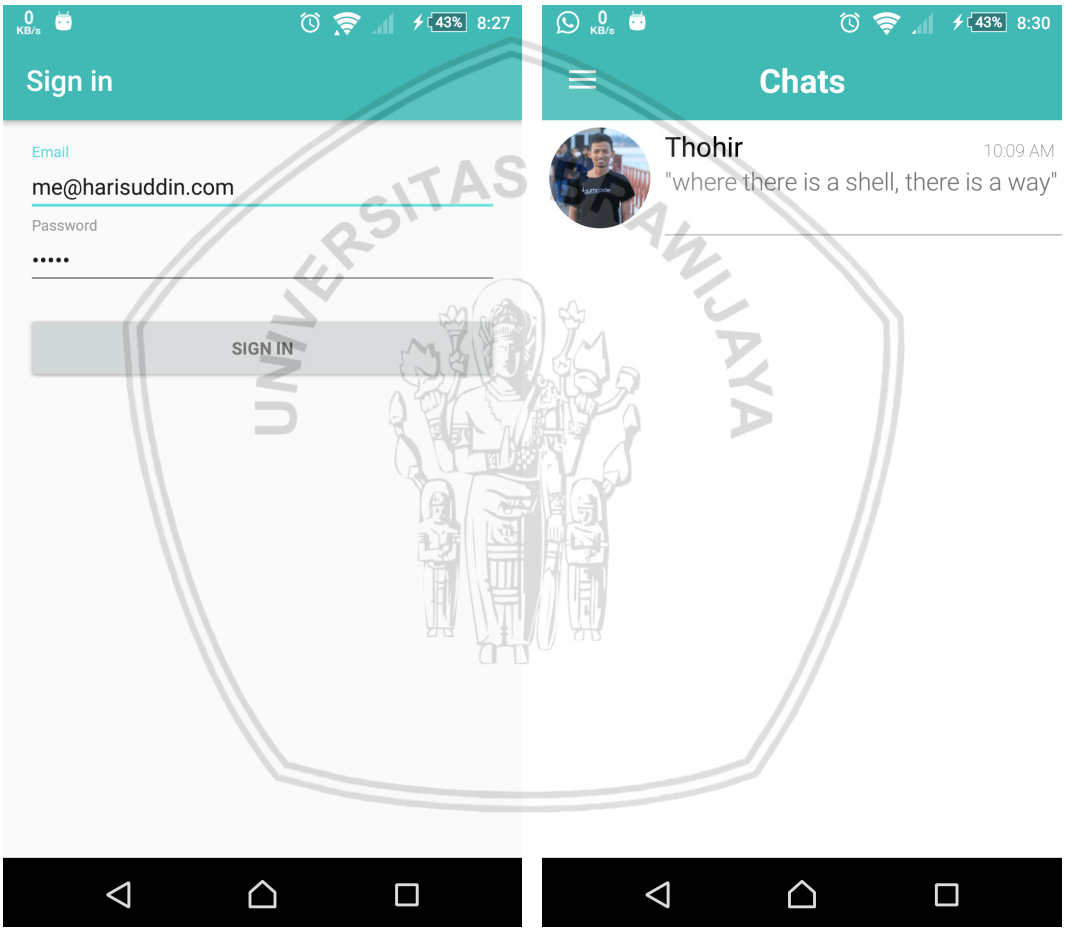
5.8 Implementasi Antarmuka Aplikasi

Pada halaman login terdapat beberapa komponen yang perlu ditampilkan yaitu input berupa email dan password untuk validasi pengguna serta tombol “singn in” untuk menjalankan fungsi guna mencocokkan apakah email dan password sesuai atau tidak, jika sudah sesuai maka pengguna akan diarahkan ke halaman utama. Antarmuka ini merupakan hasil implementasi dari perancangan pada subbab 4.4.7.1, hasil implementasi dari halaman login dapat dilihat dalam Gambar 5.1.

Halaman utama menampilkan daftar obrolan dengan pengguna lain, masing-masing daftar berisi foto, nama, isi pesan terakhir (dikirim/diterima), waktu penerimaan/pengiriman pesan yang terakhir, serta jumlah pesan yang belum terbaca jika ada. Antarmuka ini merupakan hasil implementasi dari perancangan

pada subbab 4.4.7.2, hasil implementasi halaman utama dapat dilihat dalam Gambar 5.1.

Halaman chat menampilkan daftar percakapan yang telah dilakukan masing-masing daftar percakapan berisi isi pesan, waktu pengiriman/penerimaan pesan dalam format jam, menit dan detik, dan waktu yang diperlukan untuk mengirimkan pesan dalam *millisecond*. Komponen lain yang nantinya terdapat pada halaman chat yaitu nama serta foto pengguna lain yang sedang diajak melakukan percakapan, kotak input untuk menuliskan pesan serta tombol untuk mengirimkan pesan. Antarmuka ini merupakan hasil implementasi dari perancangan pada poin 4.4.7.3, hasil implementasi dapat dilihat dalam Gambar 5.2.



Gambar 5.1 Implementasi halaman login dan halaman utama

Gambar 5.2 Implementasi halaman chat

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini akan membahas mengenai prosedur pengujian masing-masing *library* Websocket *client.* Parameter yang akang digunakan dalam pengujian ini yaitu kecepatan pengiriman pesan dan daya yang digunakan. Hasil dari pengujian akan digunakan untuk menganalisis performansi masing-masing *library.*

6.1 Skenario Pengujian

Pengujian dilakukan dengan terlebih dahulu menjalankan Websocket *server* dan juga aplikasi *chat* klien dengan menentukan *library* Websocket klien yang akan digunakan untuk pengujian, kemudian masuk ke fitur *chat* dan memilih pesan (gambar) yang akan digunakan untuk pengujian, ukuran gambar yang akan digunakan pada pengujian ini yaitu 100KB, 200KB, 500KB, 1MB dan 5MB. Untuk memilih gambar memasukkan perintah "/config" kemudian memilih gambar dari gallery, setelah itu memasukkan perintah "/start 50" untuk melakukan pengiriman pesan sebanyak 50 kali dengan jeda masing-masing pesan selama 3 detik. Masing-masing pengujian akan diulang sebanyak 30 kali pada setiap masing-masing ukuran pesan dan *library* untuk mendapatkan variasi hasil yang nanti akan dicari nilai rata-ratanya.

Data dikirimkan dalam bentuk JSON yang berisi id, from, to, message, type, time, dan sent_time. Untuk gambar yang berupa hasil *encode* disimpan dalam parameter message. Pada setiap pengujian akan dicatat jumlah rata-rata penggunaan daya yang digunakan serta jumlah rata-rata waktu yang diperlukan untuk mengirimkan pesan pada masing-masing pengujian. Penerima pesan berupa *bot* yang dapat mengirimkan notifikasi secara otomatis ketika pesan telah terkirim.

6.2 Pengujian AndroidAsync

Prosedur pengujian *library* AndroidAsync dimulai dengan masuk pada halaman chat, kemudian memasukkan perintah "/set library 1" kemudian tekan tombol kirim untuk menjadikan AndroidAsync sebagai *library* yang akan digunakan.

6.2.1 Pengiriman pesan dengan ukuran 100 KB

Hasil pengujian dengan ukuran 100 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.1.

Tabel 6.1 Hasil pengujian ukuran 100 KB dengan AndroidAsync

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	663	151	8969
2	603	166	6242

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
3	625	160	6224
4	602	166	6356
5	627	159	7697
6	631	158	9128
7	631	158	8910
8	619	162	8963
9	616	162	8913
10	627	159	9047
11	622	161	8991
12	622	161	8967
13	618	162	9017
14	635	157	8985
15	636	157	9075
16	670	149	6025
17	624	160	5644
18	663	151	5289
19	612	163	5612
20	621	161	4768
21	629	159	7208
22	673	149	5942
23	652	153	6839
24	812	123	7296
25	753	133	7454
26	670	149	7540
27	693	144	7498
28	667	150	7598
29	716	140	6790
30	645	155	7492

6.2.2 Pengiriman pesan dengan ukuran 200 KB

Hasil pengujian dengan ukuran 200 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.2.

Tabel 6.2 Hasil pengujian ukuran 200 KB dengan AndroidAsync

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1024	195	9011
2	1012	198	7511
3	1018	196	5242
4	1110	180	6250
5	987	203	9053
6	643	311	9007

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
7	688	291	9244
8	667	300	7846
9	842	238	9038
10	624	321	9057
11	575	348	9065
12	613	326	7615
13	619	323	9064
14	697	287	9176
15	630	317	9140
16	641	312	9086
17	648	309	9053
18	673	297	9116
19	644	311	9027
20	621	322	9054
21	769	260	9102
22	724	276	9078
23	646	310	7534
24	675	296	7596
25	606	330	7253
26	938	213	7595
27	630	317	7569
28	631	317	7578
29	621	322	7516
30	625	320	7548

6.2.3 Pengiriman pesan dengan ukuran 500 KB

Hasil pengujian dengan ukuran 500 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.3.

