BAB I
PENDAHULUAN

1.1 Latar Belakang

Jejaring sosial adalah kegiatan menjalin hubungan dengan orang lain melalui *social networking sites* atau situs jejaring sosial yang ada di *Internet*. Kegiatan ini sekarang sudah menjadi bagian dari gaya hidup di semua kalangan masyarakat. Situs - situs jejaring sosial menyediakan kumpulan layanan dan cara yang beragam bagi penggunanya untuk dapat berinteraksi satu sama lain seperti *chat*, *messaging*, *microbloging*, diskusi grup, dan lain-lain.

Kampus merupakan salah satu tempat dimana sebuah layanan aplikasi jejaring sosial sangat dibutuhkan baik itu antara sesama mahasiswa, sesama dosen maupun antara mahasiswa dengan dosennya. Jejaring sosial yang ada pada kampus diharapkan juga harus dapat membantu kelancaran aktivitas akademik di kampus seperti tersedianya layanan untuk menemukan lokasi keberadaan dosen atau mahasiswa secara cepat dan akurat. Layanan ini biasa disebut dengan *geotagging*. *Geotagging* sendiri sebenarnya adalah proses menambahkan metadata identifikasi geografis ke berbagai media dan merupakan bentuk dari metadata geospasial [LAR-11].

Salah satu keuntungan yang diperoleh dengan adanya layanan *geotagging* di dalam kampus adalah seorang mahasiswa tidak perlu lagi bingung jika suatu saat ingin mencari dan bertemu dengan dosennya. Kendala yang ada saat ini adalah jejaring sosial yang ada belum menyediakan layanan untuk melakukan *geotagging* secara otomatis (*autogeotagging*). Berdasarkan ulasan tersebut, maka diusulkan sebuah aplikasi jejaring sosial internal di dalam kampus yang memiliki fitur untuk meng-*update* secara otomatis posisi *device* seorang *user* dan mengirimkan lokasi *user* tersebut ke GPS (*Global Positioning System*). Aplikasi ini diharapkan mampu memudahkan seorang *user* dalam mengetahui lokasi terakhir dimana *user* lain berada secara *real time.*

Salah satu cara untuk melakukan *autogeotagging* adalah memanfaatkan *Global Positioning System* (GPS) yang tertanam pada perangkat *mobile* atau *smartphone* android. Android mempunyai prospek tingkat penetrasi pasar yang

sangat menjanjikan dimana diperkirakan dalam waktu 6 bulan pertama penjualan saja, akan terdapat 4,2 juta *handset* android di seluruh dunia [WIR-11]. Hal ini berarti hampir semua pengguna android dapat dengan cepat mengakses *account* jejaring sosial miliknya kapan saja dan dimana saja melalui perangkat *mobile*-nya serta berpotensi dapat dilacak *via* perangkat *mobile*-nya.

1.2 Rumusan Masalah

Berdasarkan uraian pada bagian latar belakang, maka masalah yang ada dapat dirumuskan sebagai berikut :

1. Jejaring sosial yang ada pada saat ini masih belum ada yang menyediakan layanan *autogeotagging*.
2. Merancang sebuah aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android.
3. Implementasi aplikasi jejaring sosial kampus berbasis GPS sehingga dapat diintegrasikan pada *smartphone* android.
4. Pengujian dan analisis kerja sistem aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android.

1.3 Batasan Masalah

Batasan masalah pengembangan aplikasi perangkat lunak dalam skripsi ini, adalah:

1. Pembahasan difokuskan pada pembuatan aplikasi *client* jejaring sosial berbasis GPS pada *Smartphone* Android.
2. Sistem operasi yang digunakan adalah Microsoft Windows Vista 32bit.
3. *Platform* pengembangan yang digunakan adalah JSE 6 (Java *Standard Edition* 6).
4. Sistem android yang digunakan adalah 2.3 (gingerbread).
5. Pengujian Sistem dilakukan pada lingkungan Kampus Teknik Informatika Universitas Brawijaya.
6. Pengujian performa difokuskan pada nilai *transfer rate* dan *request per second* sebagai *throughput* dari jumlah *request* yang dikirimkan ke *server*.

1.4 Tujuan

Tujuan penulisan tugas akhir ini adalah merancang dan mengembangkan sebuah aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android yang memiliki layanan *autogeotagging* sehingga dapat membantu penggunanya dalam mencari informasi posisi dan lokasi dari pengguna lain.

1.5 Manfaat

a. Bagi penulis

1. Menerapkan ilmu yang telah diperoleh dari Teknik Informatika konsentrasi Rekayasa Perangkat Lunak Universitas Brawijaya.
2. Mendapatkan pemahaman tentang perancangan dan pengembangan aplikasi android yang menggunakan Java sebagai *platform* pengembangan.

b. Bagi pengguna

1. Membantu menyediakan sarana hubungan sosial antar mahasiswa, antar dosen maupun antara mahasiswa dengan dosen sekaligus memudahkan pengguna dalam mengetahui lokasi dan posisi pengguna lain melalui perangkat *mobile* android.

1.6 Sistematika Penulisan

Sistematika penulisan laporan skripsi ini adalah sebagai berikut :

BAB I Pendahuluan

BAB ini memuat latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi pembahasan, dan sistematika penulisan.

BAB II Dasar teori

Dasar teori berisi tentang teori dasar dan teori penunjang yang berkaitan dengan Jejaring sosial, android, *web service,* GPS (*Global Positioning System*), Rekayasa Perangkat Lunak, *UML (Unified Modelling Language), Database* dan pengujian perangjat lunak .

BAB III Metodologi

Pada BAB ini akan dibahas metode yang digunakan dalam penulisan yang terdiri dari studi literatur, analisis kebutuhan

perangkat lunak, perancangan perangkat lunak, implementasi perangkat lunak, pengujian perangkat lunak dan analisis hasil dan pengambilan kesimpulan.

BAB IV Analisis kebutuhan dan Perancangan

BAB ini membahas analisis kebutuhan dan perancangan sistem Aplikasi Jejaring Sosial Kampus Berbasis GPS pada *Smartphone* Android sebagai dasar tahap implementasi yang akan dilakukan sesudahnya.

BAB V Implementasi dan Pembahasan

BAB ini berisi tentang pembahasan hasil dari perancangan serta implementasi dari sistem dan aplikasi yang akan di bangun.

BAB VI Pengujian

BAB ini membahas proses dan hasil pengujian terhadap sistem yang telah direalisasikan.

BAB VII Kesimpulan dan Saran

BAB ini memuat kesimpulan serta saran untuk pengembangan sistem lebih lanjut.

BAB II
DASAR TEORI

Pada bab ini dijelaskan tentang dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai " Rancang Bangun Aplikasi Jejaring Sosial Kampus Berbasis GPS pada *Smartphone* Android ". Beberapa dasar teori yang dimaksud diantaranya adalah jejaring sosial, android, *web service*, *Global Positioning System (GPS)*, rekayasa perangkat lunak, UML (*Unified Modelling Language*) dan pengujian perangkat lunak.

2.1 Jejaring Sosial

Istilah jaringan sosial mengacu kepada struktur dan jumlah orang dan kelompok dengan siapa anda menjalin kontak atau memandang diri anda sendiri berada di dalam kontak. Jejaring sosial dapat dideskripsikan dalam arti kualitas struktural dan fungsional seperti [ROB-09:105-106] :

a. Kualitas struktural
    1. Besaran - jumlah keseluruhan orang yang ada di dalam jaringan.
    2. Komposisi keberagaman kelompok atau kerumunan di dalam jaringan - seperti anggota rumah tangga, saudara, teman, tetangga dan seterusnya.
    3. Frekuensi - seberapa sering orang di dalam jejaring berinteraksi satu sama lain.
    4. Stabilitas - berapa lama orang di dalam jejaring sudah mengenal satu sama lain, konsistensi komposisi jejaring sepanjang waktu.
    5. Intensitas - kekuatan relasi di dalam jaringan.
    6. Penyebaran (*dispersion*) - kenyamanan yang dengannya anggota jaringan dapat berkontak dan berkomunikasi satu sama lain.
b. Kualitas fungsional
    1. Velensi (*valence*) - kualitas emosional dari relasi jaringan.
    2. Multipleksitas (*multiplexity*) - sejauh mana relasi jaringan melayani lebih dari satu fungsi atau memberikan lebih dari satu jenis dukungan.

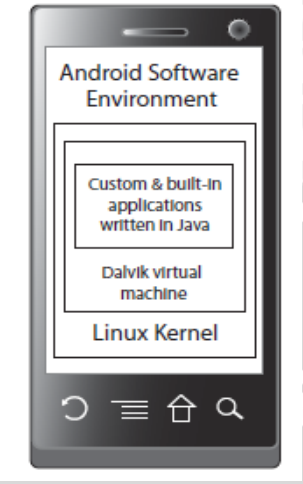
3. Resiprositas (*reciprocity*) - Jumlah yang diberikan dan yang diterima, sejauh mana dukungan yang diberikan seimbang dengan dukungan yang diterima.
4. Homogenitas (*homogenity*) - sejauh mana anggota jaringan memiliki sifat yang sama.

Situs jejaring sosial adalah layanan berbasis *web* yang memungkinkan individu untuk membangun profil publik atau semi-publik dalam sistem yang terbatas dan melihat daftar dari pengguna lain serta dengan siapa mereka berhubungan. Situs jejaring sosial menyediakan kumpulan cara yang beragam bagi pengguna untuk dapat berinteraksi seperti *chat*, *messaging*, *email*, video, *chat* suara, *share file*, blog, diskusi grup, dan lain-lain. Umumnya jejaring sosial memberikan layanan untuk membuat biodata dirinya. Pengguna dapat meng-*upload* foto dirinya dan dapat menjadi teman dengan pengguna lainnya. Beberapa jejaring sosial memiliki fitur tambahan seperti pembuatan grup untuk dapat saling *sharing* didalamnya [BOY-08:211-214].

2.2 Android

Android adalah *platform* perangkat lunak dari Google dan Open Handset Alliance yang dibangun untuk perangkat *mobile*. Android mencakup *kernel* berbasis Linux OS, UI yang kaya, kode *libraries*, kerangka aplikasi, dukungan multimedia, dan masih banyak lagi termasuk fungsi telepon. Android hanya sebuah *software* dan menggunakan *kernel* Linux sebagai antarmuka dengan *hardware*. Komponen - komponen sistem operasi yang mendasari ditulis dalam bahasa C atau C++ sedangkan aplikasi pengguna yang dibangun untuk android ditulis dalam Java menggunakan Android *Software Development Kit* (SDK) [ABL-11: 4].

Salah satu fitur dari *platform* android adalah bahwa tidak ada perbedaan antara aplikasi *built-in* dengan aplikasi-aplikasi yang dibuat dengan SDK. Pihak *developer* dapat membuat aplikasi yang mampu untuk mengakses semua sumber daya yang tersedia pada perangkat android [ABL-11:4]. Hubungan antara aplikasi android dan *hardware* perangkat android itu berjalan ditunjukkan dalam Gambar 2.1.



Gambar 2.1 Hubungan antara android dengan *hardware*
Sumber : [ABL-11: 4]

Fitur yang paling menonjol pada android adalah *platform* ini *open source*, unsur yang hilang atau belum tersedia dapat disediakan oleh komunitas pengembang secara global. Android berbasis *kernel* Linux tidak dilengkapi dengan lingkungan *shell* yang canggih, tetapi karena *platform* ini terbuka, *shell* yang tidak disediakan dapat ditulis dan di-*install* secara manual pada perangkat. *Multimedia codec* dapat disediakan oleh pengembang pihak ketiga dan tidak perlu bergantung pada Google atau siapapun untuk menambah fungsionalitas baru. Itulah kekuatan sebuah *platform open source* yang dibawa ke pasar *smartphone* [ABL-11:4].

2.2.1 Fitur –fitur Android :

Mark Murpy [MUR-10:3] menyebutkan bahwa android memiliki beberapa fitur yang akan memudahkan pengembang (*developer*) dalam mengembangkan sebuah aplikasi. Fitur-fitur tersebut antara lain adalah :

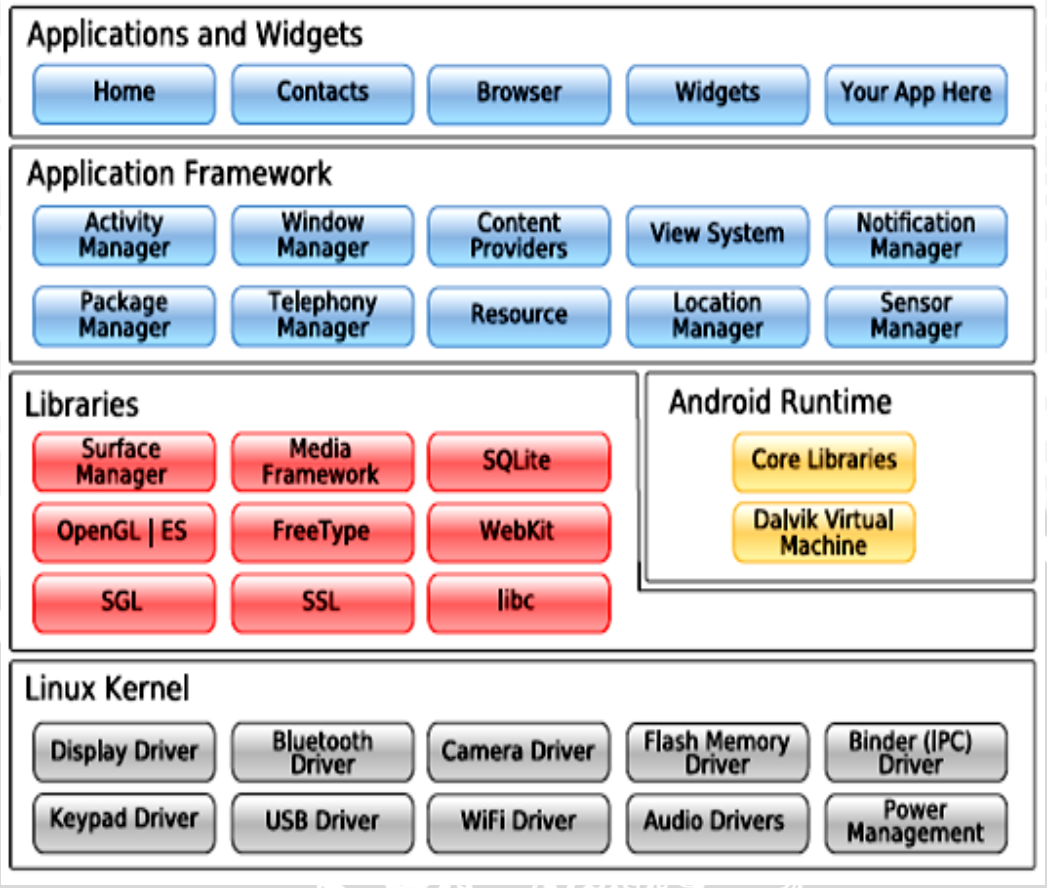
1. *Storage* : Fitur untuk membuat paket *file* data dengan sebuah aplikasi untuk hal-hal yang tidak berubah, seperti ikon atau *file* bantuan serta membuat sedikit ruang pada perangkat itu sendiri, untuk *database* atau *file* yang mengandung *user-entered* atau mengambil data yang

dibutuhkan oleh aplikasi anda. Jika pengguna menyediakan penyimpanan masal seperti *SD Card*, fitur ini mengijinkan pengembang untuk membaca dan menulis *file* yang ada di sana jika diperlukan.

2. *Network* : Perangkat android pada umumnya akan terintegrasi dengan *Internet*, melalui satu media komunikasi atau lebih. Fitur ini memberikan keuntungan dari akses *Internet* pada setiap tingkatan yang dibutuhkan, dari soket *raw Java* semua jalur tergantung pada sebuah *built-in* berbasis *WebKit*
3. *Multimedia*: Perangkat android memiliki kemampuan untuk memutar dan merekam audio dan video. Suatu perangkat dapat memiliki kemampuan yang bervariasi secara spesifik dari perangkat yang lainnya. Fitur ini memungkinkan untuk mempelajari kemampuan suatu perangkat, dan kemudian mengambil keuntungan dari kemampuan multimedianya sesuai dengan yang diinginkan.
4. *Global Positioning System* (GPS) : Perangkat android hampir semuanya memiliki akses ke penyedia lokasi seperti GPS yang dapat memberitahu suatu aplikasi dimana posisi perangkat ini di muka bumi. Fitur ini memberikan kemudahan untuk menampilkan peta atau mengambil keuntungan dari data lokasi, seperti pelacakan gerakan perangkat jika perangkat telah dicuri.
5. *Phone Services* : Perangkat android pada dasarnya adalah telepon, suatu perangkat lunak dapat melakukan panggilan, mengirim dan menerima *Short Message Service* (SMS), dan segala sesuatu yang mampu dilakukan oleh teknologi telepon modern [MUR-10:4].

2.2.2 Arsitektur Android

Ed Burnette dalam bukunya yang berjudul *Hello, Android* [BUR-10] menjelaskan bahwa arsitektur sistem operasi android terbagi menjadi beberapa bagian utama yaitu *applications and widgets, applications framework, libraries, android runtime* dan *kernel* linux. Arsitektur android secara detail ditunjukkan dalam Gambar 2.2.



Gambar 2.2 Arsitektur sistem android

Sumber : [BUR-10:31]

a. Aplikasi (*Applications and Widgets*)

Android akan dilengkapi dengan seperangkat aplikasi inti seperti *client email*, program SMS, Kalender, peta, *browser*, kontak, dan lain-lain. Semua aplikasi ini ditulis menggunakan bahas pemrograman Java [RAN-12:210].

b. *Applications Framework*

Android menyediakan sebuah *platform* pengembangan yang terbuka, sistem ini menawarkan kemampuan untuk membangun aplikasi yang sangat kaya dan inovatif. Pengembang bebas untuk mengakses perangkat keras, informasi akses lokasi, menjalankan *background services*, mengatur alarm, tambahkan pemberitahuan ke *statusbar*, dan banyak lagi [BUR-10:33].

Pengembang memiliki akses penuh ke *framework* API yang sama yang digunakan oleh aplikasi inti. Arsitektur aplikasi dirancang untuk menyederhanakan penggunaan kembali komponen, aplikasi apapun dapat

mempublikasikan kemampuannya dan aplikasi lain kemudian dapat menggunakan kemampuan mereka (diatur oleh batasan keamanan yang diberlakukan oleh *framework*). Mekanisme ini mengizinkan *user* mengganti komponen [BUR-10:34].

Komponen-komponen yang termasuk didalam *Applications Frameworks* adalah sebagai berikut [MEI-10:15]:

1. Satu set *Views* yang kaya dan *extensible* yang dapat digunakan untuk membangun sebuah aplikasi, termasuk *list*, *grid*, *text box*, *button*, dan bahkan *web browser* yang sifatnya *embeddable*.
2. *Content Providers* memungkinkan aplikasi untuk mengakses data dari aplikasi lain (seperti *Contact*) atau untuk berbagi data mereka sendiri.
3. *Resource Manager*, menyediakan akses sumber daya non-kode seperti *string* lokal, gambar, dan *layout files*.
4. *Notifications Manager* memungkinkan semua aplikasi untuk menampilkan *custom alert* pada *status bar*.
5. *Activity Manager*, mengelola siklus hidup aplikasi dan menyediakan *backstack* navigasi umum.

c. *Libraries*

Libraries adalah *layer* dimana fitur-fitur android berada, biasanya para pembuat aplikasi android mengakses *libraries* untuk menjalankan aplikasinya. Android mencakup set *library* C atau C++ yang digunakan oleh berbagai komponen dari sistem android. Beberapa library inti dari android adalah seperti tercantum dibawah ini [SMI-11:5]:

1. *System C library* - implementasi turunan BSD dari standar *library* sistem *C* (libc), dirancang untuk perangkat yang berbasis *embedded* Linux.
2. *Media libraries* - berdasarkan pada OpenCORE *PacketVideo*, *librabies* ini mendukung pemutaran (*playback)* dan perekaman berbagai format audio dan video, serta *file* gambar statis, seperti MPEG4, H.264, MP3, AAC, AMR, JPG, dan PNG.

3. *Surface Manager* - mengelola akses ke subsistem layar dan menghaluskan lapisan komposit gambar 2D dan 3D dari beberapa aplikasi.
4. *LibWebCore* - mesin *web browser* yang modern yang menunjang *browser* android dan *web view* yang tertanam.
5. *SGL* - Mesin dasar gambar 2D.
6. *3D Libraries* - implementasi berdasarkan API dari OpenGL ES 1.0, libraries ini menggunakan akselerasi 3D *hardware* (jika tersedia).
7. FreeType - *bitmap* dan *vektor rendering.*
8. *SQLite* - sebuah mesin *database* relasional yang kuat dan ringan tersedia untuk semua aplikasi.

d. *Android Runtime*

Android mencakup seperangkat *library* inti yang menyediakan sebagian besar fungsionalitas yang tersedia di *library* inti dari bahasa pemrograman Java. Setiap aplikasi Android berjalan dalam prosesnya sendiri, dengan *instance* sendiri dari *Dalvik Virtual Machine* (DVM) . Dalvik ditulis supaya sebuah perangkat dapat menjalankan beberapa VM secara efisien. DVM mengeksekusi file dalam format Dalvik *executable* (.dex) yang dioptimalkan untuk minimal memori [RAN-12:210]. DVM berbasis *register*, dan menjalankan *class* yang dikompilasi oleh *compiler* bahasa Java yang telah diubah menjadi format dex oleh "dx" *tool*. DVM bergantung pada *kernel* Linux untuk fungsionalitas dasar seperti *threading* dan manajemen memori tingkat rendah [BUR-10:32].

1. *Core Libraries* : Android aplikasi dibangun dalam bahasa java, namun Dalvik sebagai virtual mesinya bukan *Java Virtual Machine*, sehingga diperlukan sebuah *libraries* yang berfungsi untuk menterjemahkan bahasa java atau c yang dikendalikan oleh *core libraries* [MEI-10:13].
2. *Davlik Virtual Machine (DVM) :* Virtual mesin yang berbasis *register* yang dioptimalkan untuk menjalankan fungsi-fungsi secara efisien, dimana merupakan pengembangan yang mampu membuat linux *kernel* untuk *thereading* dan manajemen memori tingkat rendah. Semua *hardware* android dan akses pelayanan sistem

dikelola menggunakan Dalvik sebagai *middle tier*. DVM sebagai *host* eksekusi aplikasi memungkinkan pengembang memiliki lapisan abstraksi yang memastikan mereka tidak perlu khawatir tentang implementasi pada *hardware* [MEI-10:15]. DVM mengeksekusi *file* Dalvik *executable*, sebuah format yang dioptimalkan untuk memastikan penggunaan memori minimal. Format Davlik *executable* (.dex) dibuat dengan mengubah klas yang di-*compile* dalam bahasa Java menggunakan *tools* yang disediakan dalam Android SDK [MEI-10:15].

e. *Linux Kernel*

Android bergantung pada Linux versi 2.6 untuk layanan sistem inti seperti keamanan, manajemen memori, manajemen proses, *network stack*, dan *driver model*. *Kernel* juga bertindak sebagai lapisan abstraksi antara *hardware* dan *software stack* [MEI-10:13].

2.3 *Web Service*

Menurut W3C [W3C-11] *web service* adalah suatu sistem perangkat lunak yang didisain untuk mendukung interaksi mesin ke mesin pada suatu jaringan. *Web service* mempunyai suatu interface yang diuraikan dalam suatu format *machine-processible* seperti WSDL. Sistem lain yang berinteraksi dengan *web service* dilakukan melalui *interface* atau antar muka menggunakan pesan seperti pada SOAP. Pada umumnya pesan ini melalui HTTP dan XML yang merupakan salah satu standard *web*. Perangkat lunak aplikasi yang ditulis dalam berbagai bahasa pemrograman dan berjalan pada berbagai *platform* dapat menggunakan *web service* untuk pertukaran data pada jaringan komputer seperti *Internet*. Sebagai contoh adalah antara Java dan Python atau Microsoft Windows dan aplikasi Linux.

Menurut Michael C. Daconta [DAC-05], w*eb service* adalah aplikasi perangkat lunak yang dapat ditemukan, diuraikan, dan diakses berdasarkan pada XML atau biasa disebut *Application Programming Interface* (API) dan protokol *standard web* pada *intranet*, *extranet*, dan *Internet*. Pada skripsi ini, metode pertukaran data yang dipakai adalah JSON (*JavaScript Object Notation*).

2.3.1 JSON (*JavaScript Object Notation*)

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca, mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari bahasa pemprograman JavaScript. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun. JSON ditulis menggunakan gaya bahasa yang umum digunakan oleh *programer* keluarga C termasuk C, C++, C#, Java, JavaScript, Perl dan Python. JSON sangat ideal sebagai bahasa pertukaran-data [JSO-11].

JSON terbuat dari dua struktur [JSO-11] :

1. Kumpulan pasangan nama atau nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Penggunaan JSON secara umum terdiri dari fungsi *encode* dan *decode*. Fungsi *encode* untuk mengubah data menjadi JSON *array* dan fungsi *decode* untuk mengubah JSON *array* menjadi suatu nilai kembalian. Contoh penggunaan sintaks fungsi *encode* pada metode JSON dapat dilihat pada Gambar 2.3.

```
<?php
$arr =array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
echo json_encode($arr);
?>
```

Gambar 2.3 Sintaks JSON *encode*

Sumber : [PHP-11]

Method `json_encode($arr)` akan mengubah nilai dari variabel `$arr` menjadi suatu format yang dapat dibaca sebagai JSON *array*. Hasil keluaran dari sintaks tersebut dapat dilihat pada Gambar 2.4.