Tabel 6.3 Hasil pengujian ukuran 500 KB dengan AndroidAsync

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1204	415	9119
2	3795	132	9087
3	1345	372	9099
4	1344	372	7761
5	1195	418	9135
6	1273	393	9052
7	1126	444	9049
8	1114	449	9220
9	1121	446	9077
10	1083	462	9110

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
11	1139	439	9219
12	1089	459	9190
13	1457	343	9148
14	1415	353	9210
15	1064	470	9029
16	1104	453	9182
17	1279	391	9155
18	1104	453	9049
19	1132	442	9236
20	1158	432	9080
21	1231	406	9032
22	1075	465	9119
23	1056	473	9124
24	1062	471	9181
25	1009	496	7763
26	1064	470	9059
27	993	504	8999
28	1074	466	9032
29	1144	437	9001
30	1019	491	9093

6.2.4 Pengiriman pesan dengan ukuran 1 MB

Hasil pengujian dengan ukuran 1 MB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.4.

Tabel 6.4 Hasil pengujian ukuran 1 MB dengan AndroidAsync

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1412	708	9083
2	1335	749	9051
3	1287	777	9083
4	1247	802	9049
5	1309	764	9208
6	1264	791	9102
7	1298	770	9087
8	1296	772	9080
9	1307	765	9089
10	1358	736	9102
11	1366	732	9084
12	1305	766	9127
13	1323	756	9124
14	1341	746	9069

15	1275	784	9108
16	1279	782	9157
17	1322	756	9053
18	1288	776	9082
19	1281	781	9122
20	1278	782	9116
21	1387	721	9171
22	1297	771	9050
23	1259	794	9181
24	1270	787	9029
25	1318	759	9122
26	1324	755	9136
27	1334	750	9187
28	1452	689	9102
29	1382	724	9063
30	1458	686	9106

6.2.5 Pengiriman pesan dengan ukuran 5 MB

Hasil pengujian dengan ukuran 5 MB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.5.

Tabel 6.5 Hasil pengujian ukuran 5 MB dengan AndroidAsync

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1344	3720	9020
2	1255	3984	9162
3	1283	3897	9068
4	1253	3990	9042
5	1229	4068	9111
6	1236	4045	7552
7	1272	3931	7575
8	1269	3940	7614
9	1218	4105	9052
10	1192	4195	9044
11	1274	3925	9071
12	1201	4163	9165
13	1292	3870	9083
14	1270	3937	9048
15	1145	4367	9105
16	1155	4329	9094
17	1201	4163	9044
18	1214	4119	9063
19	1236	4045	9092

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
20	1225	4082	9013
21	1208	4139	9078
22	1229	4068	9139
23	1171	4270	9152
24	1147	4359	9108
25	1179	4241	9158
26	1344	3720	9020
27	1255	3984	9162
28	1283	3897	9068
29	1253	3990	9042
30	1229	4068	9111

6.2.6 Hasil rata-rata kecepatan menggunakan AndroidAsync

Hasil rata-rata pengujian menggunakan AndroidAsync menunjukkan nilai yang terus naik dari segi kecepatan dan mengalami fluktuasi dari segi konsumsi daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.6.

Tabel 6.6 Hasil rata-rata pengujian menggunakan AndroidAsync

No.	Ukuran (KB)	Kecepatan (KB/s)	Daya (mW)
1	100 KB	155	7483
2	200 KB	285	8301
3	500 KB	427	9020
4	1 MB	758	9104
5	5 MB	3804	8702

6.3 Pengujian Java Websocket

Prosedur pengujian *library* Java Websocket dimulai dengan masuk pada halaman chat, kemudian memasukkan perintah "/set library 2" kemudian tekan tombol kirim untuk menjadikan Java Websocket sebagai *library* yang akan digunakan.

6.3.1 Pengiriman pesan dengan ukuran 100 KB

Hasil pengujian dengan ukuran 100 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.7.

Tabel 6.7 Hasil pengujian ukuran 100 KB dengan Java Websocket

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	656	152	5540
2	599	167	6326
3	588	170	4390

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
4	615	163	6102
5	604	166	4212
6	432	231	8979
7	421	238	8956
8	399	251	9017
9	395	253	9027
10	382	262	9136
11	429	233	9068
12	456	219	8994
13	421	238	9031
14	675	148	8969
15	369	271	9107
16	389	257	9002
17	378	265	8978
18	381	262	8933
19	393	254	9047
20	378	265	9030
21	396	253	9017
22	374	267	9089
23	411	243	8998
24	862	116	8929
25	350	286	9017
26	370	270	8955
27	378	265	9125
28	387	258	9009
29	366	273	9032
30	378	265	9032

6.3.2 Pengiriman pesan dengan ukuran 200 KB

Hasil pengujian dengan ukuran 200 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.8.

Tabel 6.8 Hasil pengujian ukuran 200 KB dengan Java Websocket

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	685	292	9037
2	682	293	9120
3	639	313	9060
4	630	317	9498
5	650	308	9082
6	614	326	9659
7	639	313	9155

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
8	642	312	9144
9	656	305	9134
10	648	309	9068
11	657	304	9004
12	695	288	9131
13	666	300	9047
14	682	293	9077
15	677	295	9005
16	669	299	9004
17	685	292	9146
18	667	300	9024
19	704	284	9179
20	682	293	9030
21	685	292	9152
22	715	280	9056
23	686	292	9055
24	747	268	9109
25	781	256	8985
26	685	292	9048
27	701	285	9000
28	663	302	9087
29	756	265	9063
30	653	306	9037

6.3.3 Pengiriman pesan dengan ukuran 500 KB

Hasil pengujian dengan ukuran 500 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.9.

Tabel 6.9 Hasil pengujian ukuran 500 KB dengan Java Websocket

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1073	466	9084
2	1185	422	9018
3	1150	435	9070
4	1193	419	9133
5	1096	456	9120
6	1265	395	8997
7	1103	453	9122
8	1083	462	9053
9	1076	465	9176
10	1074	466	9180
11	1095	457	9088

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
12	1144	437	9116
13	1115	448	9154
14	1105	452	9032
15	1119	447	9042
16	1127	444	9002
17	1329	376	9151
18	1208	414	9091
19	1389	360	9044
20	1175	426	9138
21	1142	438	9346
22	1079	463	9087
23	1133	441	9112
24	1209	414	9089
25	1157	432	9177
26	1184	422	9123
27	1119	447	9080
28	1072	466	8434
29	1107	452	7593
30	1233	406	7597

6.3.4 Pengiriman pesan dengan ukuran 1 MB

Hasil pengujian dengan ukuran 1 MB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.10.

Tabel 6.10 Hasil pengujian ukuran 1 MB dengan Java Websocket

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1370	730	9179
2	1358	736	9158
3	1376	727	9068
4	1325	755	9118
5	1449	690	9182
6	1339	747	9053
7	1378	726	9089
8	1338	747	9174
9	1550	645	9044
10	1631	613	9214
11	1456	687	9066
12	1428	700	9045
13	1382	724	9016
14	1483	674	9009
15	1425	702	9037

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
16	1400	714	9200
17	1486	673	9029
18	1541	649	9020
19	1513	661	9073
20	1429	700	9134
21	1563	640	9131
22	1419	705	9095
23	1433	698	9028
24	1461	684	9054
25	1508	663	9041
26	1412	708	9150
27	1446	692	9118
28	1497	668	9051
29	1393	718	9047
30	1403	713	9151

6.3.5 Pengiriman pesan dengan ukuran 5 MB

Hasil pengujian dengan ukuran 5 MB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.11.

Tabel 6.11 Hasil pengujian ukuran 5 MB dengan Java Websocket

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1385	3610	9045
2	1381	3621	9129
3	1383	3615	9578
4	1427	3504	9021
5	1230	4065	9133
6	1223	4088	9170
7	1192	4195	9099
8	1393	3589	9125
9	1492	3351	9145
10	1273	3928	9107
11	1294	3864	9113
12	1481	3376	9143
13	1460	3425	9048
14	1448	3453	9159
15	1337	3740	9101
16	1276	3918	9071
17	1446	3458	9166
18	1333	3751	9130
19	1413	3539	9208

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
20	1451	3446	9114
21	1491	3353	9206
22	1509	3313	9165
23	1567	3191	9243
24	2456	2036	9139
25	1757	2846	9095
26	1656	3019	9124
27	1600	3125	9085
28	1568	3189	9084
29	1549	3228	9111
30	2132	2345	9077

6.3.6 Hasil rata-rata kecepatan menggunakan Java Websocket

Hasil rata-rata pengujian menggunakan Java Websocket menunjukkan nilai yang terus naik dari segi kecepatan dan mengalami fluktuasi dari segi konsumsi daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.12.