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

Gambar 2.4 Hasil JSON *encode*

Sumber : [PHP-11]

JSON *decode* adalah proses memperoleh nilai dari suatu variable dengan format JSON *array*. Sintaks JSON pada proses *decode* dapat dilihat pada Gambar 2.5.

```
<?php
$json = '{"foo-bar": 12345}';
$obj = json_decode($json);
print $obj->{'foo-bar'};
?>
```

Gambar 2.5 Sintaks JSON *decode*

Sumber : [PHP-11]

Pada sintaks di atas, *method* `json_decode($json)` akan mengekstrak nilai dari variabel JSON *array* `$json` menjadi suatu nilai *output*. Hasil keluaran dari proses *decode* tersebut dapat dilihat pada Gambar 2.6.

```
12345
```

Gambar 2.6 Hasil JSON *decode*

Sumber : [PHP-11]

2.4 *Global Positioning System* (GPS)

GPS adalah satu-satunya sistem satelit navigasi global untuk penentuan lokasi, kecepatan, arah, dan waktu yang beroperasi secara penuh di dunia saat ini [CTI-11]. GPS *receiver* harus berada dalam *line-of-sight* (LoS) terhadap beberapa satelit untuk menentukan posisi, sehingga GPS hanya ideal untuk digunakan dalam *outdoor positioning*. GPS dapat menyediakan tingkat akurasi yang sangat tinggi yaitu hingga mencapai ketepatan ±15 meter [MCN-04:55]. Informasi yang dapat diperoleh dari sebuah GPS *receiver* antara lain adalah waktu, koordinat lokasi (*latitude*, *longitude*, dan *altitude*), kecepatan, dan arah [MCN-04:58].

Format keluaran data dari penerima GPS yang paling lazim adalah format NMEA (*National Marine Electronics Association*) 0183. Dengan menggunakan keluaran data ini, penerima GPS dapat dihubungkan dengan perangkat lain melalui *port* serial. Data keluaran dalam format NMEA 0183 berupa kalimat (*string*) yang merupakan karakter ASCII 8-bit dan ditransmisikan dengan kecepatan 4.800 *bps* [ELR-06:112]. Setiap awal kalimat diawali dengan karakter "$", dua karakter *Talker* ID, tiga karakter *Type data* ID, dan diikuti oleh *data fields* yang masing-masing dipisahkan oleh koma (,) serta diakhiri oleh *optional checksum* dan karakter *cariage return* atau *line feed* (CR/LF) [ELR-06:113].

Sebuah GPS mempunyai *Talker* ID berupa GP. Berdasarkan informasi yang ada di dalamnya, tipe data yang dikeluarkan oleh GPS dengan format NMEA dapat dikelompokkan menjadi beberapa jenis antara lain GGA (*Global Positioning System Fix Data*), GLL (*Geographic position, Latitude and Longitude*), WPL (*Waypoint Location*) dan RMC (*Recommended Minimum sentenceC*) [PET-12]. Tipe data yang mencakup informasi posisi yang dibutuhkan dalam skripsi ini adalah RMC (*Recommended Minimum Specific GPS/TRANSIT Data*). Contoh data yang diterima oleh sebuah GPS *receiver* dengan format NMEA 0183 bertipe data RMC ditunjukkan pada Gambar 2.7.

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A
```

Gambar 2.7 Contoh format data standar NMEA dari GPS

Sumber : [PET-12]

Kode tersebut berisi sederetan data yang diterima oleh sebuah GPS *receiver*. Data yang ada di dalamnya antara lain adalah koordinat posisi, kecepatan, dan waktu data tersebut diambil. Keterangan mengenai *data fields* yang terdapat pada kode RMC dijelaskan pada Tabel 2.1.

Tabel 2.1 Penjelasan dari format kode RMC

Kode	Keterangan
$	Awal (*delimiter*) dari kode.
GP	*Talker identifier* dari GPS.
RMC	*Identifier* dari tipe data.
123519	Waktu UTC data diambil (*hhmmss*).
A	Status, A=*active* atau V=*void*.
4807.038	Garis Lintang (*latitude*).
N	N = utara (*North*) atau S = selatan (*South*).
01131.000	Garis Bujur (*longitude*).
E	E = timur (*East*) atau W = barat (*West*).
022.4	Kecepatan dalam *knots*.
084.4	Arah dengan kategory *true*.
230394	Tanggal data diambil (*ddmmyy*).
003.1	*Magnetic Variation*.
W	W = barat (*West*) atau E = timur (*East*).
*6A	Akhir kalimat (*Checksum*<CR><LF>), selalu dimulai dengan *

Sumber : [PET-12]

Data geografis yang digunakan dalam skripsi ini adalah data koordinat lintang (*latitude*) dan koordinat bujur (*longitude*). GPS *receiver* ada beberapa macam antara lain adalah DGPS (*Differential*-GPS), A-GPS (*Assisted*-GPS) dan WAAS (*Wide Area Augmentation System*) [ELR-06:79-157]. GPS *receiver* yang digunakan pada skripsi ini adalah GPS *receiver* dengan A-GPS yang tertanam pada perangkat *smartphone* android.

2.4.1 *Assisted - Global Positioning System* (A-GPS)

Frank Van Diggelen [DIG-09:1] mengatakan bahwa A-GPS merupakan penyempurnaan dari GPS sebagai satelit penentu posisi di belahan bumi. Satelit GPS yang dimiliki bumi mempunyai konstelasi 24 satelit dalam enam orbit yang mendekati lingkaran, setiap orbit ditempati oleh empat buah satelit dengan interval antara yang tidak sama. Orbit satelit GPS berinklinasi 550° terhadap bidang equator dengan ketinggian rata-rata dari permukaan bumi sekitar 20.200 km [ELR-06:1].

Metode *Advanced Positioning* yang terdapat pada A-GPS merupakan metode penentuan posisi yang paling tinggi akurasinya dibandingkan metode deteksi posisi lainnya seperti *Time Difference Of Arrival* (TDOA) maupun *Enhanced Observed Time Difference* (E-OTD). A-GPS jauh lebih efisien dan efektif dalam mengakses informasi dari satelit karena tidak perlu mencari data satu persatu dari ke-24 satelit yang ada, namun A-GPS telah mengetahui sasaran (satelit) mana yang dibutuhkan atau dituju [ELR-06:125].

Pada sistem A-GPS, telepon genggam akan menangkap sinyal satelit yang lalu dikirimkan ke *server* penyedia layanan telepon, hasil perhitungan lokasi yang dilakukan oleh *server* dikirimkan kembali ke telepon genggam. Peta juga dapat dikirimkan oleh *server* tersebut atau sudah disimpan pada telepon genggam. Sistem ini hanya berfungsi bila jaringan telepon genggam mampu disediakan oleh pengguna. Proses perhitungan data dilakukan oleh *server* penyedia jaringan telepon, maka telepon genggam tidak memerlukan *prosesor* yang canggih [ELR-06:125-126].

Kelebihan A-GPS antara lain adalah sebagai berikut [KAP-06:543-546] :

1. A-GPS menawarkan solusi terakurat dari metode-metode yang telah ada sebelumnya. Lebih lanjut, A-GPS merupakan layanan yang menggabungkan sistem GPS dan layanan GSM (*Global System for Mobile Communications*).
2. Layanan ini juga berguna untuk dapat menjembatani kekurangan dan kelebihan GPS dan LBS (*Location Based Service*). LBS sebenarnya adalah salah satu layanan tambahan dari selular GSM. LBS bukanlah

sistem tetapi merupakan layanan yang menggunakan sistem tambahan penunjang sistem GSM.

3. A-GPS menjadikan proses akses informasi menggunakan satelit menjadi lebih mudah dan cepat.

Metode A-GPS merupakan metode yang berbasis pada waktu. Pada metode ini, akan dilakukan pengukuran waktu tiba dari sebuah sinyal yang dikirimkan dari satelit GPS. Perangkat yang digunakan harus memiliki fasilitas untuk mengakses GPS. A-GPS seperti halnya GPS, juga menggunakan satelit yang memancarkan sinyal radio ke penerima yang terpasang pada permukaan atas bumi. Penerima GPS dihubungkan dengan antena yang menerima sinyal radio untuk mengkalkulasi posisi penerima GPS [DIG-09:1-2].

2.4.2 *Geotagging*

Geotagging adalah proses menambahkan metadata identifikasi geografis ke berbagai media seperti foto, video, *website*, atau *RSS feed* dan merupakan bentuk dari metadata geospasial. Data di dalamnya terdiri dari koordinat lintang (*latitude*) dan bujur (*longitude*), meskipun mereka juga dapat mencakup ketinggian (*altitude*), bantalan (*bearing*), akurasi data, dan nama tempat [MIL-09].

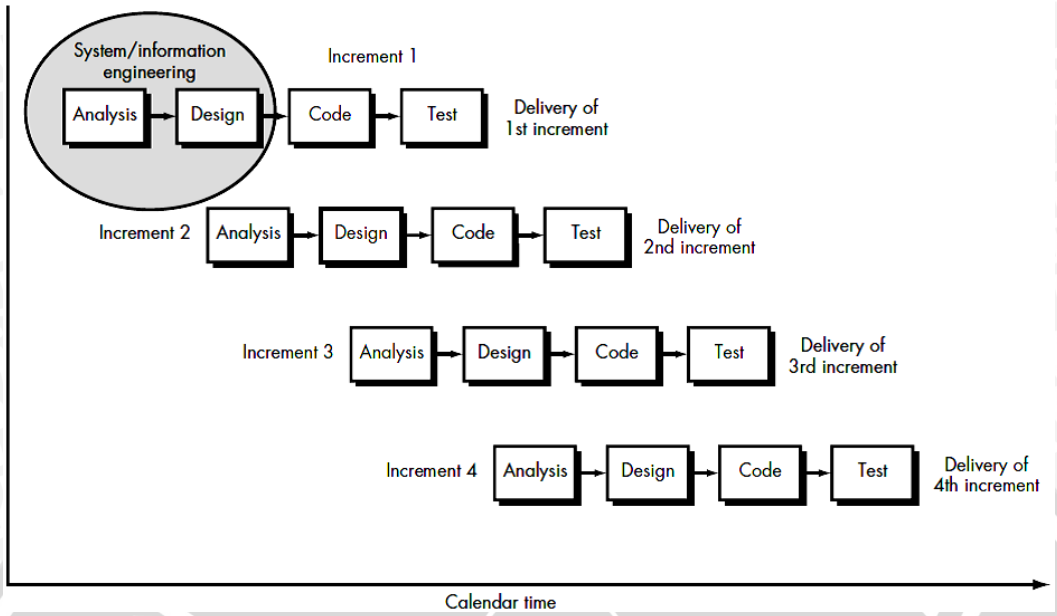
Geotagging dapat membantu pengguna menemukan berbagai macam informasi lokasi secara spesifik. Layanan geotagging juga dapat berpotensi digunakan untuk menemukan lokasi berbasis berita, *websites*, atau sumber lainnya [LAR-11]. Proses ini biasanya disebut *geocoding*. *Geocoding* adalah sebuah istilah yang lebih sering mengacu pada proses mengambil non-koordinat berbasis *identifier* geografis, seperti alamat jalan, dan menemukan koordinat geografis yang terkait. Penggunaan *geocoding* dapat dicontohkan seperti kamera yang memasukkan koordinat lokasi ketika membuat gambar dengan menggunakan *built-in* GPS *receiver* [MIL-09].

2.5 Rekayasa Perangkat Lunak

Menurut IEEE, rekayasa perangkat lunak adalah penerapan yang sistematis, disiplin, serta pendekatan yang terukur terhadap pengembangan, pengoperasian, dan pemeliharaan perangkat lunak [PRE-10:20].

Seorang *software engineer* atau sekumpulan *software engineer* harus menggabungkan strategi pengembangan perangkat lunak yang meliputi proses, metode, dan alat bantu yang digunakan dalam proses pengembangan. Strategi ini disebut sebagai model proses (*process model*) atau paradigma rekayasa perangkat lunak (*software engineering paradigm*). Model proses untuk rekayasa perangkat lunak dipilih sesuai dengan sifat dari proyek dan aplikasi yang akan dibuat. Terdapat beberapa model proses untuk rekayasa perangkat lunak antara lain *linear sequential model* (*waterfall*), *prototyping, Rapid Application Development* (RAD), *incremental model* dan *spiral model* [PRE-10:20-42]. Model proses yang digunakan dalam skripsi ini adalah *incremental model*.

Incremental model merupakan gabungan dari elemen *linear sequensial model* dan filosofi dari *prototyping* sehingga memudahkan seorang pengembang (*developer*) dalam mengidentifikasi kebutuhan pengguna [PRE-10:35]. *Incremental model* merupakan metodologi pengembangan perangkat lunak yang efisien dan efektif untuk *project* dengan skala kecil hingga skala menengah. *Incremental model* menawarkan strategi pengembangan perangkat lunak yang memungkinkan pengguna (*user*) dapat menggunakan versi awal (*increment* pertama) sebagai *prototype* untuk memperoleh informasi tentang persyaratan kebutuhan (*requirement*) mereka untuk pengembangan sistem selanjutnya [SOM-11:47]. Kelebihan dari model proses ini antara lain adalah proses *development* yang lebih cepat, lebih mudah dalam mengetahui kebutuhan pengguna, dan *resource* yang dibutuhkan untuk melakukan perubahan terhadap perangkat lunak yang dikembangkan lebih sedikit [SOM-11:33]. Proses pengembangan menggunakan model proses *incremental* ini dapat dilihat pada Gambar 2.8.



Gambar 2.8 *Incremental model*

Sumber : [PRE-10:35]

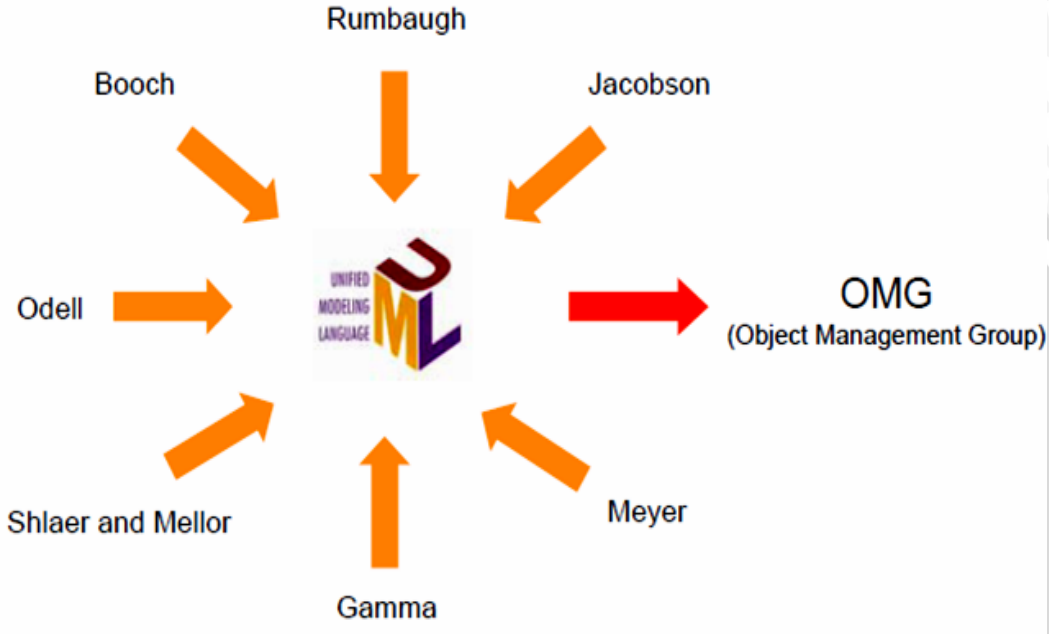
Versi awal dari perangkat lunak (*increment* 1) dievaluasi oleh pengguna sehingga pihak pengembang mengetahui dengan jelas apa yang dibutuhkan oleh pengguna. Pihak pengembang kemudian mengembangkan perangkat lunak untuk *increment* selanjutnya sesuai dengan saran dan permintaan pengguna. Proses ini berulang sampai perangkat lunak selesai atau mencapai tahap yang diinginkan oleh pengguna [PRE-10:35].

Pada skripsi ini digunakan metode analisis kebutuhan, perancangan, implementasi, dan pengujian berorientasi objek menggunakan bahasa pemodelan UML (*Unified Modelling Language*).

2.6 UML (*Unified Modelling Language*)

UML adalah sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem perangkat lunak yang dapat berjalan pada sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun [DHA-06:2].

UML dirancang oleh Grady Booch, Ivar Jacobson, dan James Rumbaugh untuk menyatukan bermacam-macam bahasa pemodelan dan metode berorientasi objek dengan cara menggabungkan bahasa pemodelan dan metode berorientasi objek terbaik yang telah ada [DHA-06:2]. Penggabungan metode pemodelan ini dtunjukkan oleh Gambar 2.9.



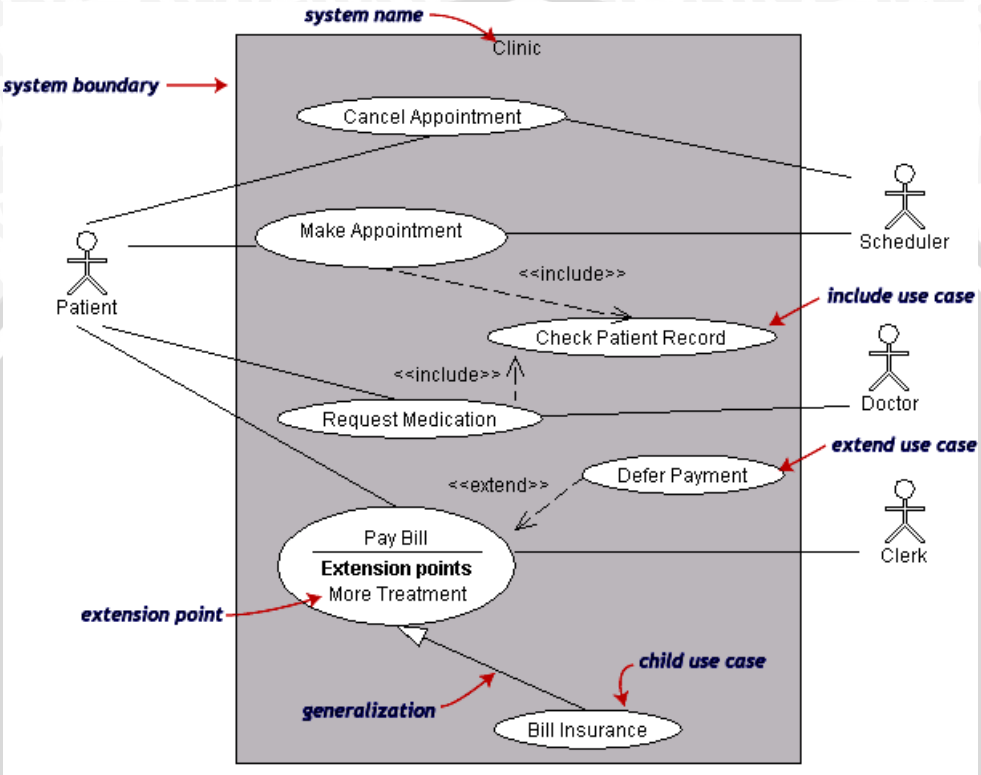
Gambar 2.9 Penyatuan metode *modelling* UML

Sumber : [DHA-06:3]

2.6.1 *Use Case Diagram*

Use case diagram merupakan fitur penting dari notasi UML. Sebuah *Use case* harus merepresentasikan semua kemungkinan interaksi antara aktor dengan sistem serta semua kebutuhan sistem [SOM-11:107]. *Use case* juga memodelkan sistem dari perspektif *end-user*. Selama penggalian kebutuhan sistem, *use case* harus mampu mencapai beberapa objektif. Objektif-objektif tersebut antara lain adalah *use case* mampu untuk mendefinisikan kebutuhan yang bersifat fungsional dan operasional pada sistem dengan cara mendefinisikan skenario yang disetujui oleh *end-user* dan *software engineer team.* Penjelasan tentang cara *end-user* berinteraksi dengan sistem pada *use case* harus jelas dan tidak ambigu. *Use case* juga harus menyediakan sebuah dasar untuk *validation testing* [PRE-10:581].

Use case diagram sangat diperlukan dalam penyusunan *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem [DHA-06:4]. Contoh *use case diagram* ditunjukkan pada Gambar 2.10.

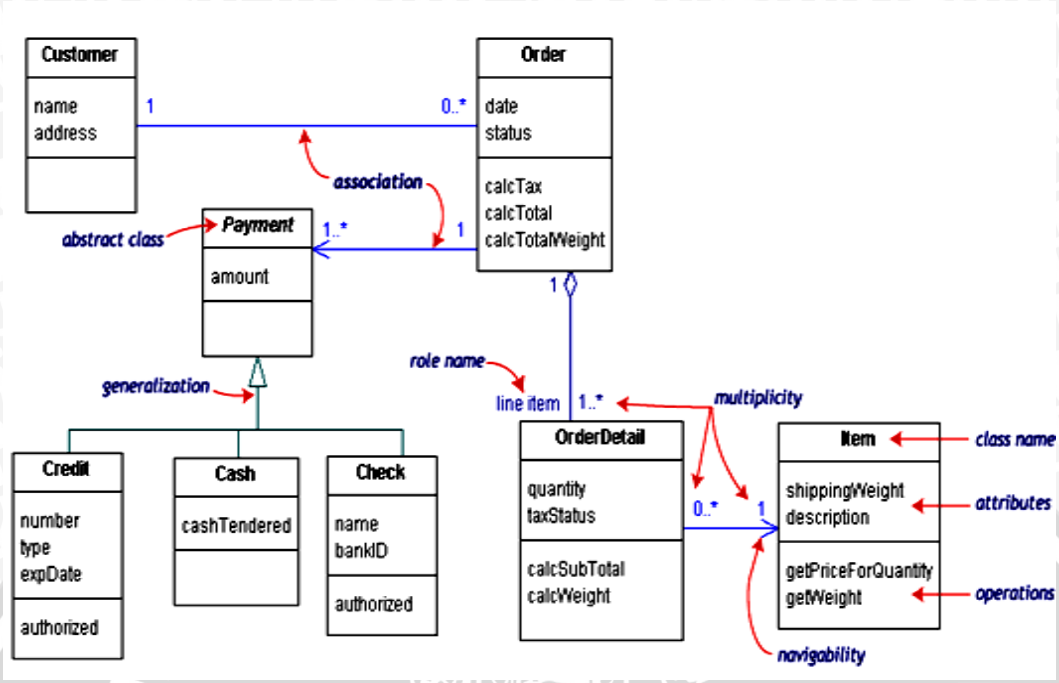


Gambar 2.10 Contoh *use-case diagram*

Sumber : [DHA-06:5]

2.6.2 *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut atau properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda atau fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. Sebuah *class* terdiri dari tiga area pokok yaitu nama *class* , atribut, dan metoda atau operasi [DHA-06:5]. Contoh dari *class diagram* dapat dilihat pada Gambar 2.11.



Gambar 2.11 Contoh *class diagram*

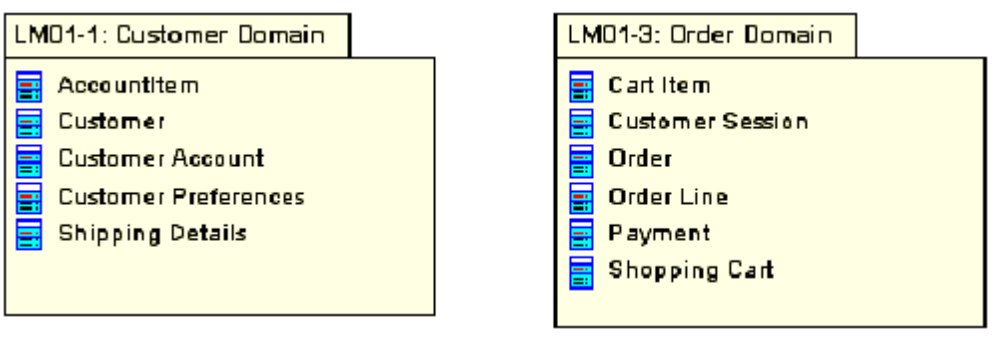
Sumber : [DHA-06:6]

Dalam penggunaan metode *object-oriented* dapat dipastikan *class-class* memiliki hubungan-hubungan tertentu sesuai dengan fungsinya sendiri-sendiri. Hubungan antar *class* ada beberapa, yaitu [DHA-06:6] :

1. Asosiasi (*Asscociation*), adalah hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi (*Aggregation*) adalah hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, adalah hubungan hirarkis antar *class* dimana sebuah *class* dapat diturunkan dari *class* lain dan mewarisi semua atribut serta metoda *class* asalnya. *Class* tersebut dapat menambahkan fungsionalitas baru, sehingga dapat disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi (*generalization*).

4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Sebuah *package* dapat berisi beberapa *class*. Contoh penggunaan diagram *class* yang terdiri atas banyak *package* ditunjukkan oleh Gambar 2.12.

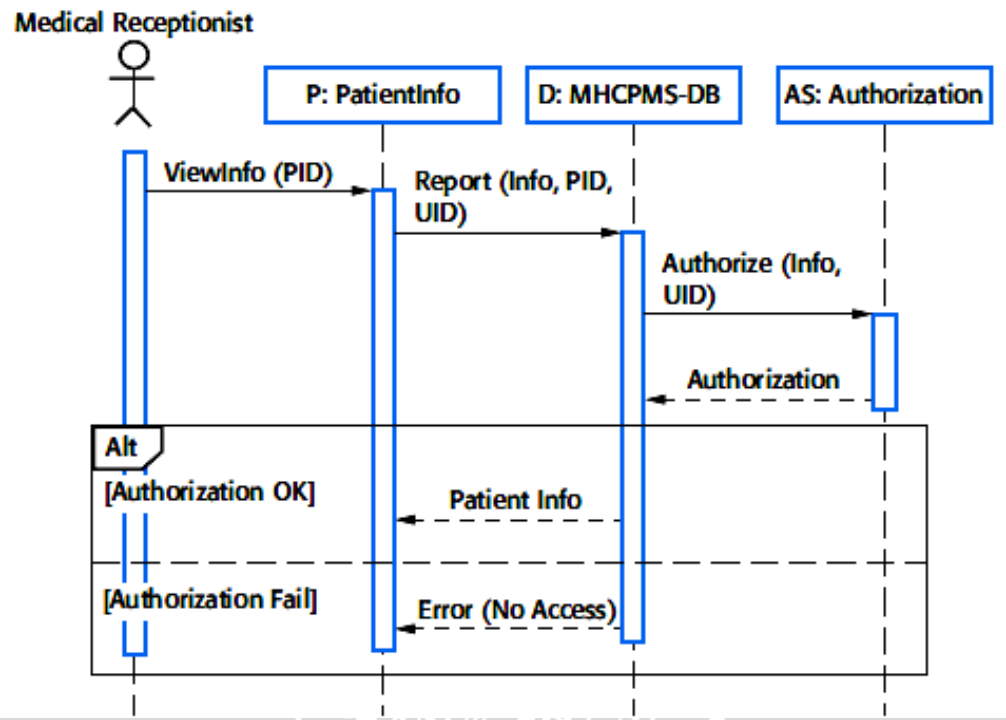


Gambar 2.12 Contoh *Package*

Sumber : [DHA-06:6]

2.6.3 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antara aktor dengan objek serta interaksi antar objek di dalam sistem (termasuk pengguna, display, dan sebagainya), berupa *message* yang digambarkan terhadap waktu [SOM-11:126]. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan [DHA-06:8]. Hal ini ditunjukkan pada Gambar 2.13.



Gambar 2.13 Contoh *sequence diagram*

Sumber : [SOM-11:127]

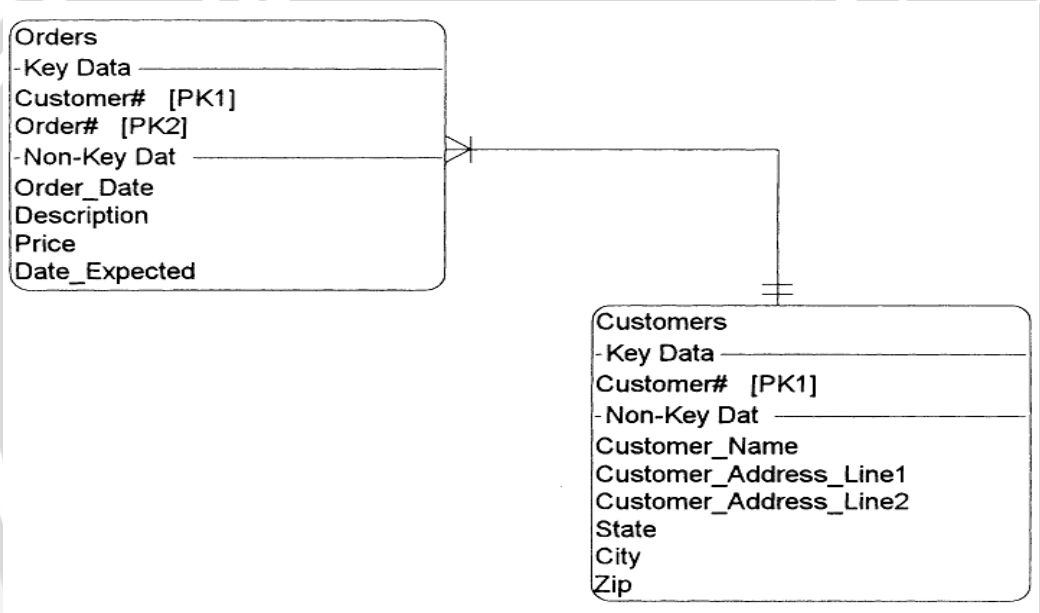
Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi atau metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity* [DHA-06:8].

2.7 Basis Data (*Database*)

Database merupakan tempat penyimpanan informasi yang disimpan secara terstruktur dengan cara yang sederhana, yaitu [DUB-09]:

1. Kumpulan dari data pada *database* diorganisir kedalam tabel-tabel.
2. Setiap tabel diorganisir kedalam baris dan kolom.
3. Setiap baris pada suatu tabel disebut dengan *record*.
4. *Record* dapat terdiri dari beberapa bagian dari informasi.

Data yang tersimpan didalam *database* memiliki hubungan satu sama lain. Hubungan antara data yang tersimpan dalam *database* digambarkan dalam *entity relationship diagrams* (ERD) [LAN-08:81]. Suatu *entity* pada perancangan *database* adalah objek dari data yang disimpan. Suatu *entity* dapat memiliki beberapa atribut. ERD menggambarkan *entity-entity* yang berinteraksi satu sama lain atau disebut dengan *relationship.* Rasio cardinalitas dalam sebuah *relationship* adalah 1:1, 1:N, dan N:N [LAN-08:82]. Contoh penggunaan dari ER-*diagram* yang merupakan *relationship* antara *entity Customers* dengan *entity Orders* ditunjukkan pada Gambar 2.14.



Gambar 2.14 Contoh ERD

Sumber: [LAN-08:82]

2.8 Pengujian Perangkat Lunak

Pengujian adalah kegiatan yang dilakukan untuk mengevaluasi kualitas produk dan mengidentifikasi cacat atau masalah. Arsitektur dari perangkat lunak berorientasi objek menghasilkan sekumpulan *layered subsystems* yang mengenkapsulasi *class-class* yang berkolaborasi. Setiap elemen sistem (*subsistem* dan *class*) melakukan fungsi yang membantu untuk mencapai kebutuhan sistem.

Hal ini sangat penting untuk menguji sebuah sistem berorientasi objek pada berbagai macam *level* yang berbeda dalam sebuah usaha untuk menemukan kesalahan-kesalahan yang mungkin terjadi dari kolaborasi *class-class* dan komunikasi subsistem melewati *architetural layer* [PRE-10:631].

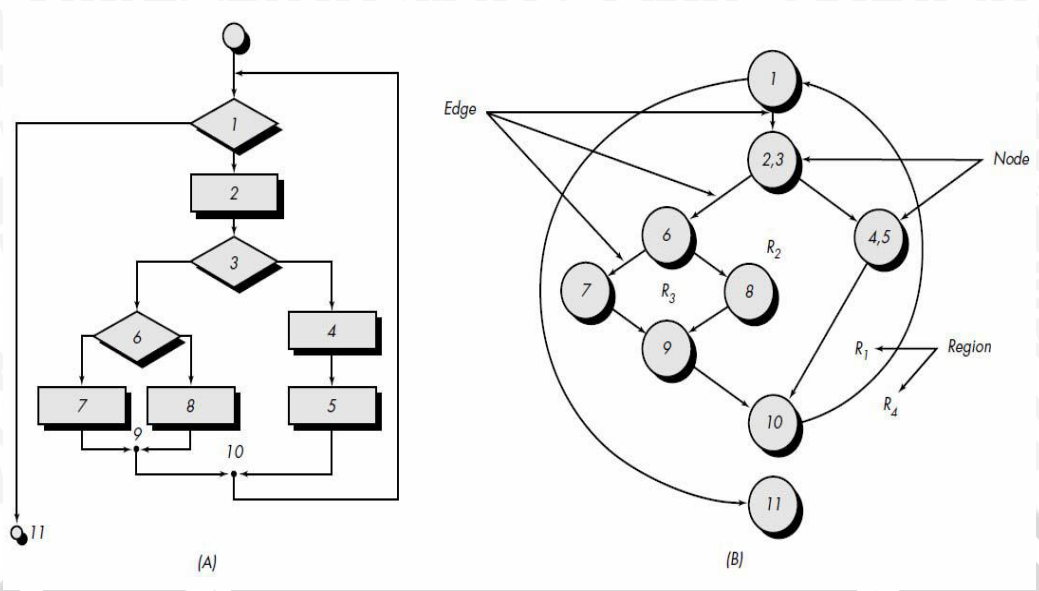
2.8.1 Teknik Pengujian

Pengujian perangkat lunak memerlukan perancangan kasus uji (*test case*) agar dapat menemukan kesalahan dalam waktu singkat dan usaha minimum. Berbagai macam metode perancangan kasus uji telah berevolusi. Metode-metode ini menyediakan pihak *developer* pendekatan yang sistematis untuk pengujian. Terlebih lagi metode-metode ini menyediakan mekanisme yang dapat membantu memastikan kelengkapan dari pengujian dan menyediakan kemungkinan tertinggi untuk menemukan kesalahan-kesalahan dalam perangkat lunak [PRE-10:443]. Teknik atau metode perancangan kasus uji yang digunakan adalah *white-box testing* dan *black-box testing*.

2.8.1.1 *White-Box Testing*