Tabel 6.12 Hasil rata-rata pengujian menggunakan Java Websocket

No.	Ukuran (KB)	Kecepatan (KB/s)	Daya (mW)
1	100 KB	232	8402
2	200 KB	296	9106
3	500 KB	440	8982
4	1 MB	696	9092
5	5 MB	3439	9138

6.4 Pengujian Nv Websocket Client

Prosedur pengujian *library* Nv Websocket Client dimulai dengan masuk pada halaman chat, kemudian memasukkan perintah "/set library 3" kemudian tekan tombol kirim untuk menjadikan Nv Websocket Client sebagai *library* yang akan digunakan.

6.4.1 Pengiriman pesan dengan ukuran 100 KB

Hasil pengujian dengan ukuran 100 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.13.

Tabel 6.13 Hasil pengujian ukuran 100 KB dengan Nv Websocket Client

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	473	211	8949
2	430	233	9033
3	441	227	8935

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
4	397	252	8970
5	371	270	8976
6	385	260	8990
7	382	262	9025
8	368	272	8935
9	357	280	9010
10	404	248	9076
11	367	272	9096
12	434	230	9017
13	371	270	9068
14	395	253	8959
15	413	242	8955
16	404	248	9031
17	359	279	8926
18	400	250	8981
19	383	261	9229
20	402	249	8964
21	353	283	9060
22	377	265	9067
23	406	246	9011
24	368	272	9030
25	380	263	9086
26	354	282	9034
27	319	313	9136
28	339	295	9050
29	338	296	9064
30	381	262	9057

6.4.2 Pengiriman pesan dengan ukuran 200 KB

Hasil pengujian dengan ukuran 200 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.14.

Tabel 6.14 Hasil pengujian ukuran 200 KB dengan Nv Websocket Client

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	962	208	9046
2	1068	187	9107
3	903	221	9018
4	968	207	9031
5	822	243	9197
6	921	217	9130
7	688	291	9039

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
8	694	288	9019
9	618	324	9048
10	628	318	9046
11	583	343	8955
12	607	329	9067
13	625	320	9007
14	627	319	9041
15	615	325	9002
16	584	342	9003
17	628	318	9097
18	943	212	9042
19	689	290	9095
20	702	285	9095
21	764	262	9095
22	680	294	9091
23	767	261	9047
24	715	280	9031
25	809	247	9007
26	751	266	9042
27	810	247	8932
28	772	259	9065
29	792	253	8944
30	811	247	9118

6.4.3 Pengiriman pesan dengan ukuran 500 KB

Hasil pengujian dengan ukuran 500 KB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.15.

Tabel 6.15 Hasil pengujian ukuran 500 KB dengan Nv Websocket Client

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1218	411	9146
2	1219	410	9028
3	1140	439	9065
4	1194	419	7658
5	1176	425	7450
6	1213	412	9129
7	1244	402	9229
8	1331	376	9175
9	1278	391	9129
10	1221	410	9209
11	1187	421	9096

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
12	1177	425	9163
13	1341	373	9190
14	1314	381	9178
15	1252	399	9220
16	1351	370	9160
17	1215	412	9162
18	1615	310	9146
19	1161	431	9234
20	1203	416	9100
21	1196	418	9133
22	1203	416	9192
23	1219	410	9222
24	1263	396	9250
25	1233	406	9153
26	1125	444	9189
27	1239	404	9306
28	1248	401	9134
29	1261	397	9208
30	1194	419	9153

6.4.4 Pengiriman pesan dengan ukuran 1 MB

Hasil pengujian dengan ukuran 1 MB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.16.

Tabel 6.16 Hasil pengujian ukuran 1 MB dengan Nv Websocket Client

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1656	604	9083
2	1436	696	9087
3	1516	660	9150
4	1515	660	9151
5	1486	673	9126
6	1494	669	9220
7	1478	677	9189
8	1525	656	9136
9	1481	675	9187
10	1907	524	9170
11	1754	570	8178
12	1556	643	7580
13	1582	632	9209
14	1579	633	9154
15	1476	678	9130

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
16	1348	742	9106
17	1390	719	9099
18	1377	726	9081
19	1321	757	9126
20	1380	725	9090
21	1528	654	9154
22	1666	600	8674
23	1519	658	9142
24	1595	627	9063
25	1580	633	9203
26	1643	609	9157
27	1351	740	9127
28	1444	693	9135
29	1600	625	9143
30	1497	668	9161

6.4.5 Pengiriman pesan dengan ukuran 5 MB

Hasil pengujian dengan ukuran 5 MB menunjukkan nilai yang fluktuatif baik dari segi kecepatan pengiriman data maupun daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.17.

Tabel 6.17 Hasil pengujian ukuran 5 MB dengan Nv Websocket Client

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
1	1287	3885	9075
2	1311	3814	9200
3	1222	4092	9129
4	1251	3997	9097
5	1245	4016	9139
6	1220	4098	9145
7	1301	3843	9138
8	1307	3826	9154
9	1206	4146	9277
10	1202	4160	9109
11	1249	4003	9171
12	1198	4174	9058
13	1221	4095	9108
14	1200	4167	9195
15	1210	4132	9402
16	1174	4259	9237
17	1200	4167	9140
18	1179	4241	9163
19	1181	4234	9163

No.	Waktu (millisecond)	Kecepatan (KB/s)	Daya (mW)
20	1268	3943	9072
21	1272	3931	9186
22	1492	3351	9102
23	1296	3858	9128
24	1353	3695	9119
25	1404	3561	9142
26	1290	3876	9178
27	1658	3016	9192
28	1332	3754	9112
29	1372	3644	9064
30	1466	3411	9013

6.4.6 Hasil rata-rata kecepatan menggunakan Nv Websocket Client

Hasil rata-rata pengujian menggunakan Nv Websocket Client menunjukkan nilai yang terus naik dari segi kecepatan dan mengalami fluktuasi dari segi konsumsi daya yang dibutuhkan. Dari pengujian ini didapatkan kecepatan pengiriman pesan yang ditunjukkan pada Tabel 6.18.

Tabel 6.18 Hasil rata-rata pengujian menggunakan Nv Websocket Client

No.	Ukuran (KB)	Kecepatan (KB/s)	Daya (mW)
1	100 KB	262	9024
2	200 KB	273	9094
3	500 KB	405	9060
4	1 MB	661	9040
5	5 MB	3913	9147

6.5 Komparasi Antar *Library*

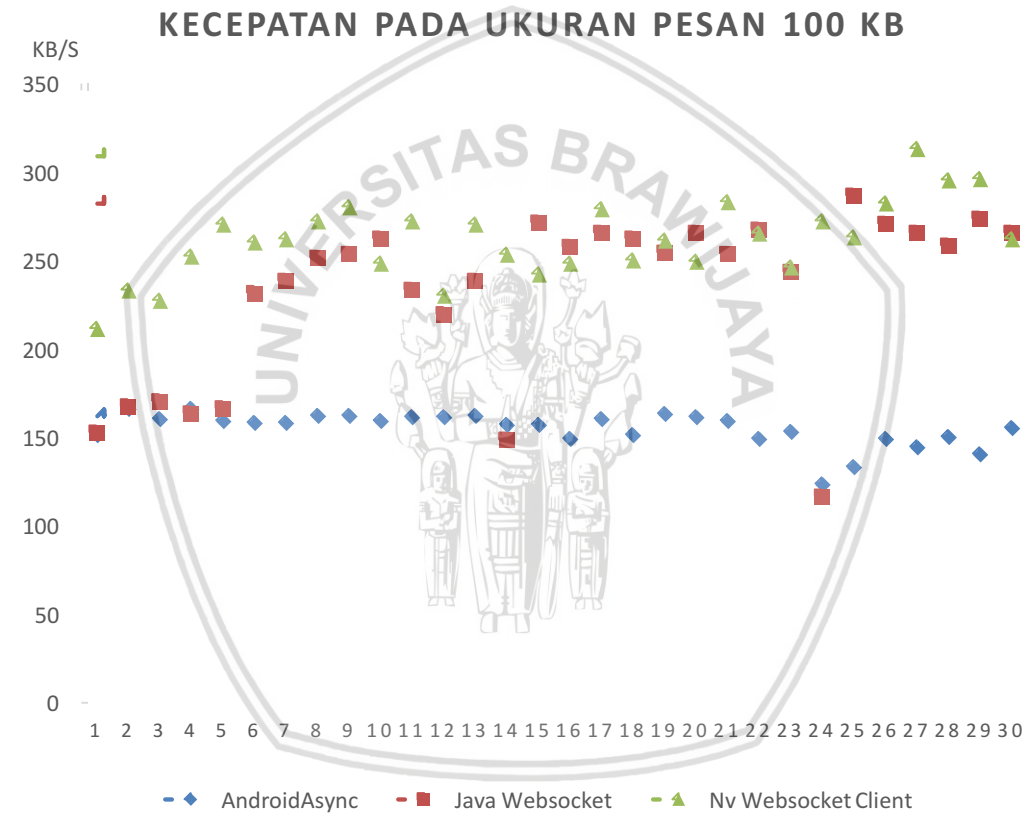
Komparasi dilakukan untuk melakukan perbandingan hasil pengujian pada setiap *library* Websocket klien. Komparasi antar *library* dibagi menjadi 4, yaitu komparasi kecepatan pengiriman pesan antar *library*, komparsi penggunaan daya antar *library*, komparasi rata-rata kecepatan pengiriman pesan dan komparsi rata-rata penggunaan daya.

6.5.1 Komparasi Kecepatan Pengiriman Pesan Antar *Library*

Pada subbab ini akan dibahas mengenai perbandingan kecepatan pengiriman pesan dari masing-masing *library* yang telah diuji yaitu AndroidAsync, Java Websocket, dan Nv Websocket Client. Perbandingan kecepatan dilakukan dengan mengambil hasil pengujian masing-masing *library* pada ukuran pesan yang sama, yaitu masing-masing pada ukuran 100 KB, 200 KB, 500 KB, 1 MB, dan 5 MB. Data akan disajikan dalam bentuk grafik sehingga memudahkan dalam melakukan perbandingan kecepatan pengiriman pesan.

6.5.1.1 Perbandingan kecepatan dengan ukuran 100 KB

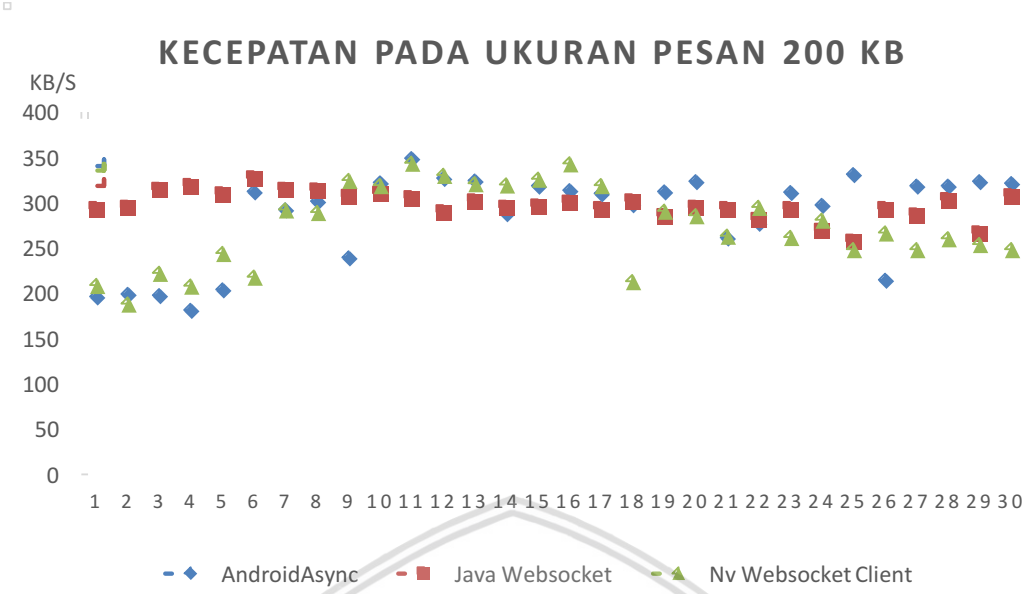
Hasil pengujian kecepatan pengiriman data masing-masing *library* dengan ukuran pesan yang dikirimkan sebesar 100 KB menunjukkan bahwa kecepatan pengiriman data AndroidAsync lebih lambat jika dibandingkan dengan Java Websocket dan juga Nv Websocket Cient. Sedangkan Java Websocket mengalami fluktuasi terhadap kecepatan pengiriman data antara kecepatan 150 KB/s dan 250 KB/s. Nv Websocket Client cenderung lebih stabil jika dibandingkan dengan Java Websocket dengan kecepatan di atas Java websocket dan AndroidAsync yaitu antara 200 KB/s sampai 300 KB/s. Grafik perbandingan kecepatan pengiriman pesan dari masing-masing *library* yang telah diuji dengan ukuran pesan 100 KB dapat dilihat pada Gambar 6.1.



Gambar 6.1 Kecepatan pada Ukuran Pesan 100 KB

6.5.1.2 Perbandingan kecepatan dengan ukuran 200 KB

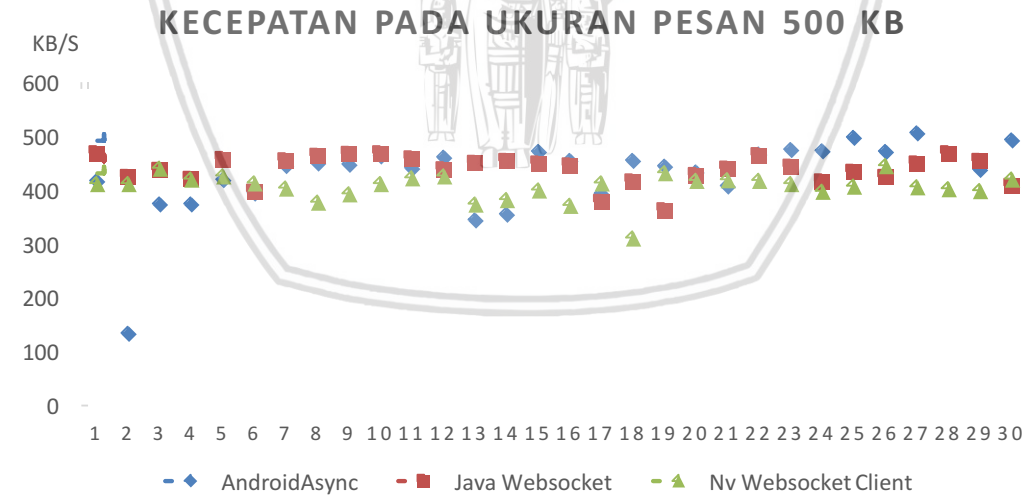
Hasil pengujian kecepatan pengiriman data masing-masing *library* dengan ukuran pesan 200 KB menunjukkan bahwa kecepatan pengiriman data AndroidAsync dan Nv Websocket Client cenderung fluktuatif dan mengalami kenaikan *trend* kecepatan pengiriman data. Sedangkan Java Websocket cenderung lebih stabil jika dibandingkan dengan AndroidAsync dan Nv Websocket Client akan tetapi mengalami penurunan *trend* kecepatan. Grafik perbandingan kecepatan pengiriman pesan dari masing-masing *library* yang telah diuji dengan ukuran pesan 200 KB dapat dilihat pada Gambar 6.2.



Gambar 6.2 Kecepatan pada Ukuran Pesan 200 KB

6.5.1.3 Perbandingan kecepatan dengan ukuran 500 KB

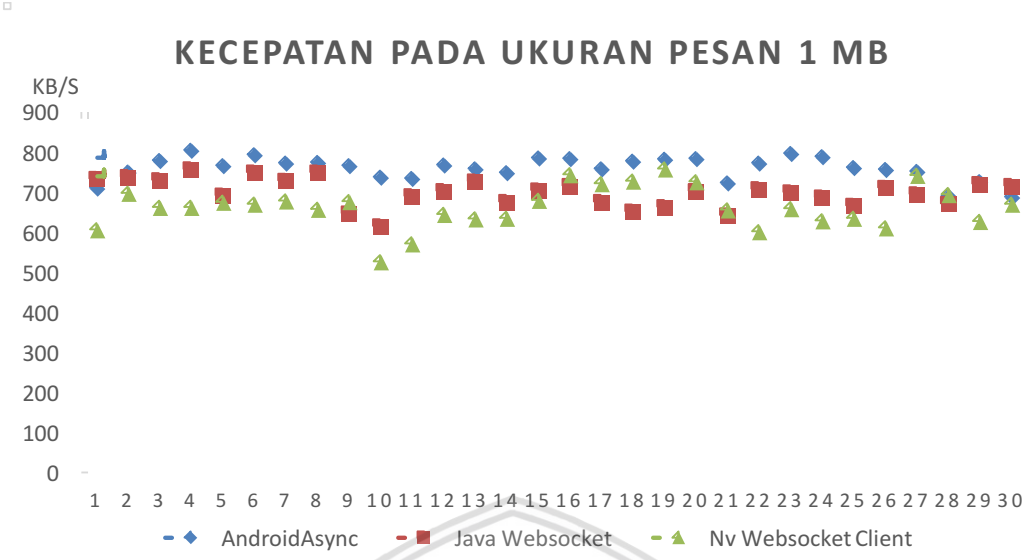
Hasil pengujian kecepatan pengiriman data masing-masing *library* dengan ukuran pesan 500 KB menunjukkan bahwa Java Websocket lebih unggul. Grafik perbandingan kecepatan pengiriman pesan dari masing-masing *library* yang telah diuji dengan ukuran pesan 500 KB dapat dilihat pada Gambar 6.3.