White-box testing atau *glass-box testing* merupakan sebuah metode perancangan kasus uji yang menggunakan struktur kontrol dari perancangan prosedural untuk memperoleh kasus uji [PRE-10:444]. Ada dua jenis pengujian yang termasuk *white-box testing* yaitu *basis path testing* dan *control structure testing*.

White-box dalam skripsi ini dilakukan dengan menggunakan *basis path testing* yang diusulkan pertama kali oleh Tom McCabe [PRE-10:445]. Sebelum metode *basis path* dapat digunakan, notasi sederhana untuk representasi aliran kontrol yang disebut diagram alir (*flow graph*) harus didefinisikan. Setiap representasi desain prosedural yang berupa *flow chart* dapat diterjemahkan ke dalam *flow graph*. Setelah *flow graph* didefinisikan maka harus ditentukan ukuran kompleksitasnya (*cyclomatic complexity*). Gambar 2.14 menunjukkan proses transformasi *flow chart* ke *flow graph*.



Gambar 2.15 Transformasi *flow chart* (A) ke *flow graph* (B)

Sumber : [PRE-10:447]

Cyclomatic complexity adalah *metriks* perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila metriks ini digunakan dalam konteks metode pengujian *basis path*, maka nilai yang terhitung untuk *cyclomatic complexity* menentukan jumlah jalur independen (*independent path*) dalam *basis set* suatu program dan memberi batas atas bagi jumlah pengujian yang diharus dilakukan untuk memastikan bahwa semua *statement* telah dieksekusi sedikitnya satu kali. Untuk menentukan *cyclomatic complexity* bisa dilakukan dengan beberapa cara, diantaranya [PRE-10:448]:

1. Jumlah *region* pada *flow graph* sesuai dengan *cyclomatic complexity*.
2. *Cyclomatic complexity* V(G), untuk grafik G adalah V(G) = E – N + 2, dimana E adalah jumlah *edge*, dan N adalah jumlah *node.*
3. V(G) = P + 1, dimana P adalah jumlah *predicate node* yaitu *node* yang merupakan kondisi (ada dua atau lebih *edge* akan keluar *node* ini).

2.8.1.2 *Black-Box Testing*

Black-box testing atau *behavioral testing* dilakukan pada fungsionalitas perangkat lunak [PRE-10:459]. Pengujian *black-box* dilakukan untuk menemukan kesalahan dalam kategori berikut [PRE-10:460]:

1. Fungsi-fungsi yang tidak benar atau hilang.
2. Kesalahan *interface.*
3. Kesalahan dalam struktur data atau akses *database eksternal.*
4. Kesalahan kinerja.
5. Inisialisasi dan kesalahan terminasi.

2.8.2 Strategi Pengujian

Strategi untuk pengujian perangkat lunak merupakan aktifitas mengintegrasikan metode perancangan kasus uji (*test case*) perangkat lunak ke dalam sederetan langkah pengujian yang terencana sehingga menghasilkan konstruksi perangkat lunak yang baik [PRE-10:477]. Sejumlah strategi pengujian perangkat lunak telah diusulkan di dalam literatur. Strategi pengujian harus mengakomodasi pengujian tingkat rendah yang diperlukan untuk membuktikan bahwa segmen kode sumber yang kecil telah diimplementasikan dengan tepat, demikian juga pengujian tingkat tinggi yang memvalidasi fungsi-fungsi sistem mayor yang berlawanan dengan kebutuhan pelanggan. Proses pengujian dimulai dengan pengujian yang berfokus pada setiap modul secara individual (*unit testing*), dilanjutkan dengan pengujian integrasi (*integration testing*) dan berakhir pada pengujian validasi (*validation testing*) [PRE-10:481].

2.8.2.1 Pengujian Unit (*Unit Testing*)

Pengujian unit berfokus pada usaha verifikasi pada inti terkecil dari desain perangkat lunak, yakni modul. Dengan menggunakan gambaran desain prosedural sebagai panduan, jalur kontrol yang penting diuji untuk mengungkap kesalahan di dalam batas modul tersebut. Kompleksitas relatif dari pengujian dan kesalahan yang diungkap dibatasi oleh ruang lingkup batasan yang dibangun untuk pengujian unit. Pengujian unit biasanya berorientasi pada *white-box*, dan langkahnya dapat dilakukan secara paralel untuk model bertingkat [PRE-10:485].

2.8.2.2 Pengujian Integrasi (*Integration Testing)*

Integration testing merupakan pengujian yang dilakukan untuk menguji beberapa komponen yang saling berinteraksi. *Integration testing* dilakukan dengan mengidentifikasi komponen yang ada kemudian mengintegrasikannya sehingga komponen-komponen tersebut saling berinteraksi dan bekerja sama. Ketika sebuah kesalahan ditemukan, maka dapat diartikan bahwa fungsionalitas sistem akan tertanggu jika ada komponen yang bermasalah saling berinteraksi. Pengujian integrasi sebagian besar berkaitan dengan menemukan *bug* dalam sistem [SOM-11:219-221].

Integration testing berorientasi *black box* dan mempunyai dua pola pengujian yaitu *top-down integration* dan *bottom-up integration*. *Top down integration* mempunyai dua jenis pendekatan integrasi yaitu *depth-first integration* (pengujian dimulai dari jalur utama) dan *breadth-first integration* (pengujian dilakukan secara urut per *level*) [PRE-10:488].

Integrasi *top-down* adalah pendekatan inkremental terhadap struktur program. Modul diintegrasikan dengan menggerakkan ke bawah melalui hirarki kontrol, dimulai dengan modul kontrol utama (program utama). *Subordinat* program terhadap modul kontrol utama digabungkan ke dalam struktur dengan cara *depth-first* atau *breadth-first* [PRE-10:489]. Langkah-langkah proses integrasi pada *top-down integration*, yaitu [PRE-10:489]:

1. Modul kontrol utama digunakan sebagai *test driver* dan *stub* digunakan untuk menggantikan semua komponen dibawahnya.
2. Pemilihan pendekatan integrasi yang diinginkan (*depth* atau *breadth first*).
3. Pengujian dikerjakan untuk setiap komponen yang diintegrasikan.
4. *Stub* digantikan dengan komponen yang sebenarnya setelah menyelesaikan serangkaian pengujian.
5. Proses akan terus dilakukan sampai membentuk sebuah perangkat lunak yang utuh.

Bottom-up integration akan memulai integrasi dari komponen pada *level* terendah di struktur program perangkat lunak. Langkah-langkah proses integrasi pada *bottom-up integration*, yaitu [PRE-10:490]:

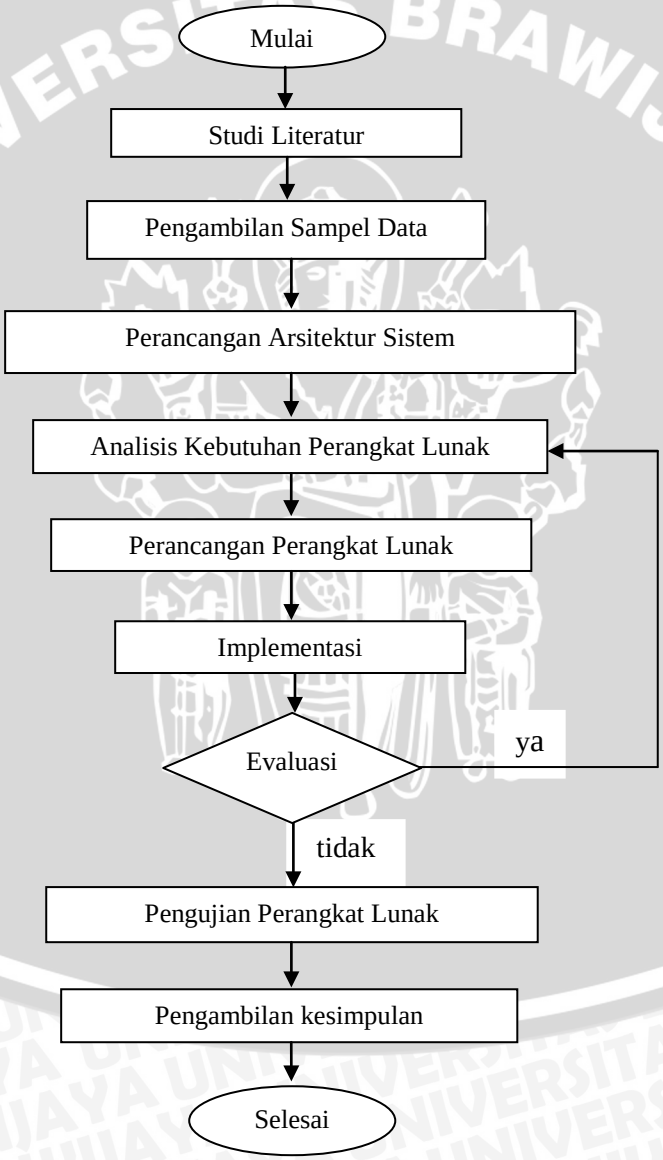
1. Komponen pada *level* terendah digabung ke dalam sebuah sub fungsi (*cluster*).
2. *Driver* akan dibuat untuk menguji setiap *cluster*.
3. *Driver* akan diganti dengan modul sesungguhnya setelah sub fungsi teruji.
4. Proses akan terus dilakukan sampai membentuk sebuah perangkat lunak yang utuh.

2.8.2.3 Pengujian Validasi (*Validation Testing*)

Validation testing merupakan pengujian yang dilakukan pada perangkat lunak secara keseluruhan untuk memastikan bahwa perangkat lunak telah memenuhi seluruh kebutuhan (*requirement*) yang telah ditentukan. Pengujian validasi dapat ditentukan dengan berbagai cara, tetapi definisi yang sederhana adalah bahwa validasi berhasil bila perangkat lunak berfungsi dengan cara yang dapat diharapkan secara bertanggung jawab oleh pengguna. Pengujian ini dilakukan untuk memastikan apakah semua persyaratan fungsional dan persyaratan lainnya sudah dipenuhi (transportabilitas, kompatibilitas, pembetulan kesalahan dan maintanabilitas). Validasi perangkat lunak dicapai melalui sederetan pengujian *black-box* [PRE-10:495].

BAB III
METODE PENELITIAN

Pada bab ini dijelaskan langkah-langkah yang akan dilakukan dalam perancangan, implementasi dan pengujian dari aplikasi perangkat lunak yang akan dibuat. Kesimpulan dan saran disertakan sebagai catatan atas aplikasi dan kemungkinan arah pengembangan aplikasi selanjutnya. Berikut diagram alir runtutan pengerjaan tugas akhir ini :



Gambar 3.1 *Flowchart* prosedur pengerjaan tugas akhir

Sumber : [Metode Penelitian]

3.1 Studi Literatur

Metode ini digunakan untuk mendapatkan dasar teori sebagai sumber acuan untuk penulisan skripsi dan pengembangan aplikasi. Teori dan pustaka yang berkaitan dengan tugas akhir ini meliputi :

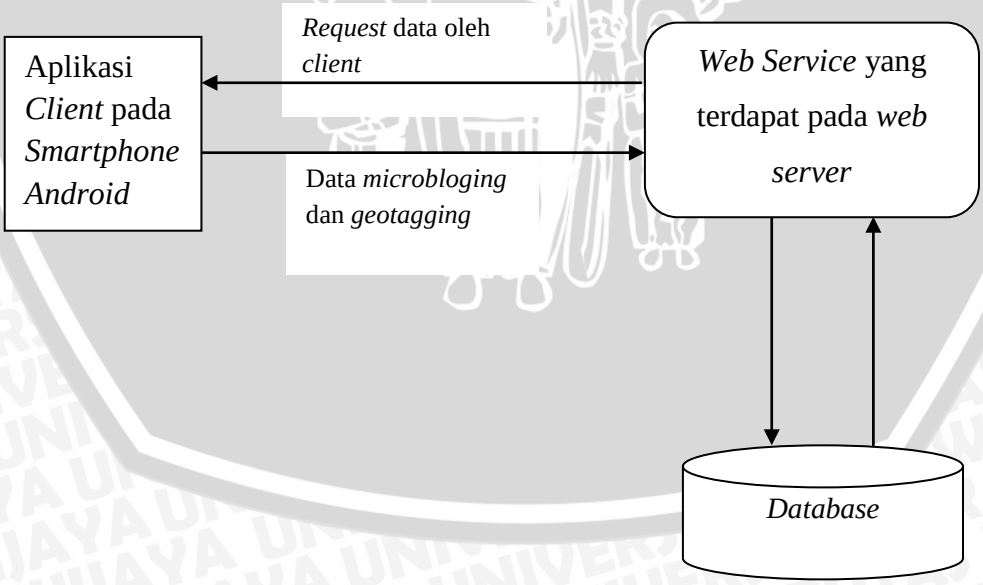
1. Jejaring Sosial (*Social Network*)
2. Android
   a. Fitur-fitur android
   b. Arsitektur android
3. *Web Service*
   a. JSON (*JavaScript Object Notation*)
4. *Global Positioning System* (GPS)
   a. *Assisted-Global Positioning System*(A-GPS)
   b. *Geotagging*
5. Rekayasa Perangkat Lunak
   a. *Incremental Model*
6. UML (*Unified Modeling Language*)
   a. *Use case Diagram*
   b. *Class Diagram*
   c. *Sequence Diagram*
7. Basis Data (*Database*)
8. Pengujian Perangkat Lunak
   a. Teknik Pengujian
      i. *White-Box Testing*
      ii. *Black-Box Testing*
   b. Strategi Pengujian
      i. Pengujian Unit
      ii. Pengujian Integrasi
      iii. Pengujian Validasi

3.2 Pengambilan Sampel Data

Metode ini digunakan untuk mendapatkan data sampel sebagai acuan untuk pengembangan perangkat lunak. Data sampel yang dimaksud adalah data nomor induk serta nama dari mahasiswa dan dosen yang terdaftar di Teknik Informatika Universitas Brawijaya Malang, data selengkapnya dapat dilihat pada Lampiran 1 dan Lampiran 2. Data ini nantinya akan dijadikan sebagai acuan untuk prasyarat pengguna aplikasi jejaring sosial berbasis GPS pada *smartphone* android.

3.3 Perancangan Arsitektur Sistem

Perancangan arsitektur sistem adalah tahap dimana penulis mulai merancang suatu sistem yang mampu memenuhi semua kebutuhan fungsional aplikasi dalam tugas akhir ini. Teori-teori dari pustaka digabungkan dan ilmu yang didapat diimplementasikan untuk merancang serta mengembangkan suatu aplikasi jejaring sosial berbasis GPS pada *smartphone* android. Arsitektur sistem yang akan dirancangan ditunjukkan dalam Gambar 3.2.



Gambar 3.2 Perancangan umum sistem

Sumber : [Metode Penelitian]

Pada *smartphone* android, telah di-*install* aplikasi *client* untuk menampilkan data-data yang terdapat pada *database*. Sebagai penjembatan antara aplikasi *client* dan *database*, penulis menggunakan *web service* yang terdapat pada *web server*. *Client* mengirimkan data ke *web service* menggunakan *Http Post*, sedangkan untuk menangkap *request* dan *response* data dari *web service* ke *client* digunakan komunikasi data JSON. *Response* yang diberikan oleh *web service* di-*decode* oleh aplikasi *client* menjadi suatu objek yang nantinya akan diproses lebih lanjut di dalam aplikasi *client* android.

3.4 Analisis Kebutuhan Perangkat Lunak

Kegiatan analisis kebutuhan perangkat lunak meliputi analisis spesifikasi perangkat lunak. Metode analisis menggunakan bahasa pemodelan UML *(Unified Modeling Language). Use Case Diagram* digunakan untuk mendeskripsikan kebutuhan-kebutuhan dan fungsionalitas sistem dari perspektif *user*. Analisis kebutuhan dilakukan dengan mengidentifikasi semua kebutuhan (*requirements*) sistem yang kemudian akan dimodelkan dalam diagram *use case.* Kebutuhan fungsional yang nantinya akan disediakan oleh aplikasi ini antara lain adalah :

1. Aplikasi ini harus menyediakan fasilitas untuk *login* sehingga hanya *user* yang terdaftar yang dapat menggunakan layanan sistem.
2. Aplikasi ini mampu menampilkan hasil *posting* yang dilakukan oleh *user* dalam bentuk *timeline*.
3. Aplikasi ini menyediakan *interface* untuk *user* dapat melakukan *posting*.
4. Aplikasi ini harus menyediakan *interface* untuk *user* dapat mencari dan menambah teman.
5. Aplikasi ini harus menyediakan *interface* untuk *user* dapat mengedit *account* miliknya.
6. Menyediakan layanan untuk *user* dapat mengetahui posisi dimana *user* lain berada dalam bentuk *map* (peta).

3.5 Perancangan Perangkat Lunak

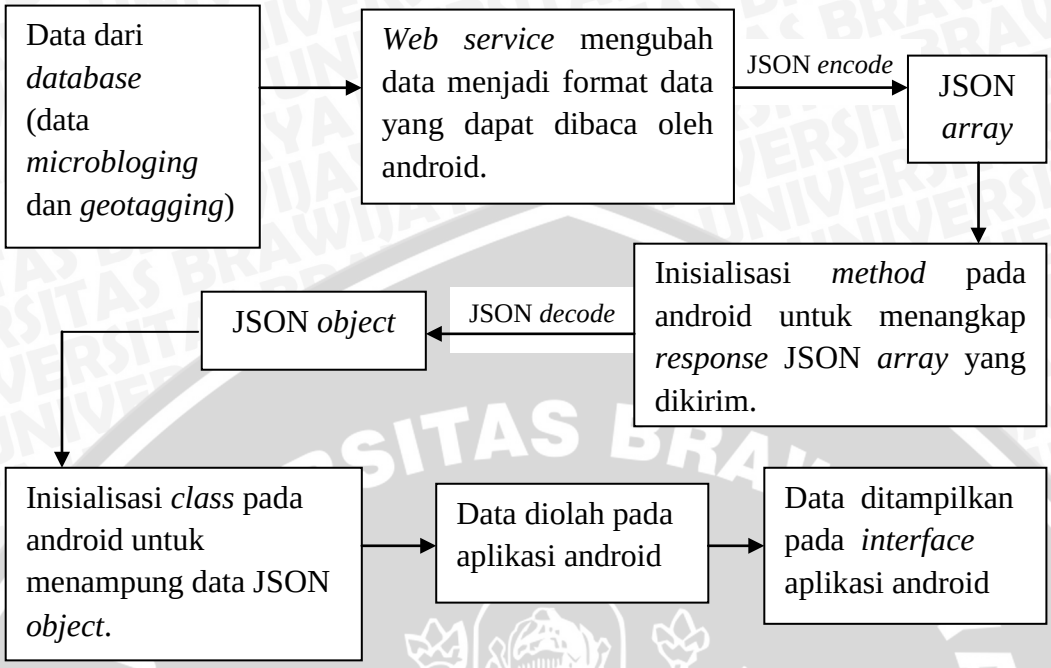
Perancangan perangkat lunak dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan. Perancangan aplikasi jejaring sosial berbasis GPS pada *Smartphone* android ini menggunakan pemodelan UML (*Unified Modeling Language*). Perancangan subsistem dilakukan dengan mengidentifikasi *class-class* dan *interface-interface* yang dibutuhkan dan memodelkannya dalam *class diagram*. Hubungan interaksi antar elemen (objek) yang telah diidentifikasi, dimodelkan dalam *sequence diagram* yang menggambarkan interaksi antar objek yang disusun dalam urutan waktu.

Model proses yang digunakan untuk pengembangan aplikasi ini adalah *incremental model. Incremental model* memungkinkan pihak pengembang melakukan perubahan terhadap kebutuhan sistem, sehingga perlu adanya dokumentasi dari setiap proses *increment* yang dilakukan (*change log*). Perancangan aplikasi ini secara garis besar meliputi perancangan *database* pada *server*, perancangan *web service* untuk pengaksesan data dari *database*, dan perancangan *aplikasi client* pada android. Proses pengiriman data dari android ke *database* ditunjukan pada Gambar 3.3.

Gambar 3.3 Proses pengiriman data dari android ke *database*

Sumber : [Metode Penelitian]

Data dari *database* tidak bisa dibaca secara langsung oleh aplikasi android, perlu adanya perantara yang mampu mengubah data dari *database* menjadi format yang dapat diterjemahkan oleh aplikasi android. Untuk proses pengambilan data dari *database* ke android digunakan JSON sebagai komunikasi datanya. Proses ini secara garis besar dapat dilihat pada Gambar 3.4.



Gambar 3.4 Proses pengambilan data dari *database* ke android

Sumber : Metode Penelitian

3.6 Implementasi

Mengacu pada desain sistem, maka implementasi sistem dilakukan dengan pembangunan aplikasi *smartphone* yang dapat dijalankan pada *platform* android. Untuk mengembangkan aplikasi jejaring sosial berbasis GPS pada *Smartphone* android dibutuhkan beberapa komponen aplikasi pendukung seperti Eclipse sebagai IDE dari bahasa pemrograman Java, MySQL sebagai *database server*, *editor tool* untuk PHP dan android SDK. Implementasi dari perangkat lunak ini dapat dikelompokan menjadi dua bagian yaitu *server side* dan *client side*. Pada *server side* implementasi dilakukan dengan menggunakan PHP sebagai bahasa pemrograman untuk *web service* dan MySQL sebagai *database*-nya. Implementasi pada *client side* menggunakan bahasa pemrograman berorientasi objek yaitu menggunakan bahasa pemrograman Java (*Java Standard Edition*) dan android SDK.

3.7 Evaluasi

Pada tahap ini akan dilakukan evaluasi terhadap aplikasi terkait kesesuaian terhadap kebutuhan fungsional maupun non-fungsional yang telah didefinisikan di tahap analisis kebutuhan. Jika hasilnya belum sesuai maka proses akan kembali ke tahap analisis kebutuhan dengan menambahkan *software requirement* yang belum tersedia dan jika sudah maka akan dilanjutkan ke proses berikutnya.

3.8 Pengujian Perangkat Lunak

Pengujian dari perangkat lunak ini berkaitan dengan pengujian sistem (*white box* dan *black box*). Proses pengujian dilakukan melalui tiga tahapan (strategi) yaitu pengujian unit, pengujian integrasi, dan pengujian validasi. Pada pengujian unit dan pengujian integrasi, akan digunakan teknik pengujian *white box* (*White Box Testing*). Pada pengujian validasi akan digunakan teknik pengujian *black box* (*Black Box Testing*). Pengujian performa juga akan dilakukan untuk mengetahui kinerja *server* dalam melayani *request* dari aplikasi jejaring sosial dan mengirimkan *response* ke aplikasi *client*. Pengujian perangkat lunak bertujuan untuk mengetahui kinerja dari perangkat lunak yang telah dirancang dan dikembangkan serta faktor-faktor yang mempengaruhi kemampuan *server* dalam melayani *request* dari *client*.

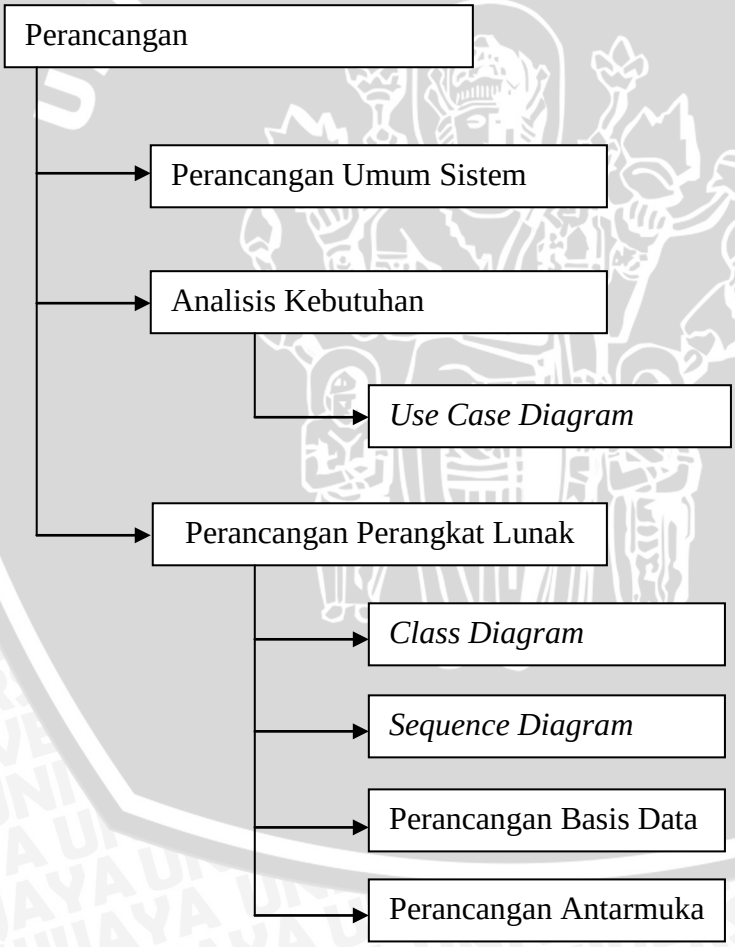
3.9 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian sistem aplikasi telah selesai dilakukan dan didasarkan pada kesesuaian antara teori dan praktik. Kesimpulan diambil untuk menjawab rumusan masalah yang telah ditetapkan sebelumnya. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

BAB IV
ANALISIS KEBUTUHAN DAN PERANCANGAN

Pada bab ini dijelaskan mengenai perancangan perangkat lunak. Perancangan yang dilakukan meliputi tiga tahap. Proses perancangan sistem secara umum dilakukan pada tahap pertama kemudian dilanjutkan dengan analisis kebutuhan pada tahap selanjutnya dan tahap yang terakhir adalah proses perancangan perangkat lunak secara lebih spesifik. Diagram blok yang menggambarkan perancangan perangkat lunak sistem ini ditunjukkan oleh Gambar 4.1.

.

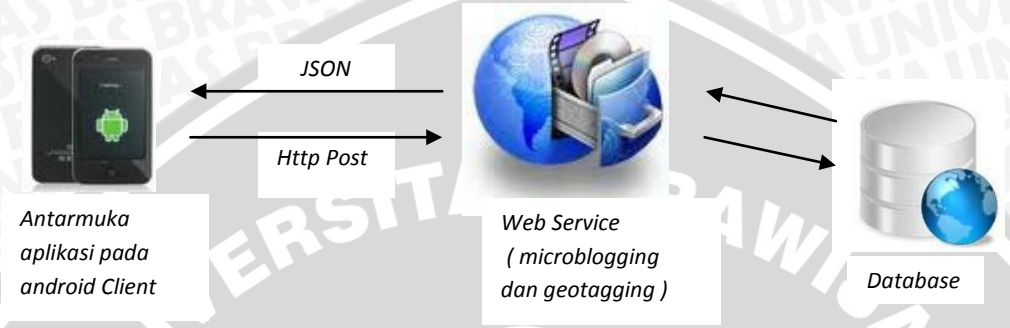


Gambar 4.1 Diagram blok perancangan

Sumber : [Perancangan]

4.1 Perancangan Sistem

Perancangan sistem merupakan tahap awal dari perancangan perangkat lunak. Perancangan sistem dilakukan untuk merepresentasikan arsitektur sistem yang akan dibuat secara umum. Arsitektur sistem yang akan dirancangan ditunjukkan dalam Gambar 4.2.



Gambar 4.2 Perancangan sistem

Sumber : [Perancangan]

Pada *smartphone* android, telah di-*install* aplikasi *client* untuk menampilkan data-data yang terdapat pada *database*. Sebagai penjembatan antara aplikasi *client* dan *database*, penulis menggunakan *web service* yang terdapat pada *web server*. *Web service* tersebut berupa *file* php yang berfungsi untuk mengirimkan data *microbloging* dan *geotagging* dari aplikasi *client* ke *database*, menangkap *request* data dari aplikasi *client* dan memberikan *response* kepada aplikasi *client*.

Client mengirimkan data *microbloging* dan *geotagging* ke *web service* menggunakan *Http Post*, sedangkan untuk *request* dan *response* data dari *web service* digunakan komunikasi data JSON. Data dari *database* diolah di dalam *file* php sehingga membentuk suatu objek, objek tersebut kemudian di-*encode* menjadi format JSON dan dikirimkan ke aplikasi *client*. *Response* yang diberikan oleh *web service* di-*decode* oleh aplikasi *client* menjadi suatu objek yang nantinya akan diproses lebih lanjut di dalam aplikasi *client* android.

4.2 Analisis Kebutuhan (*Requirement Analysis*)

Tahap analisis kebutuhan ini diawali dengan identifikasi aktor yang terlibat dengan sistem, penjabaran kebutuhan perangkat lunak dan kemudian memodelkannya ke dalam suatu *diagram use case.* Analisis kebutuhan ini ditujukan untuk menggambarkan kebutuhan-kebutuhan yang harus disediakan oleh sistem agar dapat memenuhi kebutuhan pengguna.

4.2.1 Identifikasi Aktor

Tahap ini mempunyai tujuan untuk melakukan identifikasi terhadap aktor yang akan berinteraksi dengan sistem. Penjelasan dari masing-masing identifikasi aktor dapat dilihat pada Tabel 4.1.

Tabel 4.1 Deskripsi Aktor

Kategori Pengguna	Tugas	Hak Akses	Kemampuan yang harus dimiliki
Mahasiswa dan Dosen	Menggunakan fitur-fitur yang terdapat pada sistem.	*User*	Dapat mengoperasikan *smartphone* android.
Administrator	Mengelola pengguna di dalam sistem dan me-*reset password* pengguna.	Admin	Dapat mengoperasikan *smartphone* android.

Sumber : [Analisis Kebutuhan]

Administrator adalah pengguna yang memiliki hak untuk mengelola *user* di dalam aplikasi jejaring sosial ini. Aktor ini memiliki beberapa hak akses ke sistem antara lain adalah sebagai berikut :

1. Mencari *user* berdasarkan *userid.*
2. Me-*reset password* dari *user*.
3. Menghapus *user*.

User adalah pengguna biasa dari aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android. Beberapa fasilitas yang dapat diakses oleh *user* di dalam aplikasi ini secara garis besar adalah sebagai berikut :

1. Melakukan autentifikasi ke aplikasi (*Login*).
2. Melihat informasi mengenai daftar *posting* yang ada di aplikasi.
3. Melakukan *posting* ke aplikasi (*add*, *reply* dan *forward posting*).
4. Menghapus *posting.*
5. Mengedit *account*.
6. Mencari *user* berdasarkan *username*.
7. Menambah teman.
8. Menghapus teman.
9. Melihat informasi lokasi dimana *user* lain berada.
10. Melihat jalur yang dapat dilewati untuk menuju lokasi *user* lain.

4.2.2 Daftar Kebutuhan

Daftar kebutuhan terdiri dari kebutuhan fungsional dan kebutuhan non-fungsional yang harus disediakan oleh sistem. Daftar kebutuhan ini disebut dengan *Software Requirement Specification* (SRS). Daftar spesifikasi kebutuhan fungsional keseluruhan sistem dispesifikasikan menjadi dua bagian yaitu spesifikasi kebutuhan fungsional *user* dan spesifikasi kebutuhan fungsional *administrator*. SRS yang ditampilkan pada skripsi ini adalah SRS pada *increment* ke tiga dari aplikasi jejaring sosial yang dibuat. *Change log* dari *increment-increment* sebelumnya dapat dilihat pada Lampiran 3. Spesifikasi kebutuhan fungsional *user* ditunjukkan pada Tabel 4.2.

Tabel 4.2 Daftar kebutuhan fungsional *user*

SRS-Id	Deskripsi	*Use Case*
SRS-001-01	Sistem harus menyediakan fasilitas untuk membuat *account* di sistem.	Registrasi
SRS-001-02	Sistem harus menyediakan fasilitas untuk melakukan autentifikasi ke sistem.	*Login*
SRS-001-03	Sistem harus menyediakan komponen untuk *user* yang telah *login* dapat keluar dari sistem.	*Logout*

SRS-Id	Deskripsi	*Use Case*
SRS-002-01	Sistem harus menyediakan komponen untuk melakukan *posting* ke sistem.	Melakukan *Posting*
SRS-002-02	Sistem harus menyediakan fasilitas untuk melihat *timeline* dari daftar *posting*.	Melihat *Timeline*
SRS-002-03	Sistem harus menyediakan komponen untuk menghapus *posting* yang dilakukan.	Menghapus *Posting*
SRS-003-01	Sistem harus menyediakan fasilitas untuk melihat informasi lokasi dari *user* dalam bentuk peta (*map*).	Melihat Lokasi
SRS-003-02	Sistem harus menyediakan komponen di dalam peta (*map*) untuk melihat jalur yang dapat dilalui oleh seorang *user* menuju lokasi *user* lain.	Melihat Jalur
SRS-004-01	Sistem harus menyediakan fasilitas untuk mencari *user* berdasarkan nama *user*.	Mencari Teman
SRS-004-02	Sistem harus menyediakan fasilitas untuk melihat info dari seorang *user*.	Melihat Info
SRS-004-03	Sistem harus menyediakan fasilitas untuk seorang *user* dapat menambah *user* lain sebagai teman.	Menambah Teman
SRS-004-04	Sistem harus menyediakan fasilitas untuk menghapus teman.	Menghapus Teman
SRS-005-01	Sistem harus menyediakan fasilitas untuk *user* dapat mengganti data pribadinya seperti biodata dan *password*.	*Edit Account*

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah fungsionalitas untuk admin. Spesifikasi kebutuhan fungsional untuk admin ditunjukkan pada Tabel 4.3.

Tabel 4.3 Daftar kebutuhan fungsional admin

SRS-Id	Deskripsi	*Use Case*
SRS-006-01	Sistem harus menyediakan fasilitas untuk melakukan autentifikasi ke sistem.	*Login*
SRS-006-02	Sistem harus menyediakan fasilitas bagi admin untuk mencari *user* berdasarkan *user id*.	Mencari *User*
SRS-006-03	Sistem harus menyediakan fasilitas bagi admin untuk mengeset ulang (*reset*) *password* pengguna.	*Reset Password*
SRS-006-04	Sistem harus menyediakan fasilitas bagi admin untuk menghapus *user*.	Menghapus *User*

Sumber : [Analisis Kebutuhan]

Kebutuhan non-fungsional juga harus disediakan oleh sistem. Kebutuhan non-fungsional merupakan batasan layanan atau fungsi yang ditawarkan sistem seperti batasan waktu, batasan pengembangan proses, dan standarisasi. Kebutuhan non-fungsional berikut tidak akan diuji karena tidak menjadi fokus dari skripsi ini. Kebutuhan non-fungsional yang disebutkan adalah kebutuhan non-fungsional yang umum dimiliki oleh aplikasi jejaring sosial. Daftar kebutuhan non-fungsional aplikasi jejaring sosial berbasis GPS pada *smartphone* android ditunjukan pada Tabel 4.4.

Tabel 4.4 Daftar kebutuhan non-fungsional sistem

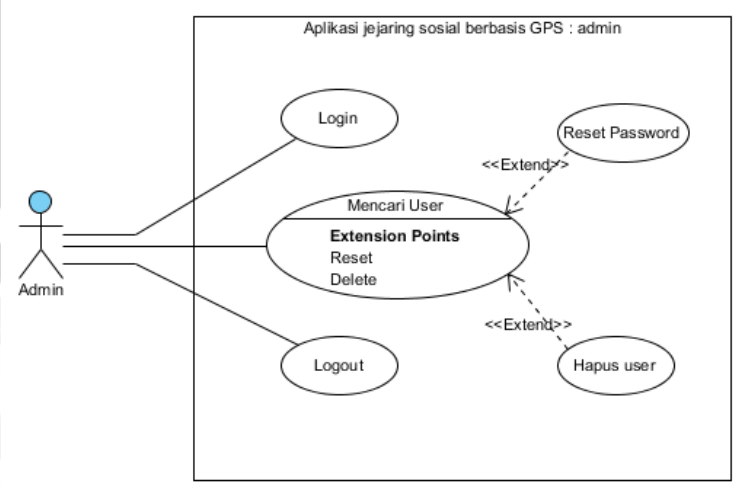
Parameter	**Deskripsi kebutuhan**
Availability	Aplikasi ini harus dapat beroperasi terus menerus selama tujuh hari per minggu, 24 jam per hari tanpa berhenti. Untuk itu dibutuhkan *server* yang selalu *online* setiap saat.
Response time	Aplikasi ini harus memiliki respon yang cepat dengan *bandwith* minimal 64 *kbps*.

Security	Aplikasi ini harus aman. Karena terdapat informasi-informasi penting, maka faktor keamanan menjadi sangat penting. *Security* pada sistem ini menggunakan fungsi *Login* dengan enkripsi *password* menggunakan MD5 bertingkat dan pengacak.
Memory	Aplikasi ini harus ringan dan tidak membutuhkan *memory* tinggi sehingga aplikasi ini dapat dijalankan pada *smartphone* android dengan spesifikasi rendah minimal *RAM* 128 *MB*.

Sumber : [Analisis Kebutuhan]

4.2.3 Diagram *Use Case*

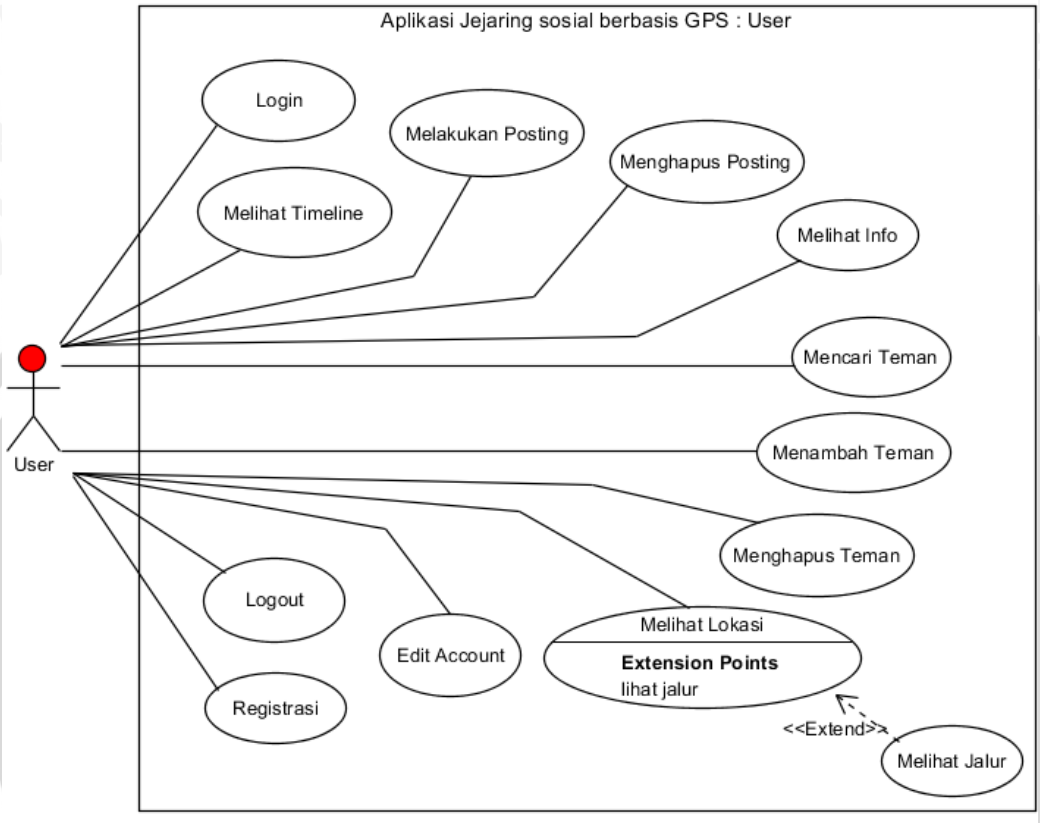
Kebutuhan-kebutuhan fungsional yang diperlukan oleh pengguna dan harus disediakan oleh sistem akan dimodelkan pada diagram *use case*. Diagram *use case* digunakan untuk memodelkan aspek perilaku sistem. Diagram *use case* menunjukkan sekumpulan *use case*, aktor, dan hubungannya. Pada diagram *use case* aplikasi ini terdapat dua aktor yaitu *user* dan *admin*. Pemodelan diagram *use case* aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android dibagi menjadi dua yaitu *use case diagram* untuk *user* dan *use case diagram* untuk *administrator*. *Use case diagram* aplikasi jejaring sosial kampus berbasis GPS untuk *administrator* ditunjukkan pada Gambar 4.3.



Gambar 4.3 Diagram *use case* untuk admin

Sumber : [Analisis Kebutuhan]

Diagram *use case* selanjutnya adalah *use case* untuk *user*. Diagram *use case* ini melibatkan *user* (mahasiswa dan dosen) sebagai aktor dan 13 buah *use case*. *Use case diagram* aplikasi jejaring sosial kampus berbasis GPS untuk *user* ditunjukkan pada Gambar 4.4.



Gambar 4.4 Diagram *use case* untuk *user*

Sumber : [Analisis Kebutuhan]

4.2.4 Skenario *Use Case*

Masing-masing *use case* yang terdapat pada diagram *use case*, dijabarkan dalam skenario *use case* secara detail. Skenario *use case* akan memuat nama *use case*, aktor di dalam *use case* tersebut, tujuan dari *use case*, deskripsi global tentang *use case*, kondisi awal yang harus dipenuhi, dan kondisi akhir yang diharapkan setelah berjalannya fungsional *use case*. Selain itu juga akan diberikan ulasan yang berkaitan dengan tanggapan dari sistem atas suatu aksi yang diberikan oleh aktor (aliran utama atau *main flow*), serta kejadian alternatif yang akan terjadi jika suatu kondisi tidak bisa terpenuhi (aliran alternatif atau *alternatif flow*). Tabel 4.5 menunjukkan skenario dari *use case* Registrasi.

Tabel 4.5 Skenario *use case* Registrasi

Nama Use Case	Registrasi
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk melakukan registrasi data *user* ke sistem
Deskripsi	*Use case* ini digunakan untuk menambahkan data *user* ke basis data.
Kondisi Awal	Aktor belum mempunyai *userid* untuk akses ke sistem dan membuka menu *Sign Up.*
Kondisi Akhir	Aktor berhasil mendapatkan *userid* dan *password* yang dapat digunakan untuk mengakses sistem.
Main Flow	1. Sistem menampilkan antarmuka untuk registrasi *user.* 2. Aktor memasukkan data-data (*userid*, *password*, *nama* dan lain-lain) yang diminta kemudian menekan tombol "Register". 3. Sistem memeriksa data-data yang diisikan aktor. 4. Jika data yang ditambahkan benar atau sesuai maka sistem akan menyimpan data-data tersebut ke basis data.
Alternatif Flow	A. Eksepsi jika gagal melakukan registrasi. Menampilkan pernyataan peringatan kesalahan (*alert*) jika *user* gagal melakukan registrasi. Kegagalan proses registrasi ini diakibatkan oleh *userid* sudah terdaftar atau masih ada *field* yang belum diisi.

Sumber : Analisis Kebutuhan

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melakukan *login* ke sistem. Kebutuhan tersebut direpresentasikan oleh *use case Login*. Tabel 4.6 merupakan skenario *use case Login.*

Tabel 4.6 Skenario *use case Login*

Nama Use Case	*Login*
Aktor	Admin *dan User* (mahasiswa dan dosen)
Tujuan	Untuk melakukan autentifikasi *user* ke sistem

Deskripsi	*Use case* ini digunakan untuk memperoleh akses ke sistem. *Login* didasarkan pada *userid* dan *password* masing-masing aktor.
Kondisi Awal	Aktor mengakses menu *Login*.
Kondisi Akhir	Aktor mendapat hak akses untuk menggunakan fitur-fitur pada sistem.
Main Flow	1. Sistem menampilkan *form* untuk *login*. 2. Aktor memasukkan *userid* dan *password* kemudian menekan tombol "*Login*". 3. Sistem memeriksa *userid* dan *password* aktor. 4. Sistem memberikan otoritas ke aktor sesuai hak aksesnya.
Alternatif Flow	A. Eksepsi jika gagal melakukan *login*. Menampilkan pernyataan peringatan kesalahan (*alert*) jika *user* gagal melakukan *login*. Kegagalan proses *login* ini diakibatkan oleh *userid* dan *password* tidak valid atau *field userid* yang belum diisi. B. Registrasi *User* dapat melakukan registrasi dengan mengakses menu *Sign Up*.

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk keluar dari sistem (*logout)*. Kebutuhan tersebut direpresentasikan oleh *use case Logout*. Tabel 4.7 merupakan skenario *use case Logout*.

Tabel 4.7 Skenario *use case Logout*

Nama Use Case	*Logout*
Aktor	Admin *dan User* (mahasiswa dan dosen).
Tujuan	Keluar dari autentifikasi *user*.
Deskripsi	*Use case* ini digunakan untuk keluar dari autentifikasi *user*. *Logout* dilakukan agar aktor yang telah *login* dapat keluar dari sistem.
Kondisi Awal	Aktor telah *login* ke sistem.

Kondisi Akhir	Aktor berhasil melakukan proses *logout* dan keluar dari autentifikasi sistem.
Main Flow	1. Sistem menampilkan antarmuka untuk *logout*. 2. Aktor kemudian menekan *option* "*Logout*". 3. Sistem menghapus *session* yang aktif. 4. Sistem menampilkan menu *login*.