Gambar 6.3 Kecepatan pada Ukuran Pesan 500 KB

6.5.1.4 Perbandingan kecepatan dengan ukuran 1 MB

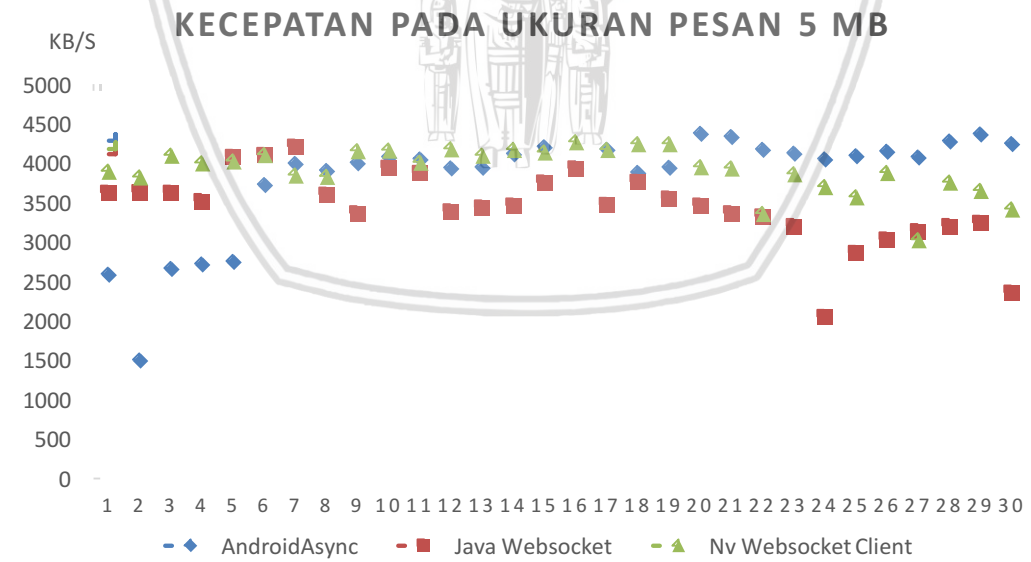
Hasil pengujian kecepatan pengiriman data masing-masing *library* dengan ukuran pesan 1 MB menunjukkan bahwa AndroidAsync lebih unggul. Grafik perbandingan kecepatan pengiriman pesan dari masing-masing *library* yang telah diuji dengan ukuran pesan 1 MB dapat dilihat pada Gambar 6.4.



Gambar 6.4 Kecepatan pada Ukuran Pesan 1 MB

6.5.1.5 Perbandingan kecepatan dengan ukuran 5 MB

Hasil pengujian kecepatan pengiriman data masing-masing *library* dengan ukuran pesan 5 MB menunjukkan bahwa Nv Websocket Client cenderung lebih unggul jika dibandingkan dengan AndroidAsync dan juga Java Websocket. Grafik perbandingan kecepatan pengiriman pesan dari masing-masing *library* yang telah diuji dengan ukuran pesan 5 MB dapat dilihat pada Gambar 6.5.



Gambar 6.5 Kecepatan pada Ukuran Pesan 5 MB

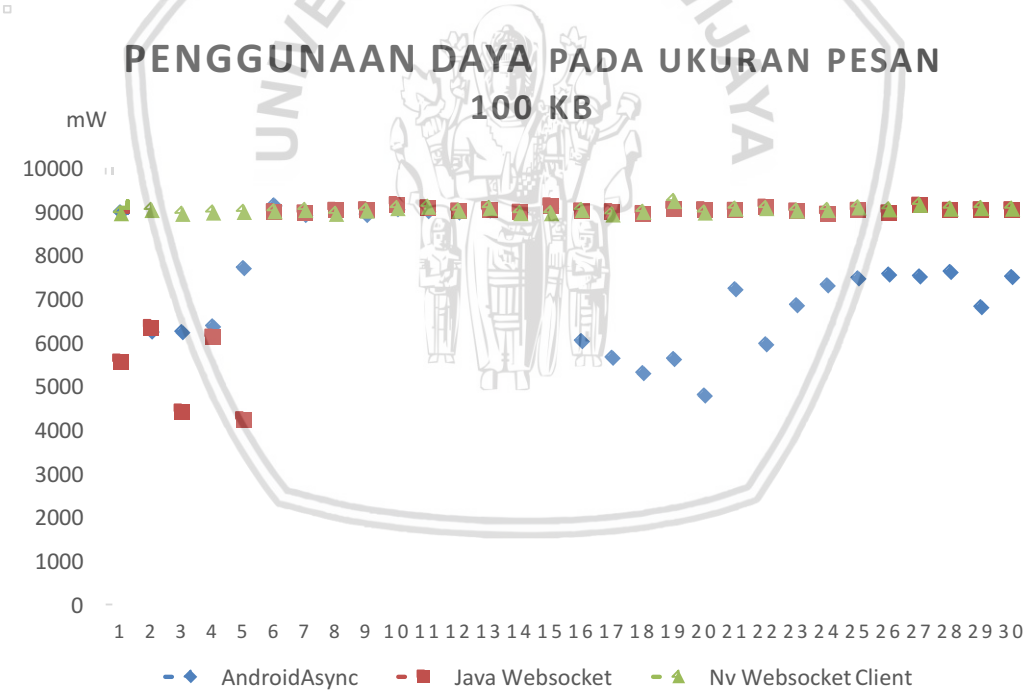
6.5.2 Komparasi Penggunaan Daya Antar *Library*

Pada subbab ini akan dibahas mengenai perbandingan konsumsi daya dari masing-masing *library* yang telah diuji yaitu AndroidAsync, Java Websocket, dan

Nv Websocket Client. Perbandingan konsumsi daya dilakukan dengan mengambil hasil pengujian masing-masing *library* pada ukuran pesan yang sama, yaitu masing-masing pada ukuran 100 KB, 200 KB, 500 KB, 1 MB, dan 5 MB. Data akan disajikan dalam bentuk grafik sehingga memudahkan dalam melakukan perbandingan kecepatan pengiriman pesan.

6.5.2.1 Perbandingan konsumsi daya dengan ukuran 100 KB

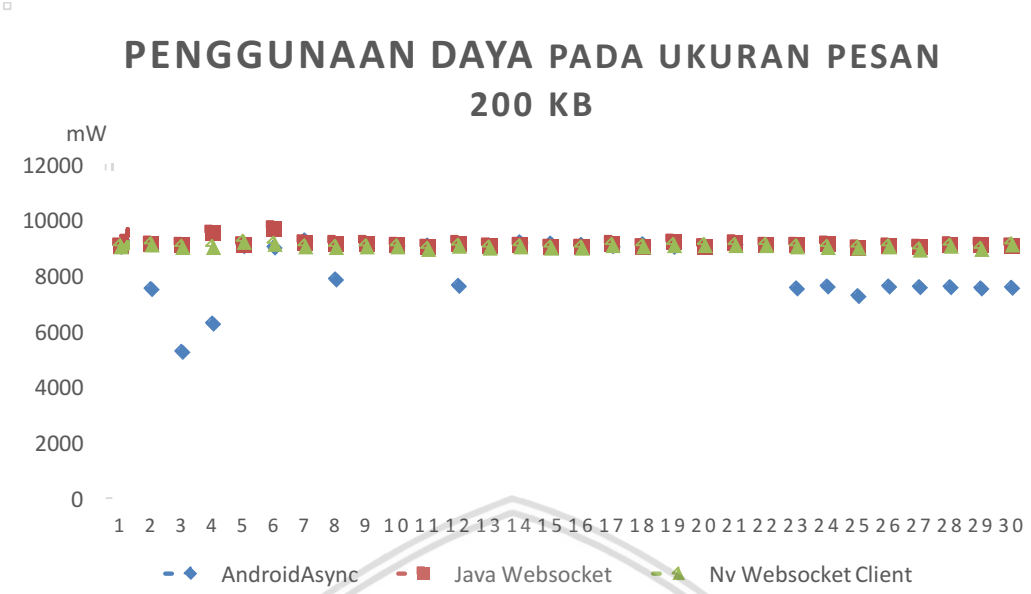
Hasil pengujian konsumsi daya pada masing-masing *library* dengan ukuran pesan 100 KB menunjukkan bahwa AndroidAsync cenderung lebih fluktuatif jika dibandingkan dengan Java Websocket dan Nv Websocket Client. Sedangkan Java Websocket mengalami fluktuasi di awal saja, setelah itu cenderung lebih stabil. Di sisi lain Nv Websocket Client cenderung lebih stabil jika dibandingkan dengan AndroidAsync dan Java Websocket. Dari sini dapat dilihat bahwa AndroidAsync lebih hemat dalam hal konsumsi daya jika dibandingkan dengan Java Websocket dan Nv Websocket Client. Grafik perbandingan konsumsi daya dari masing-masing *library* yang telah diuji dengan ukuran pesan 100 KB dapat dilihat pada Gambar 6.6.



Gambar 6.6 Penggunaan Daya Ukuran Pesan 100 KB

6.5.2.2 Perbandingan konsumsi daya dengan ukuran 200 KB

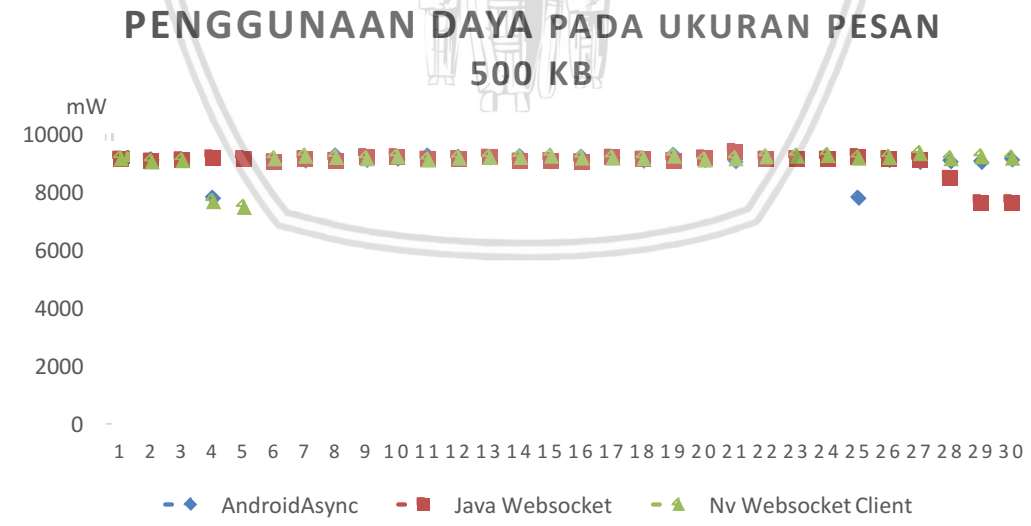
Hasil pengujian konsumsi daya pada masing-masing *library* dengan ukuran pesan 200 KB menunjukkan bahwa AndroidAsync lebih hemat dalam penggunaan daya jika dibandingkan dengan Java Websoket dan Nv Websocekt Client. Grafik perbandingan konsumsi daya dari masing-masing *library* yang telah diuji dengan ukuran pesan 200 KB dapat dilihat pada Gambar 6.7.



Gambar 6.7 Penggunaan Daya pada Ukuran Pesan 200 KB

6.5.2.3 Perbandingan konsumsi daya dengan ukuran 500 KB

Hasil pengujian konsumsi daya pada masing-masing *library* dengan ukuran pesan 500 KB menunjukkan bahwa Java Websocket lebih hemat dalam konsumsi daya. Grafik perbandingan konsumsi daya dari masing-masing *library* yang telah diuji dengan ukuran pesan 500 KB dapat dilihat pada Gambar 6.8.

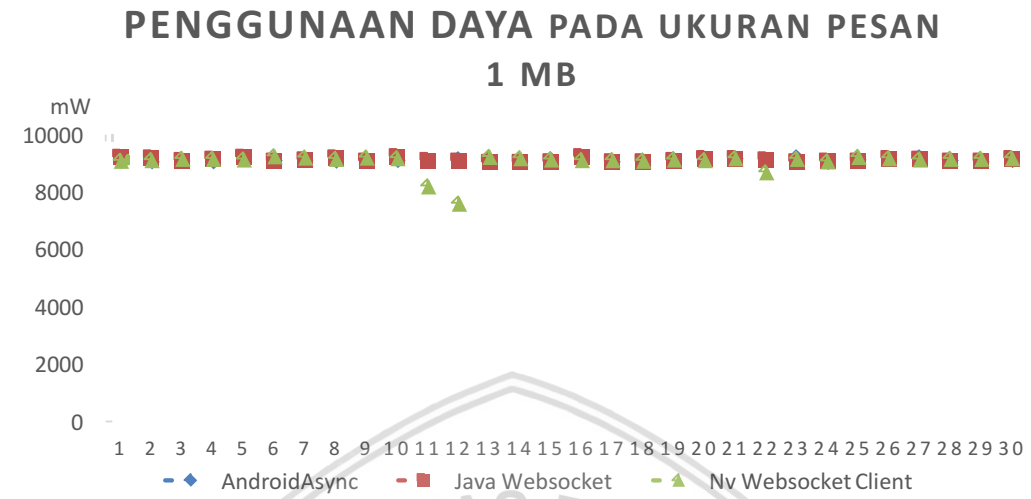


Gambar 6.8 Penggunaan Daya pada Ukuran Pesan 500 KB

6.5.2.4 Perbandingan konsumsi daya dengan ukuran 1 MB

Hasil pengujian konsumsi daya pada masing-masing *library* dengan ukuran pesan 1 MB menunjukkan bahwa Nv Websocket Client lebih hemat dalam

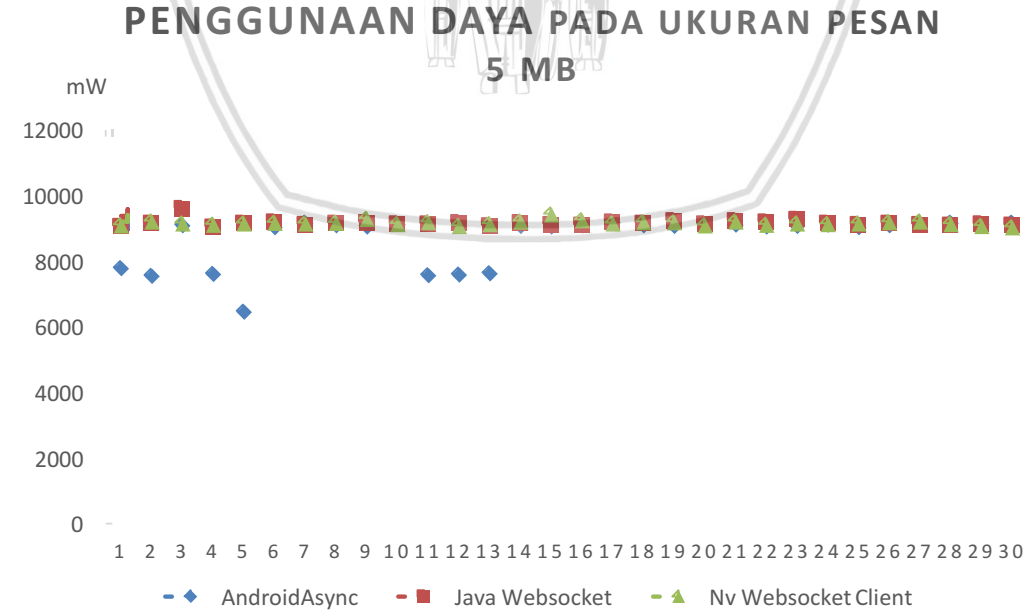
konsumsi daya. Grafik perbandingan konsumsi daya dari masing-masing *library* yang telah diuji dengan ukuran pesan 1 MB dapat dilihat pada Gambar 6.9.



Gambar 6.9 Penggunaan Daya pada Ukuran Pesan 1 MB

6.5.2.5 Perbandingan konsumsi daya dengan ukuran 5 MB

Hasil pengujian konsumsi daya pada masing-masing *library* dengan ukuran pesan 5 MB menunjukkan bahwa AndroidAsync lebih hemat dalam konsumsi daya jika dibandingkan dengan Java Websocket dan Nv Websocket Client. Grafik perbandingan konsumsi daya dari masing-masing *library* yang telah diuji dengan ukuran pesan 5 MB dapat dilihat pada Gambar 6.10.