Alternatif Flow	-

Sumber : Analisis Kebutuhan

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melakukan *posting*. Kebutuhan tersebut direpresentasikan oleh *use case* Melakukan *Posting*. Tabel 4.8 merupakan skenario *use case* Melakukan *Posting*.

Tabel 4.8 Skenario *use case* Melakukan *Posting*

Nama *Use Case*	Melakukan *Posting*
Aktor	*User* (mahasiswa dan dosen).
Tujuan	Untuk melakukan *posting* ke sistem.
Deskripsi	*Use case* ini digunakan untuk melakukan *posting* data dari *user* ke sistem.
Kondisi Awal	Aktor telah *login* dan membuka menu *posting.*
Kondisi Akhir	*Timeline* akan ter-*update* sesuai dengan *posting* yang dilakukan oleh aktor.
Main Flow	1. Sistem menampilkan antarmuka untuk *posting* data. 2. Aktor memasukkan inputan *posting* kemudian menekan tombol "Post". 3. Sistem menyimpan data-data tersebut ke basis data. 4. Sistem mengupdate *timeline* dengan *posting* terbaru.
Alternatif Flow	A. Eksepsi jika gagal melakukan *posting*. Menampilkan pernyataan peringatan kesalahan (*alert*) jika *user* gagal melakukan *posting*. Kegagalan proses *posting* ini diakibatkan oleh *fieldposting* yang belum diisi.

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melihat *timeline*. Kebutuhan tersebut direpresentasikan oleh *use case* Melihat *Timeline*. Tabel 4.9 merupakan skenario *use case* Melihat *Timeline*.

Tabel 4.9 Skenario *use case* Melihat *Timeline*

Nama *Use Case*	Melihat *Timeline*
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk melihat *timeline* dari *posting* yang dilakukan oleh *user*.
Deskripsi	*Use case* ini digunakan untuk melakukan proses pembacaan data *posting* dari *user* yang ditampilkan dalam bentuk *timeline*.
Kondisi Awal	Aktor telah *login* dan masuk ke menu *home.*
Kondisi Akhir	Aktor berhasil melihat informasi dari *posting* dalam bentuk *Timeline*.
Main Flow	1. Sistem menyediakan antarmuka dalam bentuk *timeline* untuk menampilkan data-data *posting* yang dilakukan dan data *posting* dari *user* lain yang terdaftar sebagai teman. 2. Aktor melihat informasi *posting*.
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk menghapus *posting* yang dilakukan oleh aktor. Kebutuhan tersebut direpresentasikan oleh *use case* Hapus *Posting*. Tabel 4.10 merupakan skenario *use case* Hapus *Posting*.

Tabel 4.10 Skenario *use case* Hapus *Posting*

Nama *Use Case*	Hapus *Posting*
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk menghapus data *posting* yang dilakukan oleh aktor.

Deskripsi	*Use case* ini digunakan untuk melakukan proses penghapusan data *posting* yang dilakukan oleh *user.*
Kondisi Awal	Aktor mengakses *timeline* dari menu *posting.*
Kondisi Akhir	Aktor berhasil menghapus data yang telah di-*posting*.
Main Flow	1. Sistem menampilkan data-data *posting* dari *user* dalam bentuk *timeline*. 2. Aktor memilih *posting* yang akan dihapus dari *timeline* kemudian menekan *context* menu "*Delete*". 3. Sistem menghapus data *posting* dari *database.*
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melihat lokasi dari *user* lain. Kebutuhan tersebut direpresentasikan oleh *use case* Melihat Lokasi. Tabel 4.11 merupakan skenario *use case* Melihat Lokasi.

Tabel 4.11 Skenario *use case* Melihat Lokasi

Nama *Use Case*	Melihat Lokasi
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk melihat lokasi dari *user* lain.
Deskripsi	*Use case* ini digunakan untuk melakukan proses pembacaan data lokasi dari *user* yang kemudian ditampilkan dalam bentuk peta (*map*).
Kondisi Awal	Aktor telah *login* dan masuk ke menu *map.*
Kondisi Akhir	Aktor berhasil melihat informasi lokasi dari *user* lain.
Main Flow	1. Sistem menyediakan antarmuka dalam bentuk peta (*map*) untuk menampilkan lokasi dari *user* lain. 2. Aktor melihat informasi lokasi *user* lain.
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melihat jalur yang harus dilalui untuk menuju ke lokasi dari *user* lain. Kebutuhan tersebut direpresentasikan oleh *use case* Melihat Jalur. Tabel 4.12 merupakan skenario *use case* Melihat Jalur.

Tabel 4.12 Skenario *use case* Melihat Jalur

Nama *Use Case*	Melihat Jalur
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk melihat jalur yang harus dilalui menuju lokasi dari *user* lain.
Deskripsi	*Use case* ini digunakan untuk melakukan proses pembacaan jalur yang harus dilalui menujulokasi dimana *user* lain berada yang kemudian ditampilkan di dalam peta (*map*).
Kondisi Awal	Aktor telah *login* dan masuk ke menu *map*.
Kondisi Akhir	Aktor berhasil melihat jalur yang harus dilalui.
Main Flow	1. Sistem menyediakan antarmuka dalam bentuk peta (*map*) untuk menampilkan lokasi dari *user* lain. 2. Aktor melihat informasi lokasi *user* lain kemudian menekan tombol dengan *icon* jalur. 3. Sistem menampilkan jalur yang harus dilalui di dalam peta.
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk mencari teman. Kebutuhan tersebut direpresentasikan oleh *use case* Mencari Teman. Tabel 4.13 merupakan skenario *use case* Mencari Teman.

Tabel 4.13 Skenario *use case* Mencari Teman

Nama *Use Case*	Mencari Teman
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk melakukan pencarian *user* berdasarkan *username*.

Deskripsi	*Use case* ini digunakan untuk melakukan proses pencarian *user* dengan memasukkan *username* sebagai parameternya.
Kondisi Awal	Aktor telah *login* dan membuka menu *search.*
Kondisi Akhir	Aktor berhasil menemukan *user* yang sesuai dengan *username* yang dicari.
Main Flow	1. Sistem menampilkan antarmuka untuk pencarian *user.* 2. Aktor memasukkan *username* kemudian menekan tombol dengan *icon* bergambar kaca pembesar. 3. Sistem mencari *user* yang sesuai dengan data *input* di dalam basis data. 4. Sistem menampilkan daftar *user* yang sesuai dengan *input* aktor.
Alternatif Flow	A. Eksepsi jika gagal melakukan pencarian. Menampilkan pernyataan peringatan kesalahan (*alert*) jika *user* gagal melakukan pencarian. Kegagalan proses pencarian ini diakibatkan oleh *field username* yang belum diisi atau *user* yang dicari tidak ada dalam *database*.

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk melihat info dari seorang *user*. Kebutuhan tersebut direpresentasikan oleh *use case* Melihat Info. Tabel 4.14 merupakan skenario *use case* Melihat Info.

Tabel 4.14 Skenario *use case* Melihat Info

Nama *Use Case*	Melihat Info
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk melihat informasi dari seorang *user*.
Deskripsi	*Use Case* ini digunakan untuk melakukan proses pembacaan data personal dari seorang *user*.
Kondisi Awal	Aktor telah *login* dan membuka menu *profile* dari seorang *user*.
Kondisi Akhir	Aktor berhasil melihat informasi dari seorang *user*.

Main Flow	1. Sistem menampilkan menu *profile* dari seorang *user.* 2. Aktor melihat info dari *user* tersebut dengan menekan menu "*Details*". 3. Sistem menampilkan info dari *user* tersebut.
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk menambah teman (*follow*). Kebutuhan tersebut direpresentasikan oleh *use case* Menambah Teman. Tabel 4.15 merupakan skenario *use case* Mencari Teman.

Tabel 4.15 Skenario *use case* Menambah Teman

Nama *Use Case*	Menambah Teman
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk menambahkan *user* lain sebagai teman.
Deskripsi	*Use case* ini digunakan untuk melakukan proses penambahan seorang *user* menjadi teman (*follow*).
Kondisi Awal	Aktor telah *login* dan membuka menu *profile* dari *user* lain.
Kondisi Akhir	Aktor berhasil menambahkan seorang *user* menjadi teman.
Main Flow	1. Sistem menampilkan menu *profile* dari *user* lain. 2. Aktor menambahkan *user* tersebut sebagai teman dengan menekan tombol "+". 3. Sistem menambahkan *user* tersebut sebagai teman baru di dalam basis data. 4. Sistem menampilkan notifikasi bahwa proses menambah teman berhasil. 5. Sistem menampilkan *posting* yang dilakukan oleh teman.
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk menghapus teman (*unfollow*). Kebutuhan tersebut direpresentasikan oleh *use case* Menghapus Teman. Tabel 4.16 merupakan skenario *use case* Menghapus Teman.

Tabel 4.16 Skenario *use case* Menghapus Teman

Nama *Use Case*	Menghapus Teman
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk menghapus teman dari basis data.
Deskripsi	*Use Case* ini digunakan untuk melakukan proses menghapus teman dari basis data (*unfollow*).
Kondisi Awal	Aktor telah *login* dan membuka menu *profile* dari *user* lain yang sudah menjadi teman.
Kondisi Akhir	Aktor berhasil menghapus teman.
Main Flow	1. Sistem menampilkan menu *profile* dari *user* yang sudah menjadi teman. 2. Aktor menghapus *user* tersebut sebagai teman dengan menekan tombol "-". 3. Sistem menampilkan dialog yang menanyakan apakah aktor yakin akan menghapus teman. 4. Aktor menekan tombol "OK" jika yakin akan menghapus teman. 5. Sistem menghapus *user* tersebut sebagai teman di dalam basis data. 6. Sistem menampilkan notifikasi bahwa proses menghapus teman berhasil. 7. Sistem menghilangkan *posting* yang dilakukan oleh teman yang telah dihapus.
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk mengedit *account*. Kebutuhan tersebut direpresentasikan oleh *use case Edit Account*. Tabel 4.17 merupakan skenario *use case Edit Account*.

Tabel 4.17 Skenario *use case Edit Account*

Nama *Use Case*	*Edit Account*
Aktor	*User* (mahasiswa dan dosen)
Tujuan	Untuk mengganti biodata dan *password* dari *user.*
Deskripsi	*Use Case* ini digunakan untuk melakukan proses mengganti biodata dan *password* dari *user*.
Kondisi Awal	Aktor telah *login* dan membuka menu *setting.*
Kondisi Akhir	Aktor berhasil mengganti biodata dan *password*.
Main Flow	1. Sistem menampilkan menu *setting.* 2. Aktor mengganti biodata atau *password* kemudian menekan tombol “Save”. 3. Sistem memeriksa data-data yang diisikan aktor. 4. Jika data yang ditambahkan benar atau sesuai maka sistem akan meng-*update* data-data tersebut ke basis data. 5. Sistem menampilkan notifikasi bahwa proses *edit* berhasil.
Alternatif Flow	A. Eksepsi jika gagal melakukan *edit account*. Menampilkan pernyataan peringatan kesalahan (*alert*) jika *user* gagal mengganti biodata atau *passsword*. Kegagalan proses *edit* ini diakibatkan oleh *field* biodata ada yang belum diisi atau *password* lama yang diisikan tidak sesuai.

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah fasilitas bagi admin mencari *user*. Kebutuhan tersebut direpresentasikan oleh *use case* Mencari *User*. Tabel 4.18 merupakan skenario *use case* Mencari *User*.

Tabel 4.18 Skenario *use case* Mencari *User*

Nama *Use Case*	Mencari *User*
Aktor	Admin
Tujuan	Untuk melakukan pencarian *user* berdasarkan *userid.*

Deskripsi	*Use case* ini digunakan untuk melakukan proses pencarian *user* dengan memasukkan *userid* sebagai parameternya.
Kondisi Awal	Aktor telah *login* dan mengakses menu *search.*
Kondisi Akhir	Aktor berhasil menemukan *user* yang sesuai dengan *userid* yang dicari.
Main Flow	1. Sistem menampilkan antarmuka untuk pencarian *user.* 2. Aktor memasukkan *userid* kemudian menekan tombol dengan *icon* bergambar kaca pembesar. 3. Sistem mencari *user* yang sesuai dengan data *input* di dalam basis data. 4. Sistem menampilkan *user* yang sesuai dengan *input*.
Alternatif Flow	A. Eksepsi jika gagal melakukan pencarian. Menampilkan pernyataan peringatan kesalahan (*alert*) jika *user* gagal melakukan pencarian. Kegagalan proses pencarian ini diakibatkan oleh *user* yang dicari tidak ada dalam basis data atau *field userid* yang belum diisi.

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah fasilitas bagiadmin untuk me-*reset password user*. Kebutuhan tersebut direpresentasikan oleh *use case Reset Password*. Tabel 4.19 merupakan skenario *use case Reset Password*.

Tabel 4.19 Skenario *use case Reset Password*

Nama Use Case	*Reset Password*
Aktor	Admin
Tujuan	Untuk melakukan proses *reset password* dari seorang *user.*
Deskripsi	*Use case* ini digunakan untuk melakukan proses *reset password* dari seorang *user*.
Kondisi Awal	Aktor telah *login* dan berhasil mencari seorang *user.*
Kondisi Akhir	Aktor berhasil me-*reset password* dari seorang *user*.

Main Flow	1. Sistem menampilkan daftar *user* dalam bentuk *list*. 2. Aktor memilih *user* yang akan di-*reset password*-nya dari *list* kemudian menekan *context* menu "*Reset Password*". 3. Aktor memasukkan *input password* yang baru. 4. Sistem meng-*update password* di *database.* 5. Sistem menampilkan notifikasi bahwa proses *reset password* telah berhasil.
Alternatif Flow	A. Eksepsi jika gagal melakukan *reset password*. Menampilkan pernyataan peringatan kesalahan (*alert*) jika admin gagal melakukan *reset password*. Kegagalan proses *reset password* ini diakibatkan oleh *password* yang dimasukan tidak sama atau masih ada *field* yang belum diisi.

Sumber : [Analisis Kebutuhan]

Kebutuhan fungsional selanjutnya yang harus disediakan oleh sistem adalah kebutuhan untuk admin menghapus *user*. Kebutuhan tersebut direpresentasikan oleh *use case* Hapus *User*. Tabel 4.20 merupakan skenario *use case* Hapus *User*.

Tabel 4.20 Skenario *use case* Hapus *User*

Nama *Use Case*	Hapus *User*
Aktor	Admin
Tujuan	Untuk menghapus data *user* dari basis data.
Deskripsi	*Use case* ini digunakan untuk melakukan proses penghapusan data *user* dari basis data.
Kondisi Awal	Aktor telah *login* dan berhasil mencari seorang *user*.
Kondisi Akhir	Aktor berhasil menghapus data *user* dari basis data.
Main Flow	1. Sistem menampilkan daftar *user* dalam bentuk *timeline*. 2. Aktor memilih *user* yang akan dihapus dari *timeline* kemudian menekan *context* menu "*Delete User*". 3. Sistem menghapus data *user* dari *database.*
Alternatif Flow	-

Sumber : [Analisis Kebutuhan]

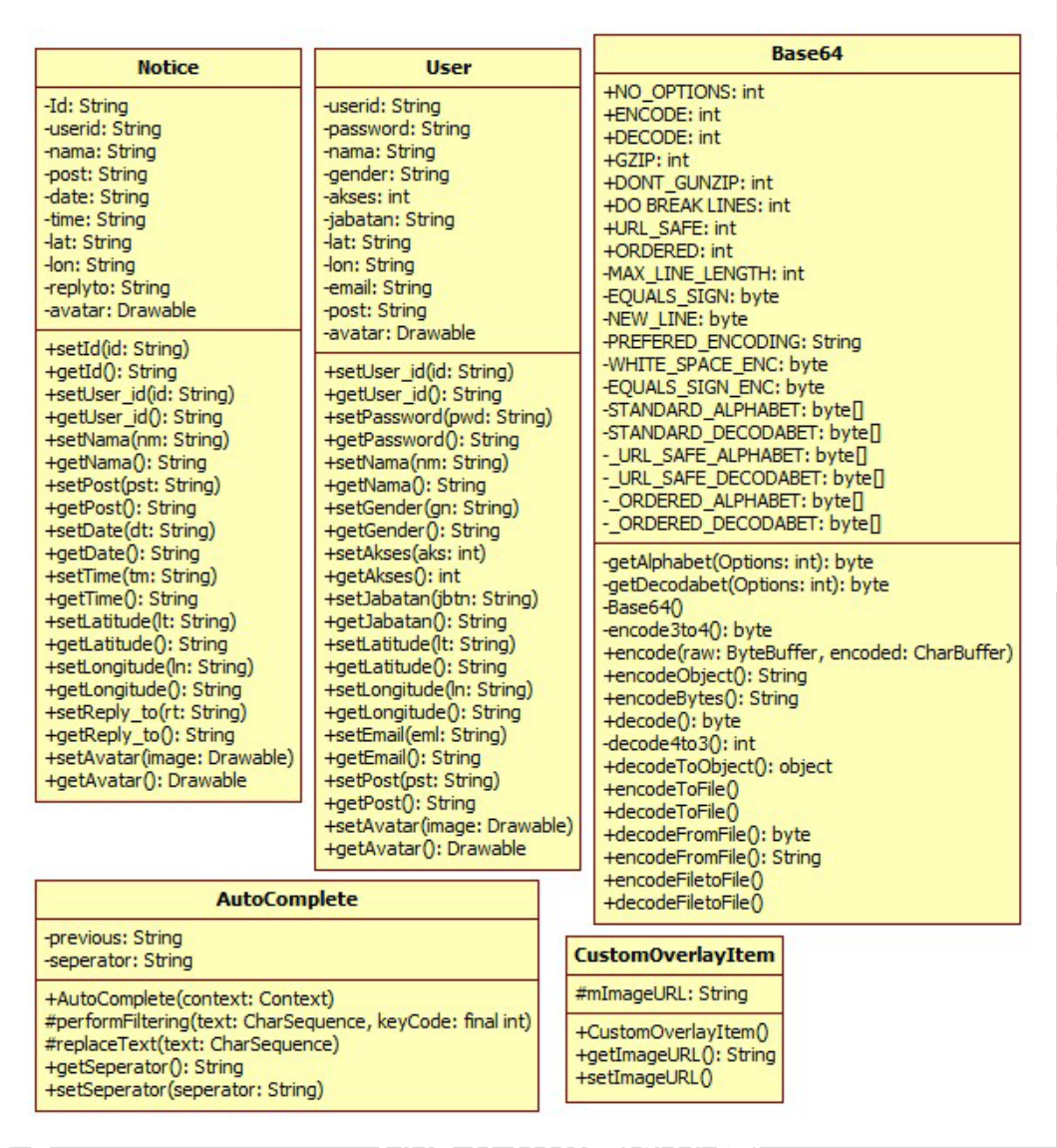
4.3 Perancangan Perangkat Lunak

Perancangan perangkat lunak pada skripsi ini menggunakan pendekatan desain berorientasi objek yang direpresentasikan dengan menggunakan UML (*Unified Modeling Language*). Pada proses desain dilakukan identifikasi terhadap *class-class* yang dibutuhkan yang dimodelkan dalam *class diagram*. Hubungan interaksi antar elemen (objek) yang telah diidentifikasi, dimodelkan dalam *sequence diagram*. Basisdata dimodelkan dalam ER diagram.