Gambar 6.10 Penggunaan Daya pada Ukuran Pesan 5 MB

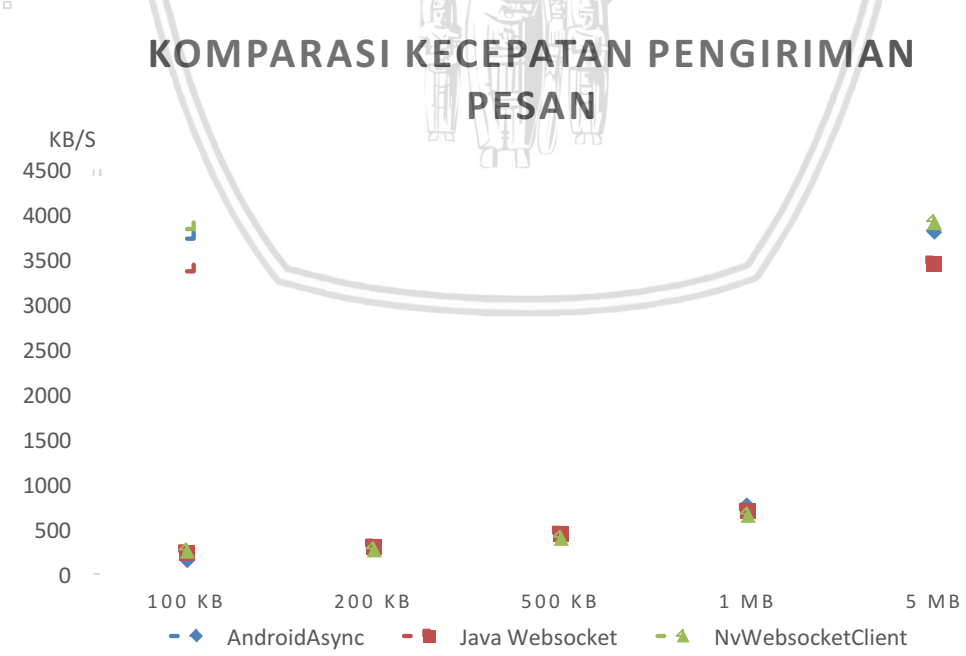
6.5.3 Komparasi Rata-rata Kecepatan Pengiriman Pesan

Berdasarkan pengujian yang telah dilakukan sebelumnya menghasilkan perbandingan kecepatan pengiriman pesan yang ditunjukkan Tabel 6.19.

Tabel 6.19 Komparasi Kecepatan Pengiriman Pesan (KB/s)

	AndroidAsync	Java Websocket	Nv Websocket Client
100 KB	155	232	262
200 KB	285	296	273
500 KB	427	440	405
1 MB	758	696	661
5 MB	3804	3439	3913
Rata-rata	**1085.8**	**1020.6**	**1102.8**

Komparasi pada Tabel 6.19 menunjukkan bahwa pada setiap *library* jika ukuran data yang dikirimkan semakin besar, maka kecepatan pengiriman datanya juga akan semakin meningkat. Dari ketiga *library* tersebut Nv Websocket Client secara konstan menunjukkan bahwa dengan perbedaan ukuran pesan yang dikrimkan Nv Websocket Client tetap lebih unggul dari pada *library* lain dalam hal kecepatan pengiriman pesan, sedangkan *library* AndroidAsync merupakan *library* yang paling lambat dalam melakukan pengiriman pesan. Grafik komparasi kecepatan pengiriman pesan ditunjukkan dalam Gambar 6.11.



Gambar 6.11 Komparasi Kecepatan Pengiriman Pesan

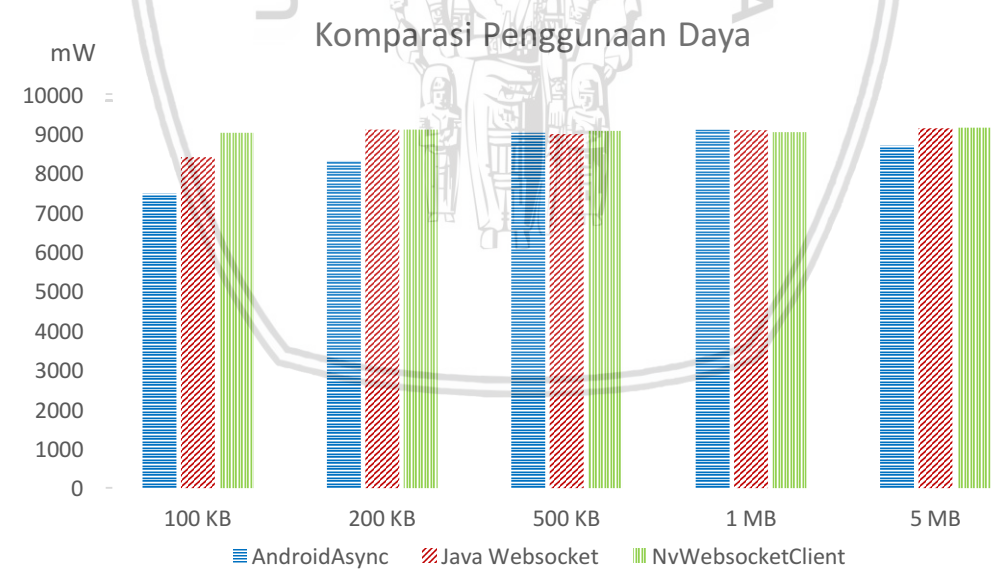
6.5.4 Komparasi Rata-rata Penggunaan Daya

Berdasarkan pengujian yang telah dilakukan sebelumnya menghasilkan perbandingan penggunaad daya yang ditunjukkan Tabel 6.20.

Tabel 6.20 Komparasi Penggunaan Daya (mW)

	AndroidAsync	Java Websocket	Nv Websocket Client
100 KB	7483	8402	9024
200 KB	8301	9106	9094
500 KB	9020	8982	9060
1 MB	9104	9092	9040
5 MB	8702	9138	9147
Rata-rata	**8522**	**8944**	**9064**

Komparasi pada Tabel 6.20 menunjukkan bahwa pada setiap *library* jika ukuran data yang dikirimkan semakin besar, maka penggunaan dayanya juga akan semakin meningkat, kecuali pada *library* Java Websocket. Dari ketiga *library* tersebut jika dirata-rata maka Nv Websocket Client merupakan *library* yang paling sedikit menggunakan daya, sedangkan *library* AndroidAsync paling banyak menggunakan data. Grafik komparasi kecepatan pengiriman pesan ditunjukkan dalam Gambar 6.12.



Gambar 6.12 Komparasi Penggunaan Daya

6.5.5 Analisis Hasil Pengujian

Dari hasil analisis yang telah dilakukan berdasarkan data-data yang telah diperoleh dari pengujian yang dilakukan, maka didapatkan hasil sebagai berikut:

1. Dari ketiga *library* yang telah diuji diperoleh hasil bahwa dari rata-rata pengiriman pesan Nv Websocket Client paling unggul dengan kecepatan

pengiriman pesan rata-rata melebihi dua *library* lainnya yaitu dengan kecepatan rata-rata 1102.8 KB/s. Hal ini disebabkan karena Nv Websocket Client selain mengimplementasikan RFC 6455 (The Websocket Protocol) juga mengimplementasikan RFC 7692 (Compression Extensions for WebSocket) yang di dalamnya ditambahkannya fungsionalitas kompresi pada protokol Websocket.

2. Sedangkan dari segi penggunaan daya rata-rata AndroidAsync merupakan *library* yang paling hemat dalam penggunaan daya jika dibandingkan dengan *library* lain yaitu menggunakan daya rata-rata sebesar 8522. Hal ini disebabkan karena AndroidAsync dikembangkan dengan basis NIO atau dikenal juga dengan nama Java NIO (New IO) yang merupakan alternatif dari Java IO. Java NIO menawarkan cara lain untuk bekerja dengan IO jika dibandingkan dengan IO API standar.
3. Dari hasil komparasi rata-rata penggunaan daya terlihat bahwa selisih penggunaan daya lebih terlihat, sedangkan pada ukuran data (pesan yang dikirimkan) yang semakin besar selisih penggunaan daya cenderung lebih tidak terlihat.