4.3.1 Diagram Klas (*Class Diagram*)

Pada perancangan perangkat lunak ini terdapat sejumlah *class* yang saling membentuk relasi. *Class-class* yang akan membangun aplikasi ini dikelompokkan pada beberapa paket yaitu paket android.logic, android.campus, android.adapter dan android.overlay.

Paket android.logic terdiri dari lima buah *class* yaitu *class* User, *class* Notice ,*class* AutoComplete, *class* CustomOverlayItem dan *class* Base64. *Class* User adalah *entity class* yang mewakili tabel *user* pada *database*. *Class* ini berfungsi untuk menyediakan operasi *setter* dan *getter* yang berguna untuk memanipulasi nilai tabel *user* pada *database*. *Class* Notice adalah *entity class* yang mewakili tabel *notice* pada *database*. *Class* ini berfungsi untuk menyediakan operasi *setter* dan *getter* yang berguna untuk memanipulasi nilai tabel *notice* pada *database*. *Class* AutoComplete adalah *class* yang berfungsi untuk memanipulasi *Auto Complete Text* pada android. *Class* CustomOverlayItem adalah *class* yang berfungsi untuk memanipulasi *map overlay item* standar dari android. *Class* Base64 adalah *class* yang berfungsi sebagai *controller* untuk mengatur dan menangani proses manipulasi data *image* (gambar) sehingga dapat dikenali oleh *server*. Operasi yang ada di dalam *class* Base64 secara garis besar merupakan operasi *encode* gambar ke format *string* yang dapat dikenali *server* dan operasi *decode* untuk mengubah data dengan format *string* dari *server* menjadi *file* gambar. Atribut dan operasi dari tiga *class* tersebut dapat dilihat dalam *class* diagram pada Gambar 4.5.

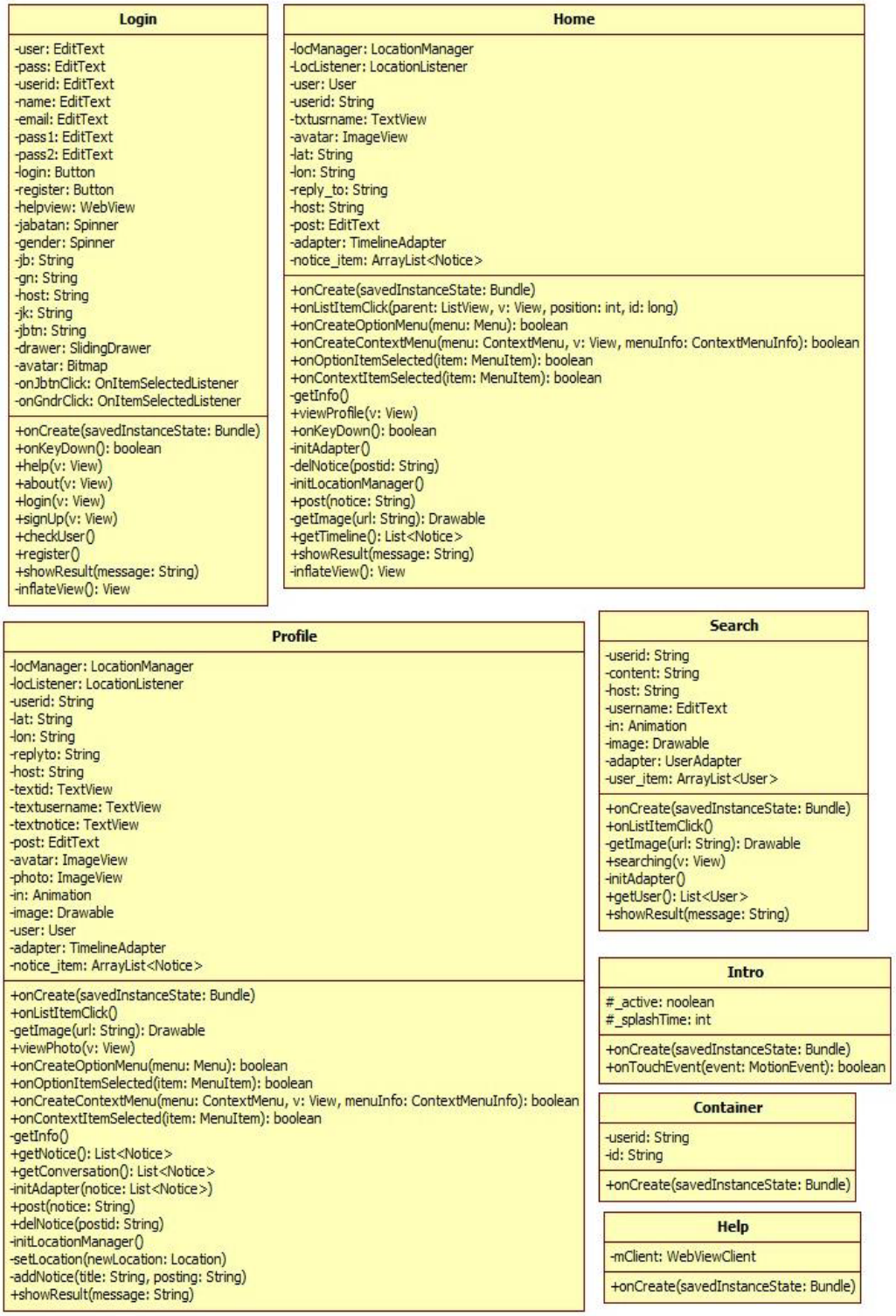


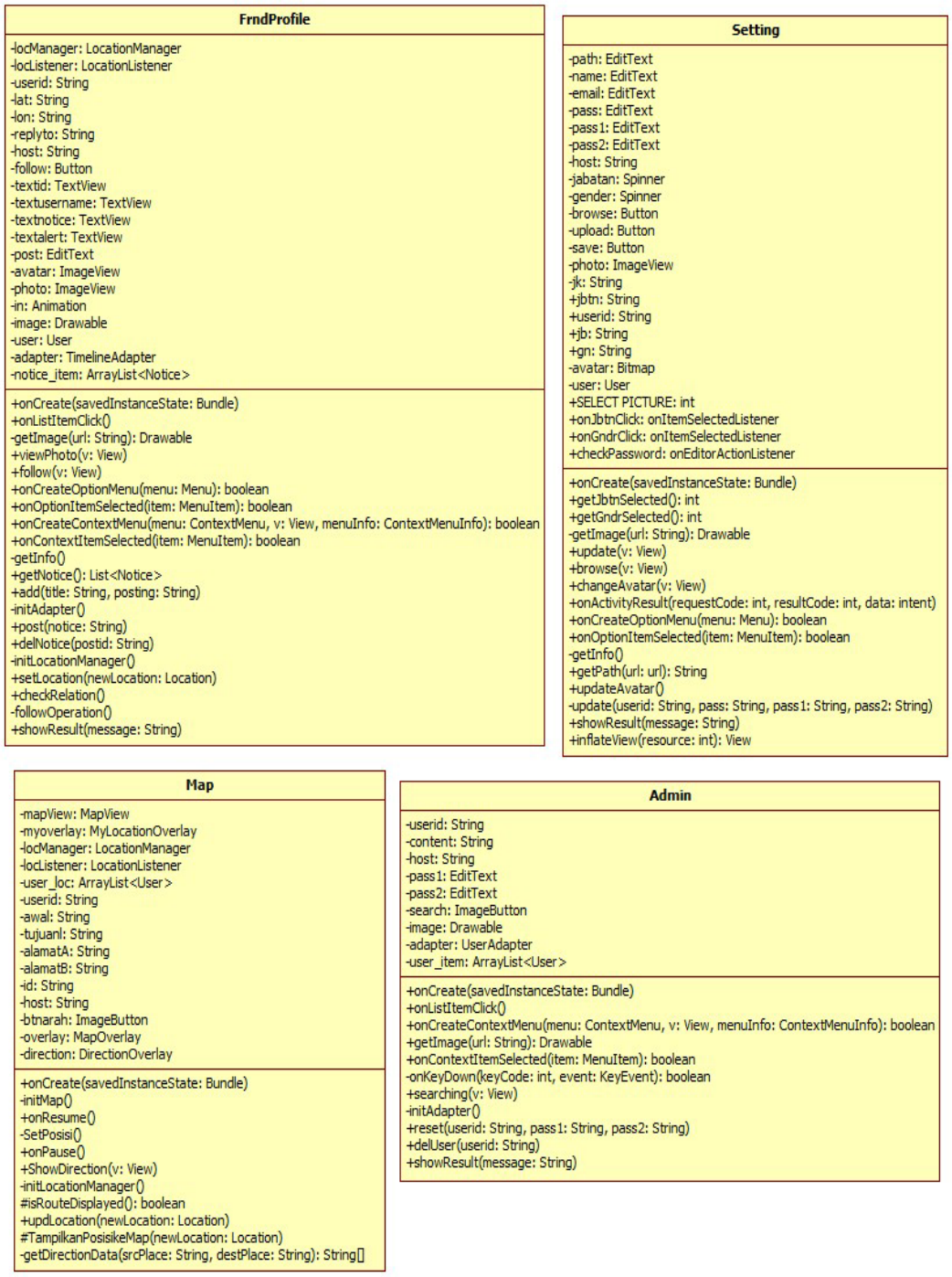
Gambar 4.5 *Class diagram* pada paket android.logic

Sumber : [Perancangan]

Paket android.campus terdiri dari sebelas *class* yaitu *class* Intro, Login, Container, Home, Profile, FrnProfile, Search, Setting, Map, Admin dan Help. Sebelas *class* tersebut merupakan *class* utama pada program. *Class-class* tersebut berperan sebagai *controller* dari *layout* antarmuka aplikasi yang dibuat dengan komponen *graphical user interface* android berupa *file* berformat xml (*.xml). *Class-class* tersebut berfungsi untuk merespon suatu *event* yang terjadi pada antarmuka aplikasi dan menjalankan operasi-operasi yang menjadi hasil *event* tersebut. *Method* `onCreate()` merupakan *method* yang dijalankan pertama kali

saat *activity* dijalankan. *Method* `setContentView()` digunakan untuk inisialisasi *layout* berformat xml pada aplikasi dengan parameter *id* dari *layout* tersebut. Atribut dan operasi dari masing-masing *class* digambarkan dalam diagram *class* pada Gambar 4.6.



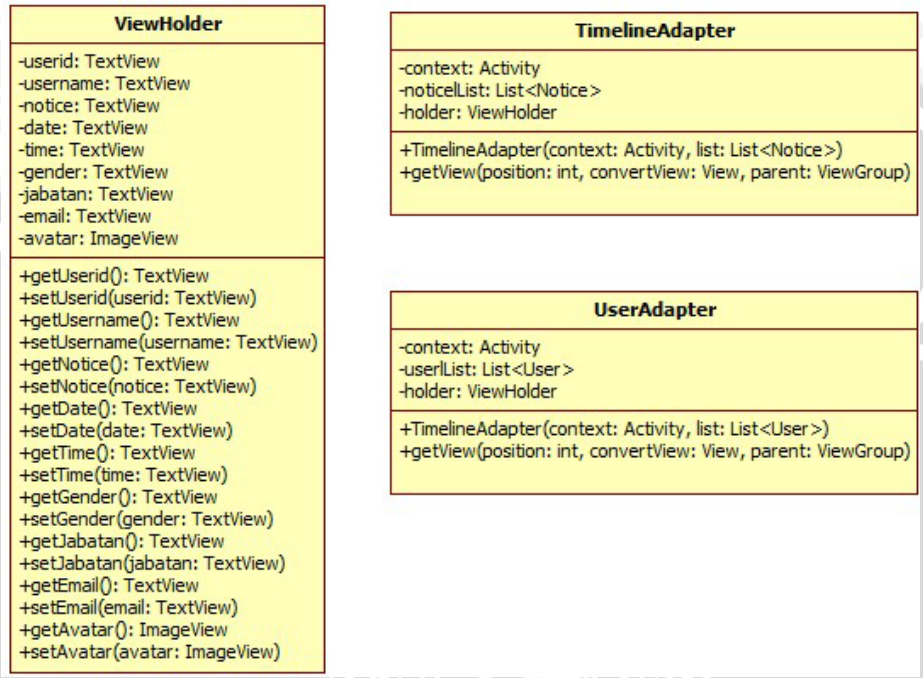


Gambar 4.6 *Class diagram* pada paket android.campus

Sumber : [Perancangan]

Paket android.adapter terdiri dari tiga buah *class* yaitu *class* ViewHolder, TimelineAdapter dan UserAdapter. *Class* ViewHolder merupakan *class* yang berfungsi sebagai penampung dari *view* yang digunakan dalam adater aplikasi. *Class* ini berfungsi untuk menyediakan operasi *setter* dan *getter* yang berguna

untuk manipulasi *view* pada antarmuka. *Class* TimelineAdapter merupakan *class* yang berfungsi sebagai adapter dari *Timeline view* pada aplikasi dengan data *notice* yang ada pada *database*. *Class* UserAdapter merupakan *class* yang berfungsi sebagai adapter dari *list search view* dengan data *user* yang ada pada *database*. Atribut dan operasi dari masing-masing *class* digambarkan dalam diagram *class* pada Gambar 4.7.

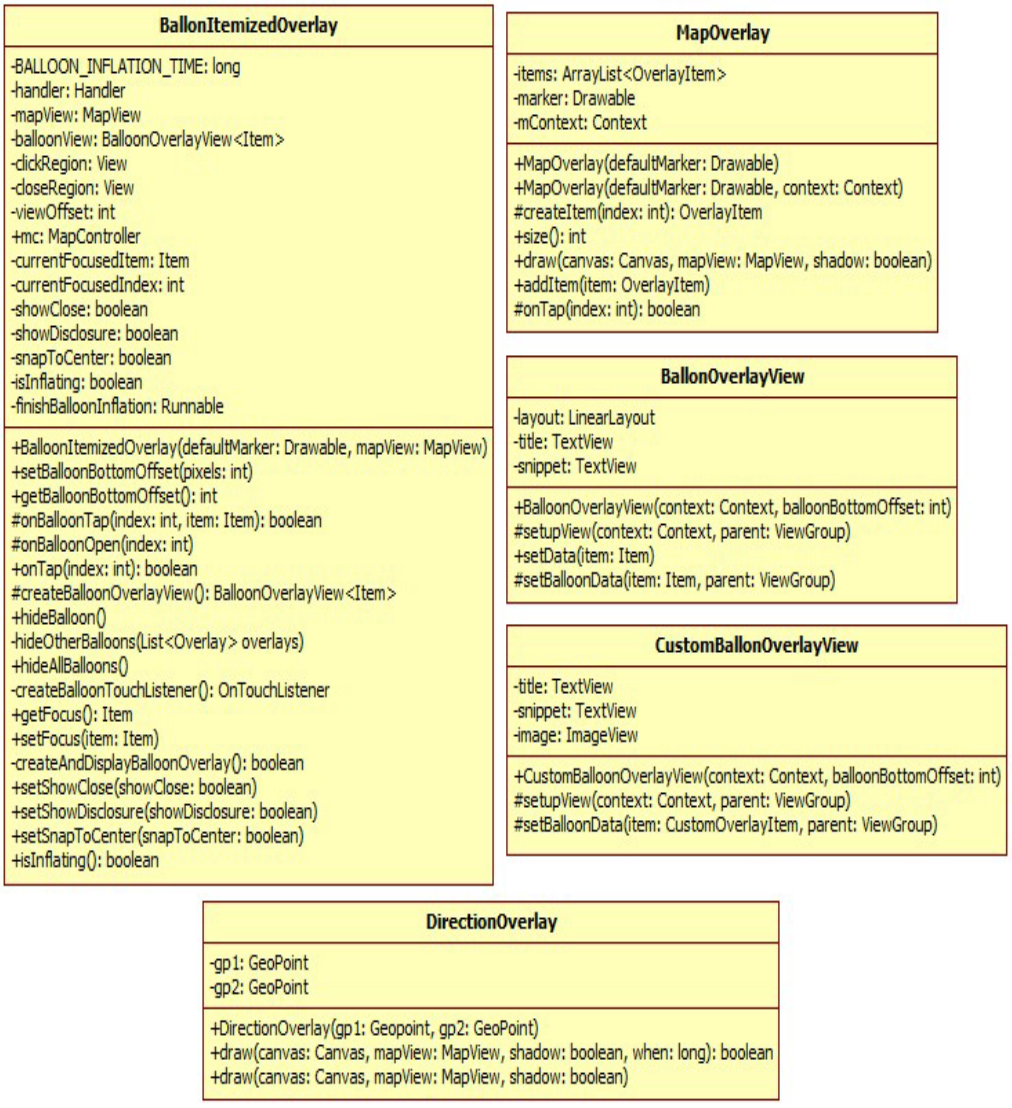


Gambar 4.7 *Class diagram* pada paket android.adapter

Sumber : [Perancangan]

Paket android.overlay terdiri dari lima buah *class* yaitu *class* BallonOverlayView, *class* CustomBallonOverlayView, *class* BallonItemizedOverlay, *class* MapOverlay dan DirectionOverlay. *Class* BallonOverlayView adalah *class* yang berfungsi untuk mengatur *view* dari *overlay* yang ditampilkan pada peta. *Class* CustomBallonOverlayView adalah *class* turunan dari BallonOverlayView yang berfungsi untuk memanipulasi *view* dari data yang ditampilan pada *overlay* peta. BallonItemizedOverlay adalah *class* yang berfungsi sebagai *event handler* dari *overlay* peta. MapOverlay merupakan *class* turunan dari BallonItemizedOverlay yang berperan sebagai *controller* dari *overlay marker* yang akan ditampilkan pada peta. *Class* ini berisi *method-method*

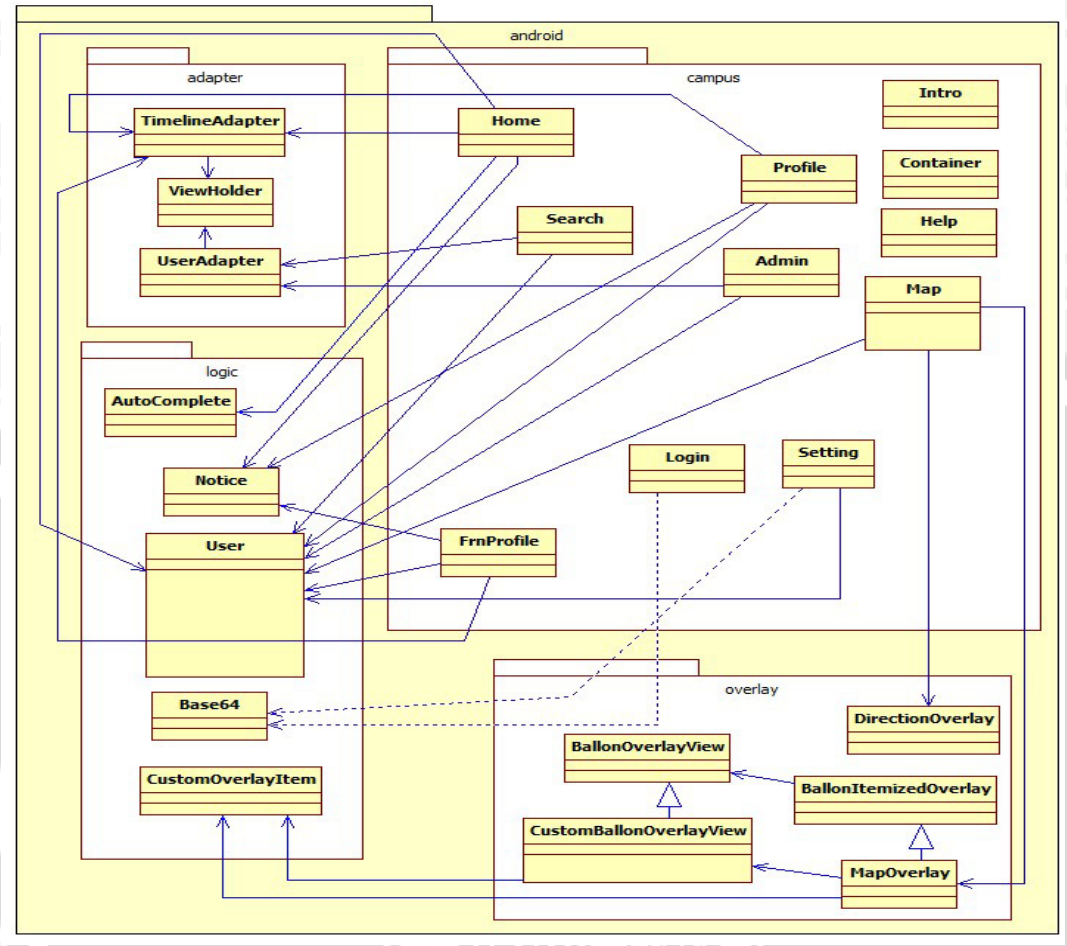
yang secara garis besar merupakan *setting* dari *overlay* peta. DirectionOverlay merupakan *class* yang berperan sebagai *controller* dari *overlay* jalur yang akan ditampilkan pada peta. *Class* ini berisi *method-method* yang secara garis besar merupakan *setting* dari *overlay* jalur yang akan ditampilkan pada peta. Atribut dan operasi dari masing-masing *class* digambarkan dalam diagram *class* pada Gambar 4.8.



Gambar 4.8 *Class diagram* pada paket android.overlay

Sumber : [Perancangan]

Class diagram memberikan gambaran pemodelan elemen-elemen *class* serta fungsi dan relasinya dengan *class* lain dalam sebuah sistem. Relasi antar *class* pada perancangan perangkat lunak ini ditunjukan pada Gambar 4.9.



Gambar 4.9 Relasi antar *class*

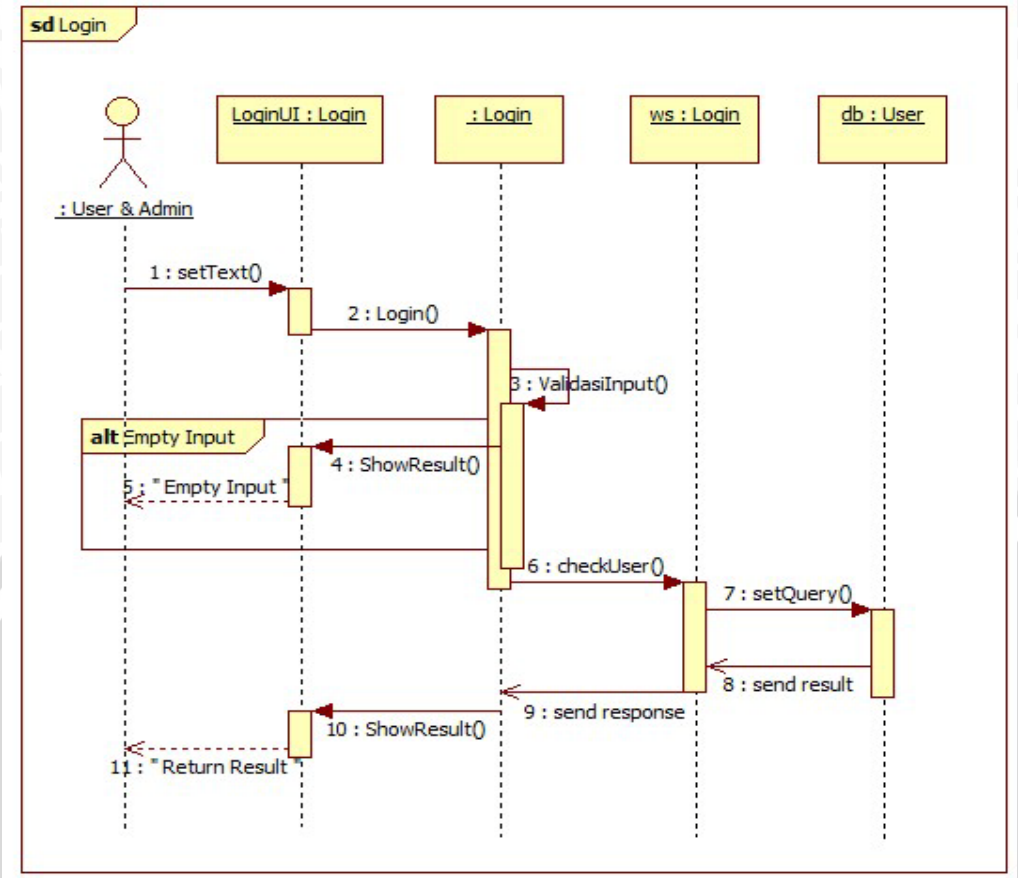
Sumber : [Perancangan]

4.3.2 Diagram Sekuensial (*Sequence Diagram*)

Sequence diagram menunjukkan aliran jalannya proses interaksi antar objek atau klas yang disusun berdasarkan urutan waktu. *Sequence diagram* digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. *Sequence diagram* disusun dengan mengambil acuan pada *use case* dan klas-klas yang membentuk fungsionalitas yang digambarkan pada *use case* tersebut.

Gambar 4.10 merupakan diagram sekuensial pada proses *Login*. Diagram sekuensial ini menggambarkan interaksi ketika *user* dan admin melakukan proses *login* ke sistem. Proses login dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* dan admin melakukan *login* dengan memasukkan *userid* dan *password* pada *EditText* yang tersedia kemudian *method* `setText()` akan aktif.
2. Sistem akan mengaktifkan *method* `login()` pada *class* Login ketika tombol *login* ditekan.
3. *Method* `login()` akan menjalankan validasi *input*, jika field *input* masih kosong maka sistem akan mengaktifkan *method* `showResult()` untuk menampilkan pesan *error* bahwa *userid* dan *password* belum didisi.
4. *Method* `login()` melakukan validasi *user* dengan mengaktifkan *method* `checkUser()`.
5. *Method* `checkUser()` mengirimkan *userid* dan *password* ke *web service.*
6. *Web service* melakukan validasi *user* ke *database* dengan *query* ke *database*.
7. *Web service* mengirimkan respon ke *class* Login.
8. jika *userid* dan *password* sesuai maka nilai kembalian atau respon yang dikirim ke *class* Login adalah "*login* sukses" dan jika tidak sesuai maka nilai kembalian atau respon yang didapat adalah "*userid* dan *password* tidak sesuai".
9. Sistem akan menampilkan notifikasi dari nilai kembalian dengan menjalankan *method* `showResult()` pada *class* Login.



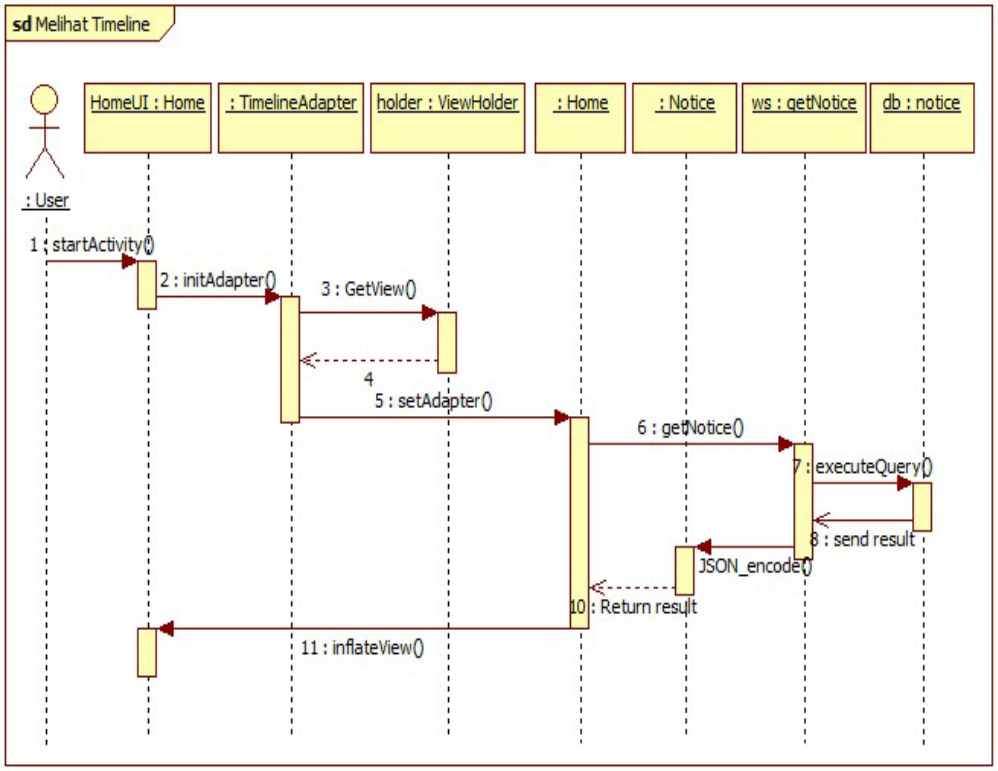
Gambar 4.10 *Sequence diagram* proses *login*

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk melihat *timeline*. Gambar 4.11 merupakan diagram sekuensial untuk melihat *timeline*. Diagram sekuensial ini menggambarkan interaksi ketika *user* melihat *timeline* di dalam menu *home*. Proses melihat *timeline* dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* mengakses menu *Home* dengan *method* `startActivity()`.
2. Sistem akan memilih tipe *adapter* yang akan digunakan untuk menampung data dari *database* dengan menjalankan *method* `initAdapter()` kemudian memanggil *class* TimelineAdapter
3. TimelineAdapter mengambil *view* (tampilan) yang dibutuhkan dari *class* ViewHolder dengan menjalankan *method* `GetView()`.

4. Data *posting* dimasukkan ke dalam *adapter* dengan menjalankan *method* `setAdapter()`.
5. *Method* `setAdapter()` mengambil data dari *database* dengan menjalankan *method* `getNotice()`, *method* ini mengakses *web service* untuk mendapatkan nilai kembalian berupa kumpulan (*list*) dari *posting* yang dilakukan *user* (*notice*).
6. Setelah data diperoleh, *class* Home menjalankan *method* `inflateView()` untuk menampilkan antarmuka dari *posting* dalam bentuk *timeline*.

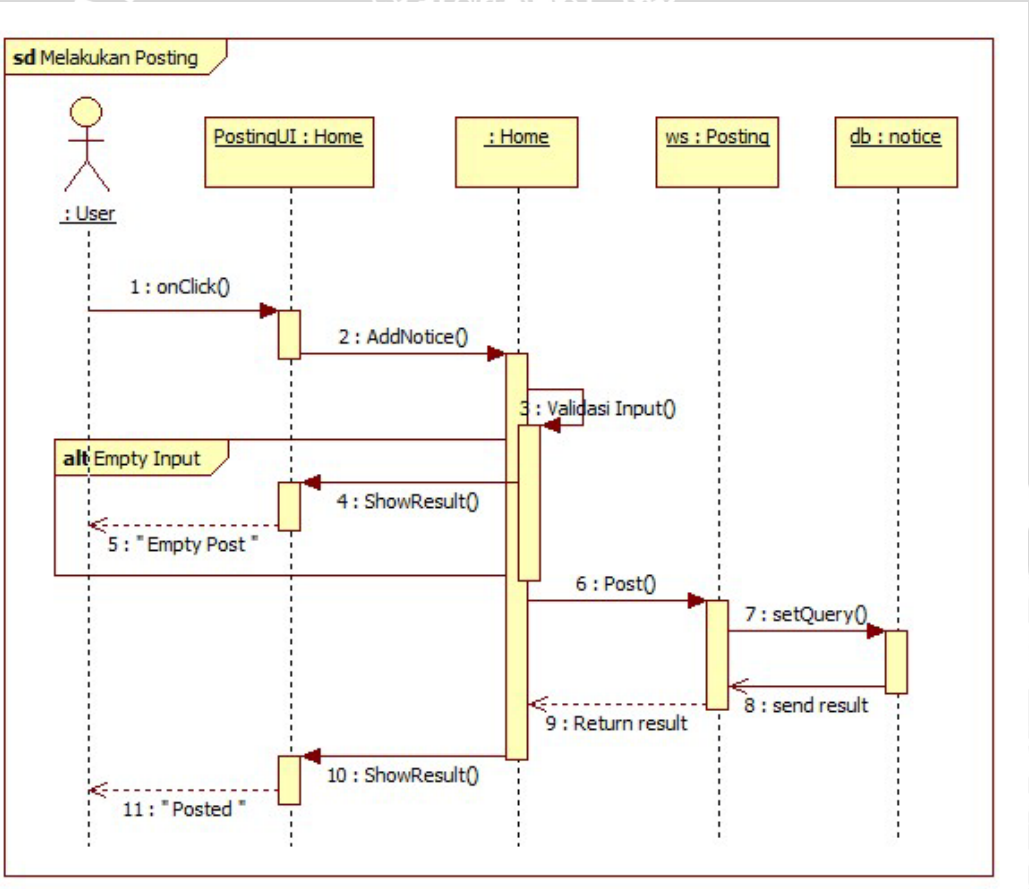


Gambar 4.11 *Sequence diagram* Melihat *Timeline*

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk melakukan *posting*. Gambar 4.12 merupakan diagram sekuensial untuk melakukan *posting*. Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan *posting*. Proses melakukan *posting* dijelaskan dengan deskripsi sebagai berikut ini :

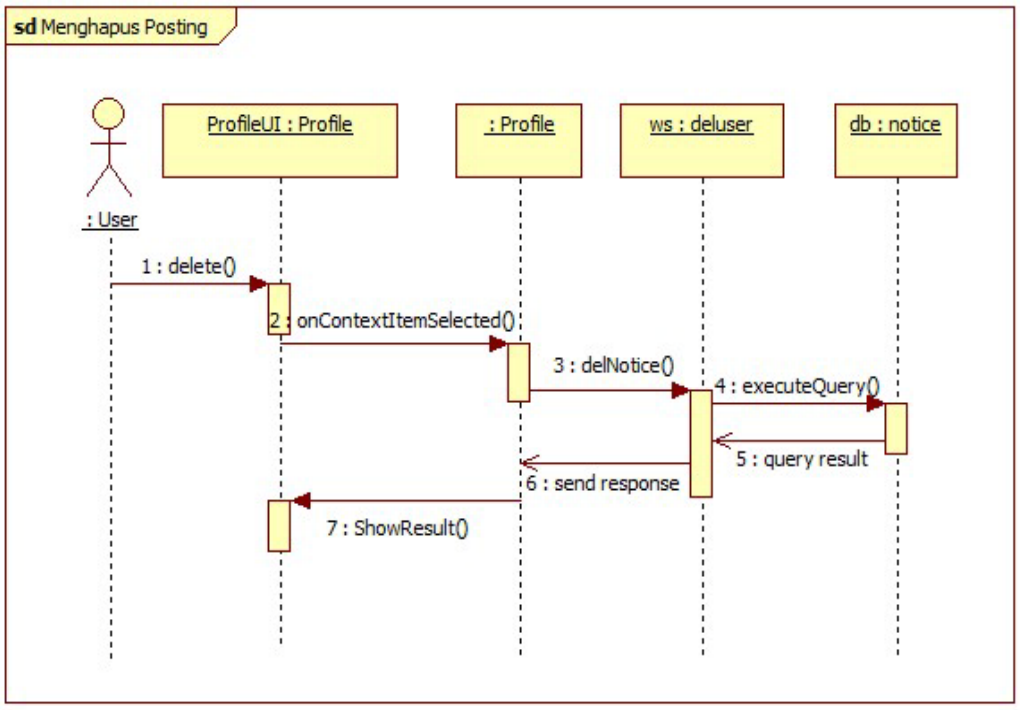
1. *User* melakukan *posting* dengan memasukan *input posting* pada menu *Posting* dan mengaktifkan *method* `onClick()` ketika menekan tombol *Post.*
2. *Method* `onClick()` pada *class* Home kemudian menjalankan *method* `AddNotice()`.
3. Sistem melakukan validasi *input posting*, Jika *input*-nya kosong maka sistem akan memberikan *alert* kepada *user* dengan menjalankan *method* `ShowResult()`.
4. Sistem menyimpan data input ke *database* dengan menjalankan *method* `Post()` untuk mengakses *web service.*
5. Respon dari *web service* ditampilkan dalam bentuk notifikasi dengan menjalankan *method* `ShowResult()`.



Gambar 4.12 *Sequence diagram* Melakukan *Posting*

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk menghapus *posting*. Diagram sekuensial ini menggambarkan interaksi ketika *user* menghapus *posting* di dalam menu *profile*. Proses menghapus *posting* dimulai ketika *user* memilih *posting* yang akan dihapus dari *timeline* kemudian mengaktifkan *method* `delete()`. Sistem menangkap *event* dari *list* yang dipilih oleh *user* dengan *method* `onContextItemSelected()`. Sistem kemudian menghapus data *posting* tersebut dari *database* dengan menjalankan *method* `delNotice()` pada *class* Profile. *Method* `delNotice()` mengakses *database* melalui *web service* deluser. Respon dari *web service* kemudian dikirimkan ke *user* dalam bentuk notifikasi bahwa data *posting* telah di hapus. Gambar 4.13 merupakan diagram sekuensial untuk menghapus *posting*.

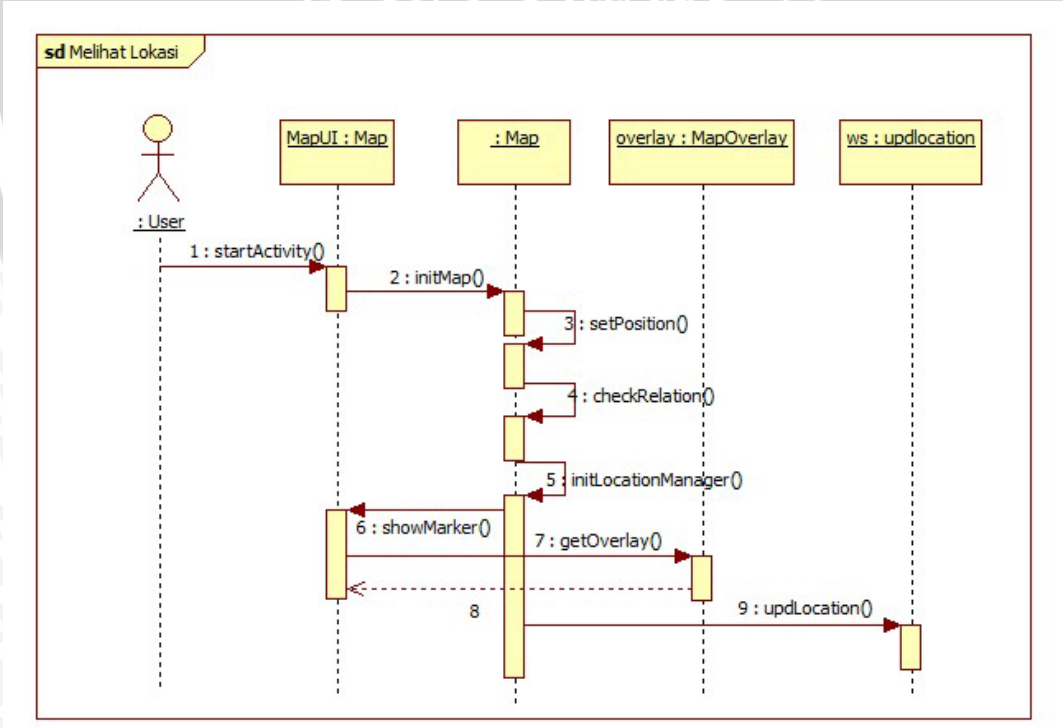


Gambar 4.13 *Sequence diagram* Menghapus *Posting*

Sumber : [Perancangan]

Gambar 4.14 merupakan diagram sekuensial pada proses melihat lokasi dari *user* lain. Diagram sekuensial ini menggambarkan interaksi ketika *user* melihat lokasi dari *user* lain pada menu *map*. Proses melihat lokasi dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* masuk ke dalam halaman *map* dari *user* lain dengan mengaktifkan *method* `startActivity()`.
2. Sistem kemudian menjalankan *method* `initMap()` untuk menampilkan peta di layar kemudian memanggil *method* `setPosition()`.
3. *Method* `setPosition()` berfungsi untuk inisialisasi koordinat dari *user* lain.
4. Sistemkemudian akan melakukan pengaturan terhadap GPS *receiver* dengan menjalankan *method* `initLocationManager()`.
5. Dalam *method* `initLocationManager()` terdapat *method* `showMarker()` yang berfungsi untuk menampilkan informasi lokasi dari *user* yang sesuai dalam peta.
6. *Method* `showMarker()` menjalankan *method* `getOverlay()` untuk mengambil data *marker* dan informasi dari *user*.
7. Jika terjadi perubahan koordinat Lokasi, maka sistem akan menjalankan *method* `updlocation()` untuk meng-*update* data yang ada di *database*.

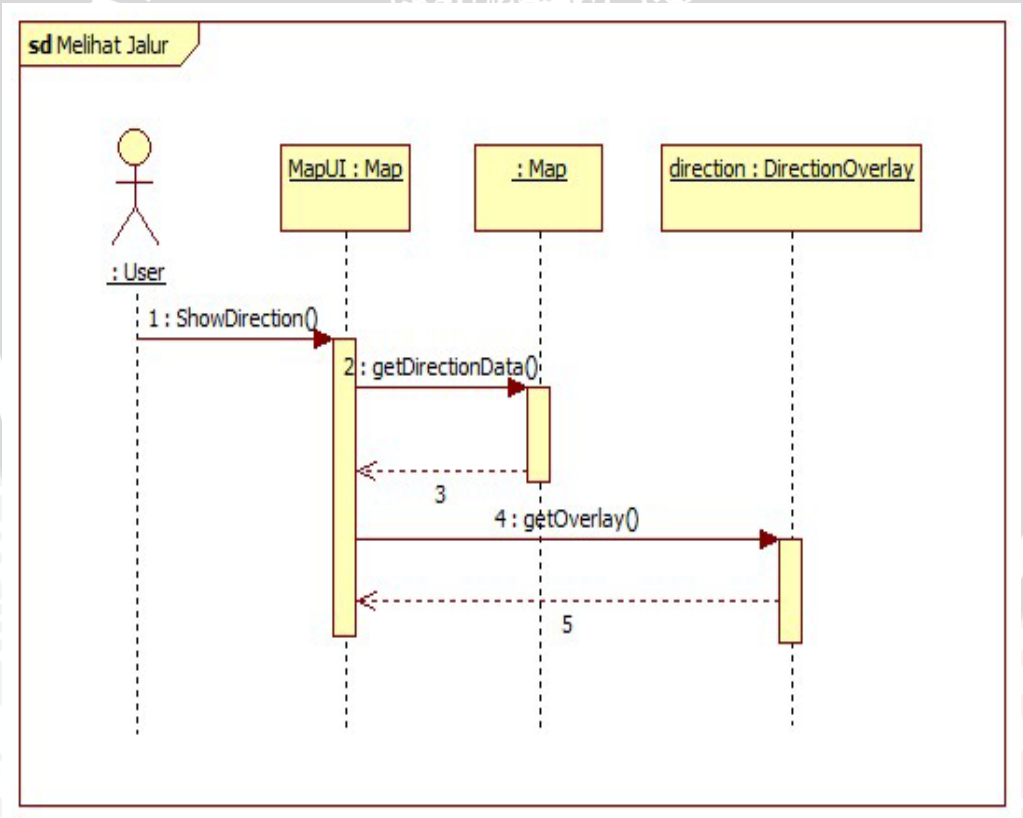


Gambar 4.14 *Sequence diagram* Melihat Lokasi

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk melihat jalur yang dapat dilalui menuju lokasi *user* lain. Gambar 4.15 merupakan diagram sekuensial untuk melihat jalur. Diagram sekuensial ini menggambarkan interaksi ketika *user* menggunakan fasilitas lihat jalur. Proses melihat jalur dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* melihat jalur yang dilalui dengan menekan tombol jalur untuk mengaktifkan *method* `showDirection()`.
2. *Method* `showDirection()` menjalankan *method* `getDirectionData()` untuk memperoleh koordinat jalur yang akan dilalui.
3. *Method* `showDirection()` menjalankan *method* `getOverlay()` untuk memperoleh menampilkan jalur yang dilalui ke dalam peta.

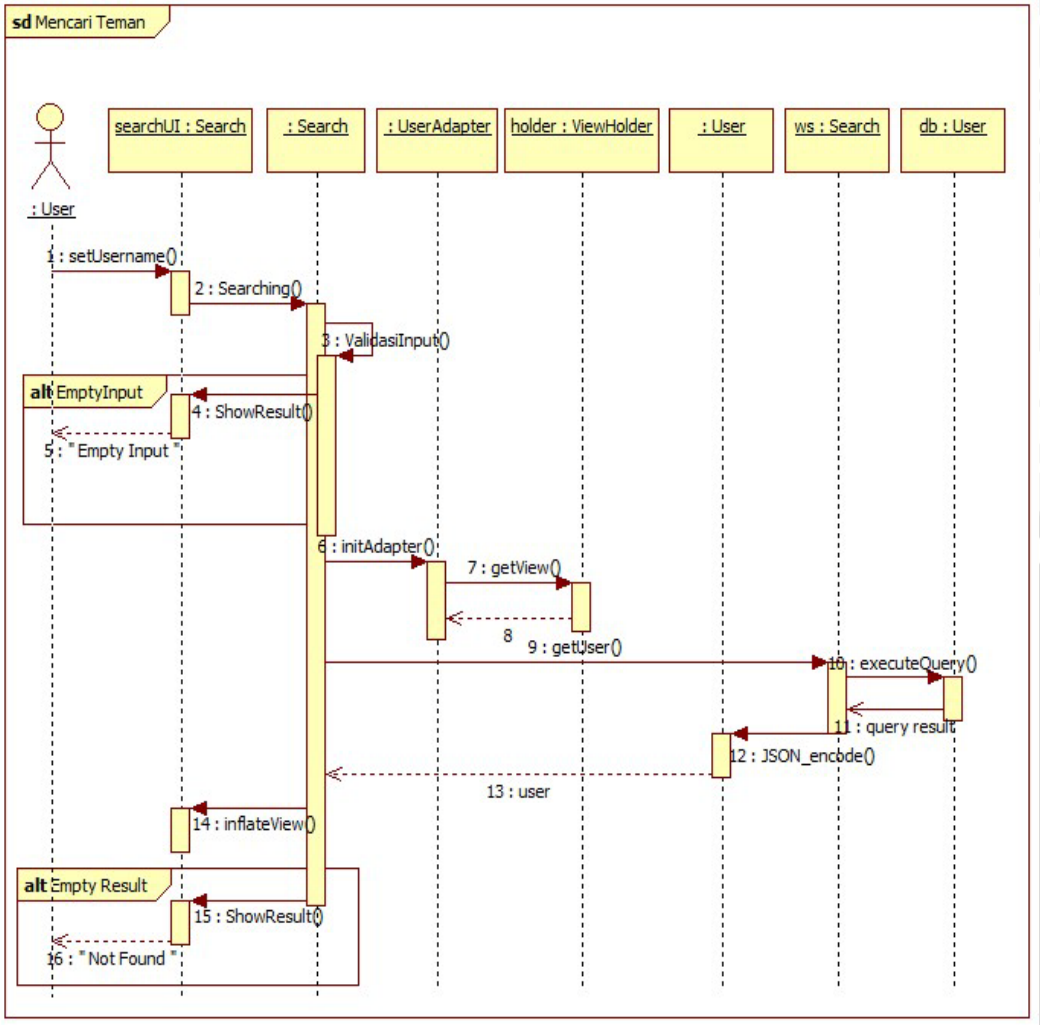


Gambar 4.15 *Sequence diagram* Melihat Jalur

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk mencari teman. Gambar 4.16 merupakan diagram sekuensial yang menggambarkan interaksi ketika *user* melakukan proses pencarian *user* berdasarkan *username*. Proses mencari teman dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* melakukan pencarian dengan memasukan *input username* pada menu *search* kemudian sistem akan menjalankan *method* `searching()`.
2. Sistem melakukan validasi *input username*, Jika *input*-nya kosong maka sistem akan memberikan *alert* kepada *user* dengan menjalankan *method* `showResult()`.
3. Sistem akan memilih tipe *adapter* yang akan digunakan untuk menampung data dari *database* dengan menjalankan *method* `initAdapter()`.
4. Sistem kemudian mengambil data *user* dari *database* menggunakan *method* `getUser()`.
5. *Method* `getUser()` berfungsi untuk mengakses *database* melalui *web service*.
6. Data hasil *query database* di-*encode* menjadi format JSON menggunakan *method* `JSON_encode()` kemudian hasilnya dikirim ke *class* Search.
7. Sistem kemudian menampilkan data ke antarmuka dengan *method* `inflateView()`.
8. Jika data *user* yang dicari tidak ada, maka sistem meberikan notifikasi hasil kepada *user* bahwa *user* yang dicari tidak ditemukan.



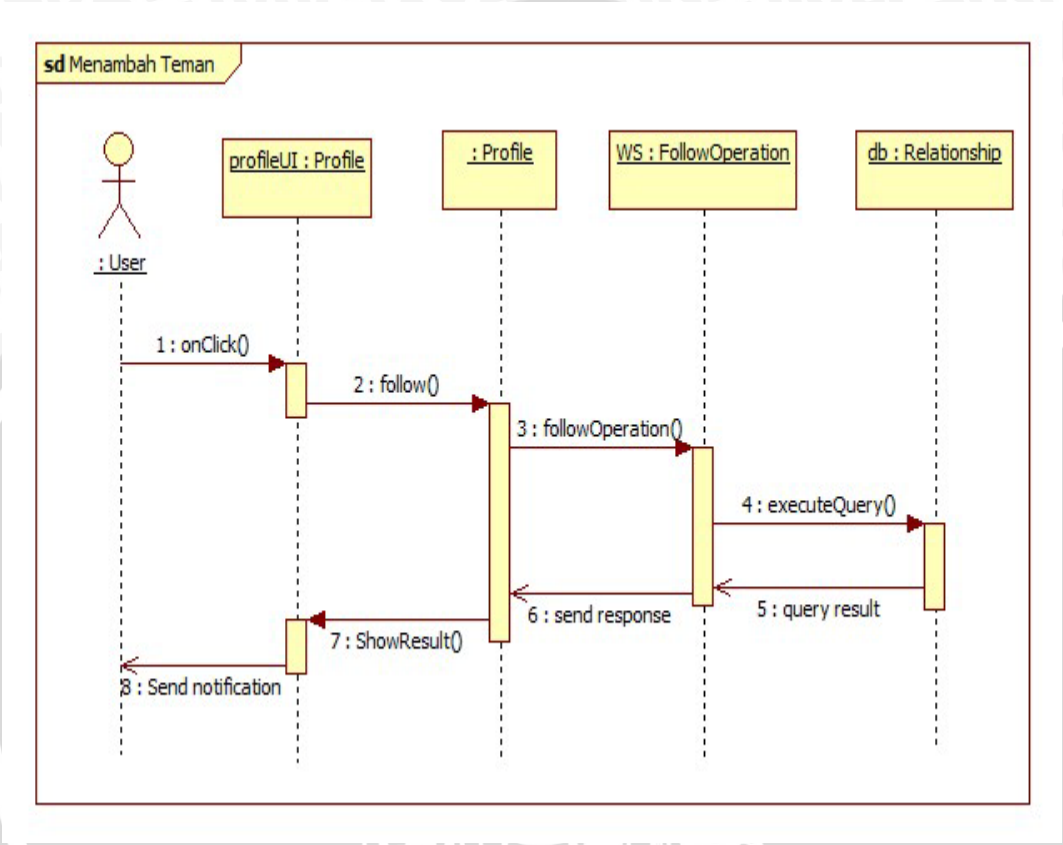
Gambar 4.16 *Sequence diagram* Mencari Teman

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk menambah teman. Gambar 4.17 merupakan diagram sekuensial untuk menambah teman. Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan proses menambah teman. Proses ini dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* menambah teman dengan menekan tombol "+" pada menu *profile* untuk mengaktifkan *method* `onClick()`.
2. Sistem menambah data pertemanan ke *database* dengan mengaktifkan *method* `followOperation()`.