BAB 7 PENUTUP

Bagian ini memuat kesimpulan dan saran terhadap skripsi. Kesimpulan dan saran disajikan secara terpisah, dengan penjelasan sebagai berikut:

7.1 Kesimpulan

Berdasarkan perancangan, implementasi, pengujian dan analisis terhadap *library* Websocket klien pada Android pada studi kasus aplikasi perpesanan, maka diperoleh kesimpulan sebagai berikut:

1. Dari ketiga *library* yang telah diuji diperoleh hasil bahwa dari rata-rata pengiriman pesan Nv Websocket Client paling unggul dengan kecepatan pengiriman pesan rata-rata melebihi AndroidAsync dan Java Websocket. Hal ini disebabkan karena Nv Websocket Client selain mengimplementasikan RFC 6455 (The Websocket Protocol) juga mengimplementasikan RFC 7692 (Compression Extensions for WebSocket) yang ditambahkan fungsionalitas kompresi pada protokol Websocketnya. Sedangkan dari segi penggunaan daya rata-rata AndroidAsync merupakan *library* yang paling hemat dalam penggunaan daya jika dibandingkan dengan Java Websocket dan Nv Websocket Client. Hal ini disebabkan karena AndroidAsync dikembangkan dengan basis NIO atau dikenal juga dengan nama Java NIO (New IO) yang merupakan alternatif dari Java IO. Java NIO menawarkan cara lain untuk bekerja dengan IO yang lebih efisien jika dibandingkan dengan IO API standar.
2. Dari segi kecepatan pengiriman pesan dengan ukuran kecil (100 KB) perbedaan tidak terlalu terlihat, perbedaan akan terlihat lebih jelas ketika data yang dikirimkan berukuran besar (5 MB). Untuk target dengan spesifikasi perangkat keras yang rendah disarankan menggunakan AndroidAsync karena dari segi penggunaan daya merupakan *library* yang paling hemat sedangkan dari segi kecepatan menduduki peringkat kedua. Sedangkan untuk target perangkat keras dengan spesifikasi standar atau di atas rata-rata disarankan untuk menggunakan Nv Websocket Client, karena meskipun menghabiskan daya yang lebih banyak jika dibandingkan dengan AndroidAsync dan Java Websocket, Nv Websocket Client dari rata-rata kecepatan pengiriman datanya lebih cepat jika dibandingkan dengan *library* lain yang telah diuji.

7.2 Saran

Penelitian ini dapat dikembangkan dengan lebih baik dengan saran sebagai berikut:

1. Menambahkan parameter pengujian berupa penggunaan *memory,* penggunaan CPU, penggunaan *disk* (penyimpanan). Dengan lebih banyak aspek yang dipertimbangkan sehingga diharapkan akan dapat

memperoleh hasil yang lebih akurat. Pengujian juga dapat dilakukan dengan menggunakan Trepn Profiler.

2. Melakukan pengujian dengan menggunakan jaringan *public* untuk dapat mengetahui performansi *library* jika menggunakan jaringan internet yang sebenarnya. Sehingga diperoleh hasil yang lebih nyata sesuai dengan kondisi yang benar-benar terjadi. Hal ini dapat dilakukan dengan cara memindahkan Websocket *server* ke dalam *server public* sehingga dapat diakses di manapun.
3. Menambahkan *library* lain yang digunakan untuk pengujian.

DAFTAR PUSTAKA

AndroidAsync. (2016). AndroidAsync. [online] Tersedis di: <https://koush.com/AndroidAsync> [diakses 1 maret 2017]

Bedregal, Jose Carlos Valdivia., Gutierrez, Eveling Gloria Castro., dan Robert E. Arisaca M. (2013). Optimizing Energy Consumption per Application in Mobile Devices. IEEE.

Bottle. (2017). Bottle: Python Web Framework. [online] Tersedia di: <https://bottlepy.org/docs/0.12/> [diakses 9 April 2017]

Cisco. (2014). [online] Tersedia di: <Why are mail attachments bigger than the original file?> [diakses 7 Desember 2017]

Dinh, Pham Cao., dan Boonkrong, Sirapat. (2013). The Comparison Of Impacts To Android Phone Battery Between Polling Data And Pushing Data. ResearchGate.

Ed, L. Stout., Moffitt, J., Cestari, E. (2014). An Extensible Messaging and Presence Protocol (XMPP) Subprotocol forWebSocket. RFC-7395

Ejabberd. (2017). XMPP use cases. [online] Tersedia di: <https://docs.ejabberd.im/xmpp/> [diakses 14 Februari 2017]

Fette, I. (2011). The WebSocket Protocol. RFC 6455

Google Play. (2017). Top Free in Android Apps. [online] Tersedia di: <https://play.google.com/store/apps/collection/topselling_free> [diakses 11 Februari 2017]

Gordon, Mark., Zhang, Lide., Tiwana, Birjodh. (2009). A Power Monitor for Android-Based Mobile Platforms. [online] Tersedia di: <http://ziyang.eecs.umich.edu/projects/powertutor/index.html> [diakses 12 April 2017]

Grossman, Lev. (2007). Invention Of the Year: The iPhone (INVENTION OF THE YEAR). [online] Tersedia di: <http://content.time.com/time/specials/2007/article/0,28804,1677329_1678542_1677891,00.html> [diakses 11 Februari 2017]

Hardianto, Fitri. (2015). Aplikasi Grupchat Di Android Menggunakan Websocket. Universitas Muhamadiah Surakarta.

Highscalability. (2014). The WhatsApp Architecture Facebook Bought For $19 Billion. [online] Tersedia di: <http://highscalability.com/blog/2014/2/26/the-whatsapp-architecture-facebook-bought-for-19-billion.html> [diakses 15 Februari 2017]

Hildebrand, J., Kaes, C., dan Waite, D. (2009). Jabber HTTP Polling, Joe Hildebrand. [online] Tersedia di: <https://xmpp.org/extensions/xep-0025.html> [diakses 14 Februari 2017]

Hows, David., Membrey, Peter., Plugge, Eelco., Hawkins, Tim. (2015). The Definitive Guide to MongoDB. Apress.

IDC. (2016). Market Share. [online] Tersedia di: <http://www.idc.com/promo/smartphone-market-share/os> [diakses 11 Februari 2017]

Java-Websocket. (2014). Java-WebSocket.[online] Tersedia di: <http://tootallnate.github.io/Java-WebSocket/> [diakses 2 maret 2017]

JSON. (2016). Introducing JSON. [online] Tersedia di : <http://www.json.org/> [Diakses 11 Maret 2017]

Keong, Ching Kin., Wei, Koh Tieng., Ghani, Abdul Azim Abd., Sharif, Khaironi Yatim. (2015). Toward using Software Metrics as Indicator to Measure Power Consumption of Mobile Application: A Case Study. IEEE

Lombardi, Andrew. (2015). WebSocket. O'Reilly Media, Inc.

Meola, Andrew. (2016). WhatsApp is the most popular chat app in more than half the world. [online] Tersedia di: <http://www.businessinsider.com/whatsapp-is-the-most-popular-chat-app-in-more-than-half-the-world-2016-5?IR=T&r=US&IR=T> [diakses 11 Februari 2017]

MongoDB. (2017). What is MongoDB?. [online] Tersedia di: <https://www.mongodb.com/what-is-mongodb> [diakses 9 April 2017]

P, Andrias Yudianto. (2017). Pengembangan Push Notification Menggunakan Websocket. Universitas Brawijaya.

Paterson, I., dkk. (2016). [online] Tersedia di: <https://xmpp.org/extensions/xep-0124.html> [diakses 14 Februari 2017]

Pithikos. (2017). Websocket Server. [online] Tersedia di: <https://github.com/Pithikos/python-websocket-server> [diakses 9 April 2017]

Prasetyo, Audi Eka. (2015). Aplikasi Chatting Terbaik di Indonesia? LINE, WhatsApp, atau BBM?. [online] Tersedia di: <https://id.techinasia.com/talk/aplikasi-chatting-terbaik-di-indonesia-line-bbm-whatsapp> [diakses 11 Februari 2017]

Pratama, Aditya Hadi. (2016). Grab Hadirkan Fitur Chat dengan Pengemudi di dalam Aplikasinya. [online] Tersedia di: <https://id.techinasia.com/grab-luncurkan-fitur-chat-grabchat> [diakses 11 Februari 2017]

Qualcomm. (2017). Trepn Power Profiler. [online] Tersedia di: <https://developer.qualcomm.com/software/trepn-power-profiler> [diakses 12 April 2017]

Realm. (2017). Realm Mobile Platform Overview. [online] Tersedia di: <https://realm.io/docs/get-started/overview/> [diakses 12 April 2017]

RIZKY, MOHAMAD. (2016). Perkembangan smartphone. [online] Tersedia di: <https://id.techinasia.com/talk/kejadian-penting-perkembangan-smartphone> [diakses 11 Februari 2017]

Vagrant. (2017). Introduction to Vagrant. [online] Tersedia di: <https://www.vagrantup.com/intro/index.html> [diakses 12 April 2017]

XMPP. (2017). History of XMPP. [online] Tersedia di: <http://xmpp.org/about/history.html> [diakses 13 Februari 2017]