3. *Method* `followOperation()` kemudian mengakses *database* melalui *web service* FollowOperation.
4. Respon dari *web service* kemudian ditampilkan dengan *method* `showResult()`.



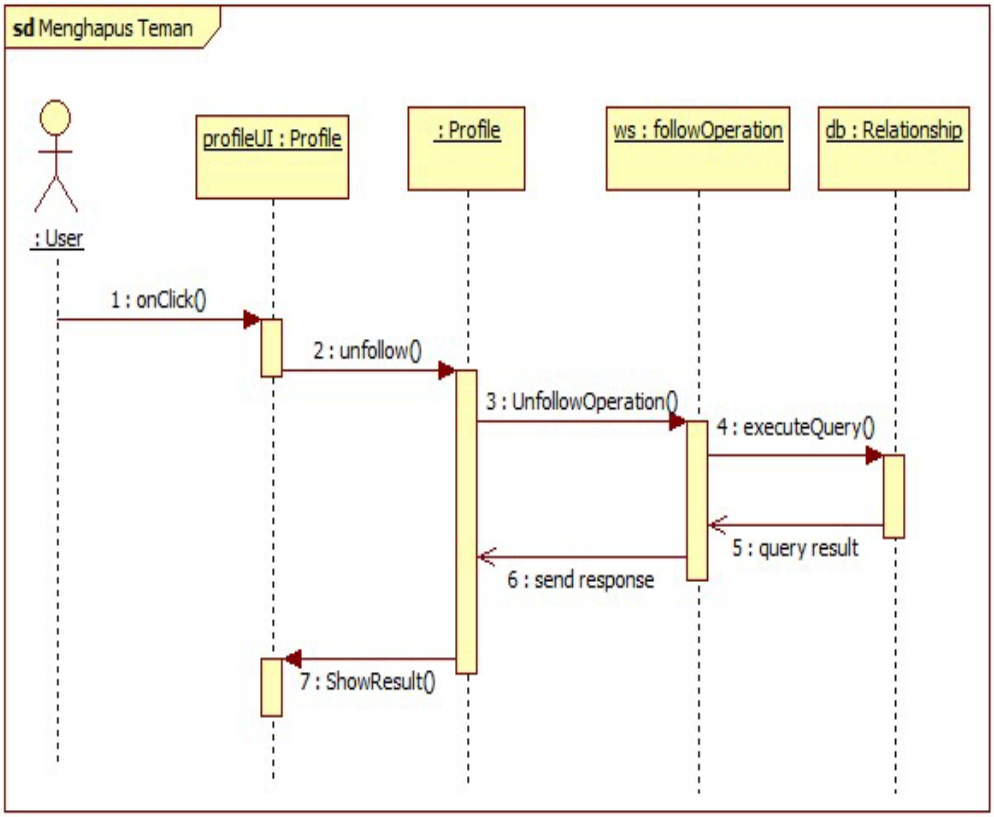
Gambar 4.17 *Sequence diagram* Menambah Teman

Sumber : [Perancangan]

Sequence diagram selanjutnya adalah *sequence diagram* untuk menghapus teman. Gambar 4.18 merupakan diagram sekuensial untuk menghapus teman. Diagram sekuensial ini menggambarkan interaksi ketika *user* melakukan proses menghapus teman. Proses ini dijelaskan dengan deskripsi sebagai berikut ini :

1. *User* menghapus teman dengan memasukan menekan tombol "-" pada menu *profile* untuk mengaktifkan *method* `onClick()`.
2. Sistem menghapus data pertemanan dari *database* dengan mengaktifkan *method* `unfollowOperation()`.

3. *Method* `unfollowOperation()` kemudian mengakses *database* melalui *web service* FollowOperation.
4. Respon dari *web service* kemudian ditampilkan dengan *method* `showResult()`.
5. Sistem kemudian menampilkan notifikasi bahwa poses hapus teman berhasil.

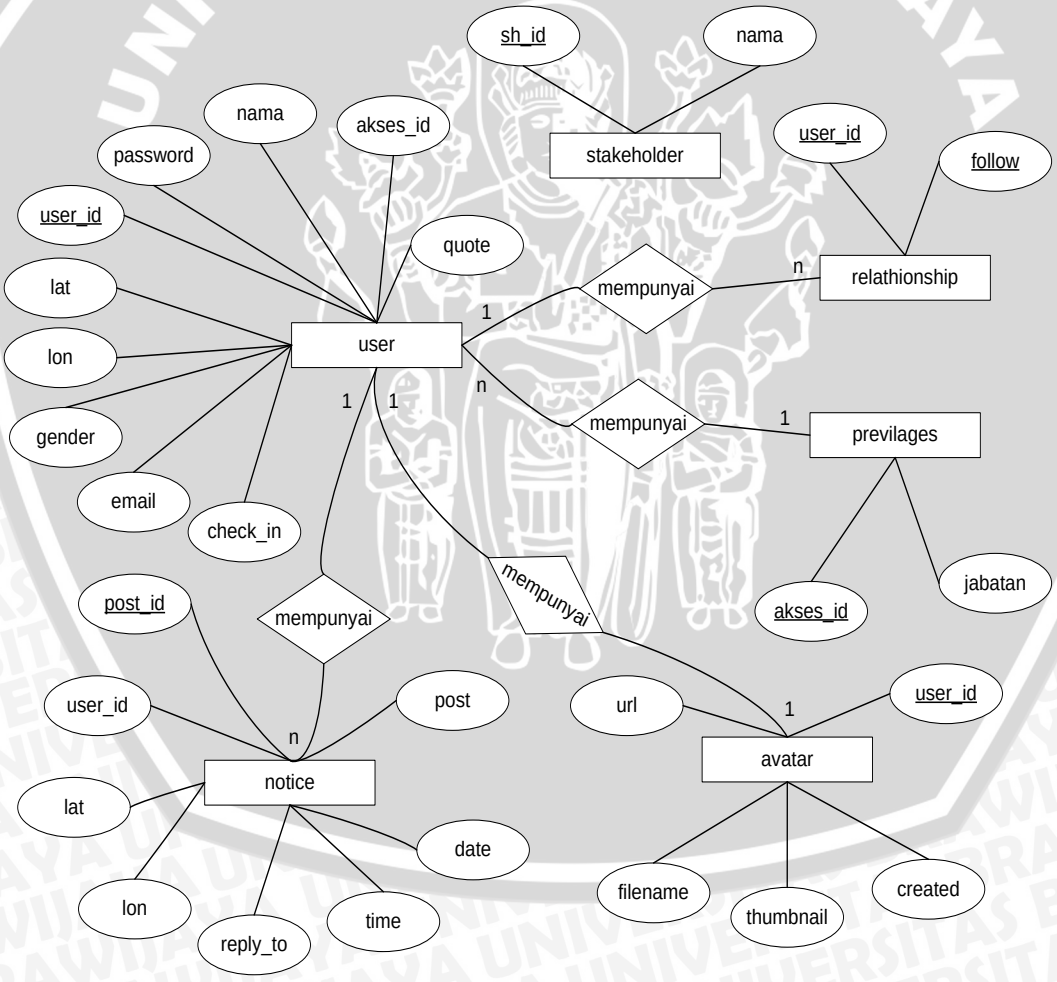


Gambar 4.18 *Sequence diagram* Menghapus Teman

Sumber : [Perancangan]

4.3.3 Perancangan Basis Data

Basis data berfungsi sebagai tempat menyimpan data. Pada skripsi ini perancangan basis data direpresentasikan dalam bentuk *Entity Relationship Diagram* (ERD). ERD menunjukkan hubungan yang terjadi diantara objek (entitas) yang terlibat dalam suatu *database*. ERD berisi komponen-komponen himpunan entitas dan himpunan relasi yang masing-masing dilengkapi dengan beberapa atribut yang mempresentasikan seluruh fakta yang ditinjau dari keadaan yang nyata. Pada perancangan basis data sistem ini terdapat enam buah tabel yaitu tabel *user*, tabel *notice*, tabel *previlages*, tabel *relationship*, tabel *avatar*, dan tabel *stakeholder*. ERD sistem ini dapat dilihat pada Gambar 4.19.



Gambar 4.19 ERD aplikasi jejaring sosial kampus

Sumber : [Perancangan]

Berikut ini merupakan struktur tabel serta keterangan masing masing tabel dan *field* yang ada pada *database*. Entitas *user* merepresentasikan tabel *user* yang berisi data-data dari pengguna aplikasi jejaring sosial seperti *userid*, nama, *password*, aksesid, *gender*, koordinat lintang, koordinat bujur dan *email*. Dalam pengembangannya nanti, atribut dari *user* dapat ditambahkan sesuai dengan kebutuhan. Struktur tabel *user* ditunjukkan pada Tabel 4.21.

Tabel 4.21 Struktur tabel *user*

No	Nama Field	Tipe	Lebar	Keterangan
1	user_id (PK)	Varchar	30	Kode dari *user.*
2	*password*	Varchar	50	*Password* dari *user.*
3	nama	Varchar	50	Nama dari *user.*
4	akses_id (FK)	Integer	2	Kode dari hak ases *user.*
5	*gender*	Varchar	6	Jenis kelamin.
6	lat	Varchar	30	Koordinat lintang.
7	lon	Varchar	30	Koordinat bujur.
9	*quote*	Text	-	Testimoni terakhir.
10	*email*	Varchar	50	*Email user.*
11	*check_in*	DateTime	-	Waktu pada lokasi terakhir.

Sumber : [Perancangan]

Entitas *previlages* merepresentasikan tabel *previlages* di dalam *database*. Tabel *previlages* berisi tipe akses yang dapat dimiliki pengguna aplikasi jejaring sosial. Ada tiga tipe hak akses yang dapat dimiliki pengguna yaitu sebagai admin, mahasiswa dan dosen. Struktur tabel *previlages* ditunjukkan pada Tabel 4.22.

Tabel 4.22 Struktur tabel *previlages*

No	Nama Field	Tipe	Lebar	Keterangan
1	akses_id (PK)	Integer	2	Kode dari hak ases *user.*
2	jabatan	Varchar	30	Hak akses ke sistem.

Sumber : [Perancangan]

Entitas *avatar* merepresentasikan tabel *avatar* di dalam *database.* Tabel *avatar* berisi data *gambar profile* dari jejaring sosial. Data tersebut meliputi *userid* pemilik gambar, nama *file*, alamat *url* dari *file* gambar, dan waktu gambar dibuat. Struktur tabel *notice* ditunjukkan pada Tabel 4.23.

Tabel 4.23 Struktur tabel *avatar*

No	Nama Field	Tipe	Lebar	Keterangan
1	user_id (PK,FK)	Varchar	30	Kode dari *user*.
2	*filename*	Varchar	50	Nama dari *file* gambar.
3	*url*	Varchar	50	Alamat dari *file* gambar.
4	*thumbnail*	Varchar	50	Alamat dari *thumbnail*.
5	*created*	DateTime	-	Tanggal *posting*.

Sumber : [Perancangan]

Entitas *notice* merepresentasikan tabel *notice* di dalam *database.* Tabel *notice* merupakan tabel utama yang berisi data *posting* dari jejaring sosial seperti *postid*, *userid*, waktu *posting*, koordinat lintang, koordinat bujur, dan kode balasan pesan. Struktur tabel *notice* ditunjukkan pada Tabel 4.24.

Tabel 4.24 Struktur tabel *notice*

No	Nama Field	Tipe	Lebar	Keterangan
1	post_id (PK)	Integer	11	kode dari *posting*.
2	user_id (FK)	Varchar	30	kode dari *user*.
3	*post*	Text	-	Isi *posting*.
4	*date*	Date	-	Tanggal *posting*.
5	*time*	Varchar	5	Waktu *posting*.
6	lat	Varchar	30	Koordinat lintang.
7	lon	Varchar	30	Koordinat bujur.
8	*reply_to*	Integer	11	Kode *posting* yang dibalas.

Sumber : [Perancangan]

Entitas *relationship* merepresentasikan tabel *relationship* di dalam *database*. Tabel *relationship* berisi daftar teman yang dimiliki oleh pengguna aplikasi jejaring sosial. Struktur tabel *relationship* ditunjukkan pada Tabel 4.25.

Tabel 4.25 Struktur tabel *relationship*

No	Nama Field	Tipe	Lebar	Keterangan
1	user_id (PK,FK)	Varchar	30	Kode dari *user*.
2	*follow* (PK)	Varchar	30	Kode *user* sebagai teman.

Sumber : [Perancangan]

Entitas *stakeholder* merepresentasikan tabel *stakeholder* di dalam *database*. Tabel *stakeholder* berisi daftar nama calon pengguna yang dapat mendaftar sebagai pengguna aplikasi jejaring sosial. Struktur tabel *stakeholder* ditunjukkan pada Tabel 4.26.

Tabel 4.26 Struktur tabel *stakeholder*

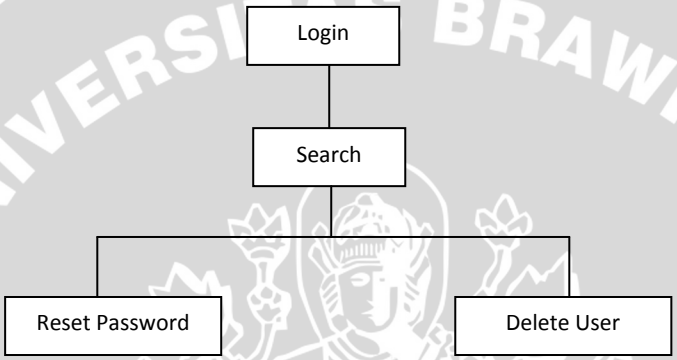
No	Nama Field	Tipe	Lebar	Keterangan
1	sh_id (PK)	Varchar	30	Nomor induk *stakeholder*.
2	nama	Varchar	50	Nama *stakeholder*.

Sumber : [Perancangan]

4.3.4 Perancangan Antarmuka

Layout aplikasi merupakan gambaran dari struktur program yang dijalankan. Perancangan antarmuka berisi rancangan tampilan aplikasi yang akan digunakan oleh *user*. Perancangan antarmuka sangat diperlukan untuk mempermudah *user* dalam menggunakan aplikasi. *Sitemap* digunakan untuk merepresentasikan keseluruhan antarmuka sistem berdasarkan otoritas pengguna. Aplikasi ini mempunyai dua menu otoritas yaitu menu untuk admin dan menu untuk *user*.

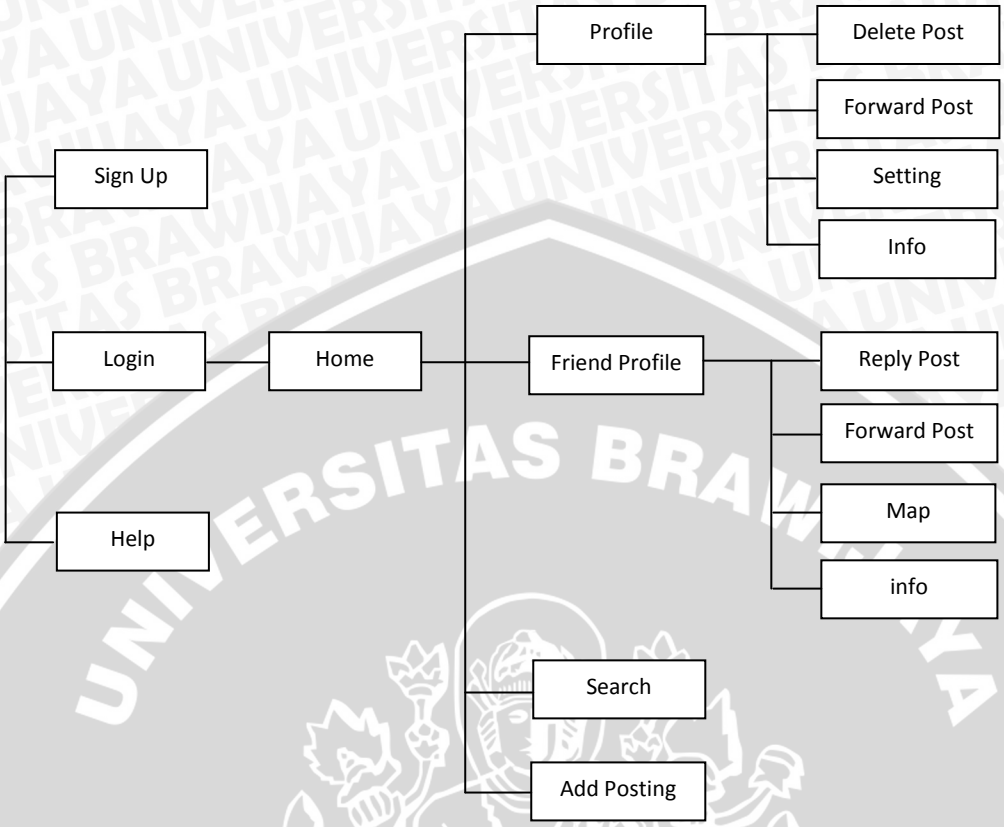
Menu admin merupakan menu khusus yang disediakan oleh aplikasi untuk admin. *Form login* adalah antarmuka pertama yang akan ditampilkan oleh aplikasi. Jika berhasil *login* sebagai admin, maka antarmuka khusus admin yang disediakan oleh aplikasi berupa menu pencarian *user* (*search*) yang berisi *listview* dari daftar *user* yang dicari. Menu ini memiliki dua *submenu* yaitu *reset password* dan *delete user*. *Submenu* ini dapat diakses dengan memilih salah satu *user* dari *listview user*. Gambar 4.20 menunjukkan *site map* dari menu admin.



Gambar 4.20 *Site map* menu untuk admin

Sumber : [Perancangan]

Site map selanjutnya adalah *site map* untuk menu khusus *user*. Menu khusus *user* merupakan antarmuka yang disediakan oleh aplikasi untuk *user*. *Form login* adalah antarmuka pertama yang akan ditampilkan oleh aplikasi. Jika otoritasnya sebagai *user*, maka aplikasi akan menyediakan *form home* yang berupa *timeline* dari *posting* yang dilakukan *user*. Pada *home* terdapat empat menu yaitu menu *profile*, *friend profile*, *search*, dan *add posting*. Menu *profile* menyediakan antarmuka bagi *user* untuk mengelola *account* dan *posting* miliknya. Menu *friend profile* merupakan antarmuka yang menampilkan informasi dari *user* lain. Pada menu *friend profile* terdapat *submenu map* yang berfungsi untuk menampilkan lokasi dari seorang *user* dalam bentuk peta. *Site map* dari menu *user* ditunjukan pada Gambar 4.21.

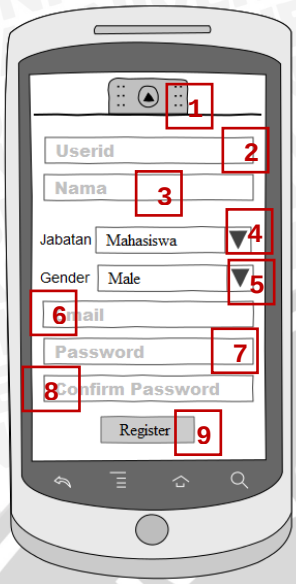


Gambar 4.21 *Site map* menu untuk *user*

Sumber : [Perancangan]

4.3.4.1 Tampilan *Sign Up*

Pada menu *sign up*, *user* yang belum terdaftar di dalam *database* aplikasi dapat melakukan registrasi. *User* memulai registrasi dengan memasukkan *userid*, *username*, jenis kelamin, jabatan, dan *password*. *Userid* adalah sederet kode yang merupakan NIM atau NIP dari *user*. Jika *userid* tidak valid maka *user* tersebut tidak bisa melakukan registrasi. Menu ini merupakan representasi dari kebutuhan fungsional dengan kode SRS-001-01. Rancangan antarmuka untuk menu *sign up* dapat dilihat pada Gambar 4.22.



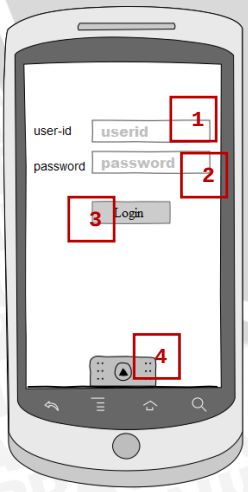
Keterangan :

1. *SlidingDrawer* untuk mengakses menu *Sign Up*.
2. *Field* untuk mengisi *userid*.
3. *Field* untuk mengisi nama atau *username*.
4. *Combo Box* pilihan jabatan yang berisi mahasiswa atau dosen.
5. *Combo Box* pilihan *gender* (jenis kelamin).
6. *Field* untuk mengisi *email*.
7. *Field* untuk mengisi *password*.
8. *Field* untuk mengisi konfirmasi *password*.
9. Tombol *register* untuk menambahkan *user* baru ke *database*.

Gambar 4.22 Tampilan menu *sign up*
Sumber : [Perancangan]

4.3.4.2 Tampilan *Login*

Pada menu *login*, *admin* ataupun *user* dapat menggunakan otoritasnya untuk masuk sekaligus membuka aplikasi jejaring sosial. Aktor memulainya dengan memasukkan *userid* dan *password*. Rancangan antarmuka untuk menu *login* (SRS-001-02) dapat dilihat pada Gambar 4.23.



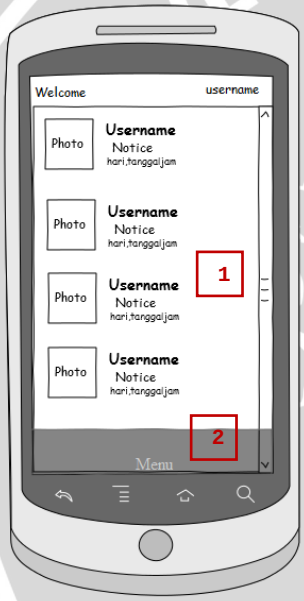
Keterangan :

1. *Field* untuk mengisi *userid*.
2. *Field* untuk mengisi *password*.
3. Tombol *Login* untuk masuk ke sistem.
4. *SlidingDrawer* untuk mengakses menu *Sign Up*.

Gambar 4.23 Tampilan menu *login*
Sumber : [Perancangan]

4.3.4.3 Tampilan *Home*

Home merupakan halaman pertama yang akan disediakan oleh aplikasi setelah *user* berhasil melakukan proses *login*. Home berisi *timeline* dari *posting* yang dilakukan oleh *user* di dalam aplikasi jejaring sosial yang merupakan representasi dari kebutuhan fungsional dengan kode SRS-002-02. *Submenu* yang terdapat pada halaman *home* antara lain adalah *add posting*, *search*, menu *profile*, n menu *friend profile*. Menu *profile* dan menu *friend profile* dapat diakses dengan cara memilih salah satu *user* yang ada pada *timeline*. Tampilan untuk menu *Home* ditunjukan pada Gambar 4.24.



Keterangan :

1. *Timeline* yang berisi kumpulan *posting* (*notice*) dari *user* yang tersimpan pada *database*.
2. *Option Menu* yang berisi menu *add*, *search*, *refresh* dan *logout*.

Gambar 4.24 Tampilan menu *home*

Sumber : [Perancangan]

4.3.4.4 Tampilan Menu *Add Posting*

Pada halaman ini *user* dapat melakukan *posting* ke dalam aplikasi. *User* dapat melakukan input *posting* pada *text field* yang disediakan untuk *posting*. Data *posting* dikirimkan ke *server* beserta dengan koordinat lokasi dimana *posting* tersebut ditulis. Gambar 4.25 Menunjukan tampilan untuk menu *add posting* (SRS-002-01).

Keterangan :

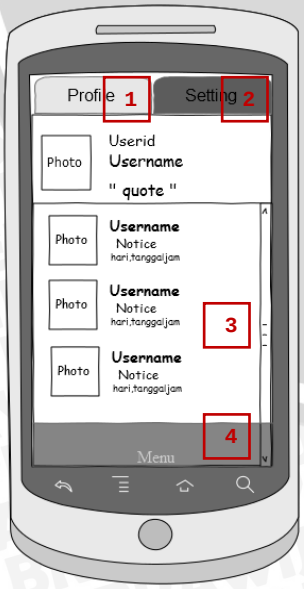
1. *Field* untuk mengisi data *posting* (*notice*).
2. Tombol *POST* untuk memasukan *posting* ke dalam *database*.
3. Tombol *Cancel* untuk membatalkan *posting* dan kembali ke menu sebelumnya.

Gambar 4.25 Tampilan menu *add posting*

Sumber : [Perancangan]

4.3.4.5 Tampilan Menu *Profile*

Pada halaman ini, *user* dapat melihat *timeline* dari *posting* miliknya (SRS-002-02). *Timeline* pada halaman ini juga menyediakan fasilitas bagi *user* untuk menghapus *posting* yang dilakukan (SRS-002-03). Tampilan untuk menu *profile* ditunjukan pada Gambar 4.26.



Keterangan :

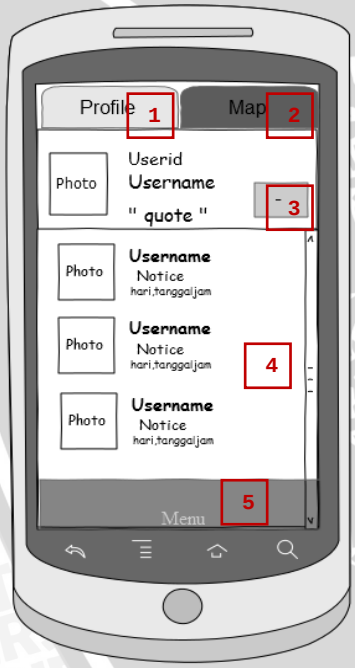
1. *Tab Menu profile* dalam keadaan aktif.
2. *Tab Menu Setting* yang berisi *form* untuk melakukan *seting account.*
3. *ListView* yang berisi *posting* (*notice*).
4. *Option Menu* yang berisi menu *home*, *detail*, *conversation* dan *logout*.

Gambar 4.26 Tampilan menu *profile*

Sumber : [Perancangan]

4.3.4.6 Tampilan Menu *Friend Profile*

Pada halaman ini *user* dapat melihat *timeline* dan info dari *user* lain (SRS-002-02 dan SRS-004-02). Jika *user* lain tersebut belum menjadi teman, maka seorang *user* hanya bisa melihat info dari *user* lain tersebut. Pada halaman ini *user* dapat menambah *user* lain sebagai teman dengan menekan tombol "+" (SRS-004-03) atau menghapus teman dengan menekan tombol "-" (SRS-004-04) sebagaimana telah dijelaskan pada skenario *use case* sebelumnya. *User* juga dapat membalas (*reply*) maupun meneruskan (*forward*) *posting* yang dilakukan oleh teman dengan cara memilih *posting* dari *timeline* kemudian memilih *submenu reply* atau *forward*. Rancangan antarmuka halaman *friend profile* dapat dilihat pada Gambar 4.27.



Keterangan :

1. *Tab Menu friend profile* dalam keadaan aktif.
2. *Tab Menu Map* yang berisi *form* untuk melihat lokasi *user*.
3. *Tombol* untuk menambah atau menghapus teman.
4. *ListView* yang berisi *posting* (*notice*).
5. *Option Menu* yang berisi menu *home*, *detail*, dan *logout*.

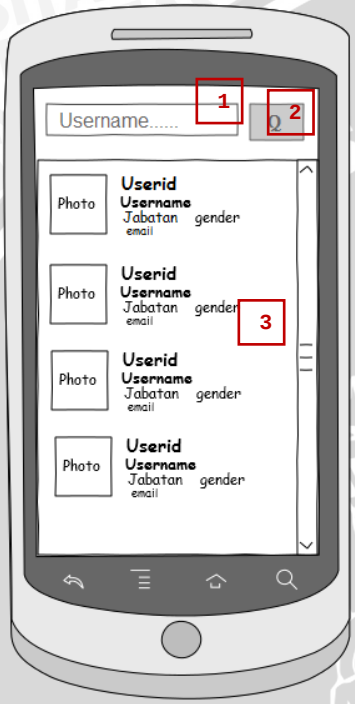
Gambar 4.27 Tampilan menu *friend profile*

Sumber : [Perancangan]

4.3.4.7 Tampilan Menu *Search*

Search merupakan halaman yang disediakan oleh aplikasi untuk mencari *user* berdasarkan parameter tertentu. Halaman *search* untuk admin menggunakan

userid sebagai parameter pencarian (SRS-006-02) sedangkan pada halaman *search* untuk *user* menggunakan *username* (SRS-004-01). Halaman ini berisi daftar *user* aplikasi jejaring sosial yang berhasil dicari sesuai dengan parameter yang dimasukkan. Rancangan antarmuka untuk halaman *search* dapat dilihat pada Gambar 4.28.



Keterangan :

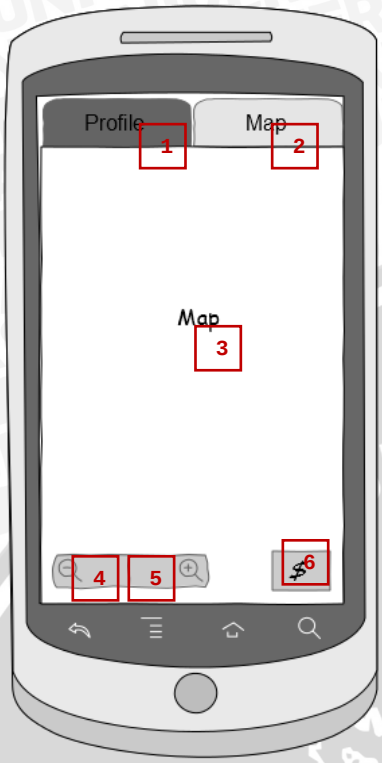
1. *Field* untuk mengisi parameter pencarian *username* (jika sebagai *user*) atau *userid* (jika sebagai admin).
2. Tombol *search* untuk melakukan pencarian *user*.
3. *ListView* yang berisi daftar *user* yang sesuai dengan parameter pencarian.

Gambar 4.28 Tampilan menu *search*

Sumber : [Perancangan]

4.3.4.8 Tampilan Menu *Map*

User dapat melihat lokasi *user* lain dengan menggunakan menu *map* (SRS-003-01). Menu ini dapat diakses setelah *user* memilih menu *profile* dari *user* lain. Halaman ini berisi peta yang akan menunjukan posisi *user* lain. Posisi akan ter-*update* secara otomatis jika terjadi perubahan koordinat dari seorang *user* (*auto geotagging*). Seorang *user* juga dapat melihat jalur mana yang harus dilewati untuk menuju lokasi dari *user* lain dengan menekan tombol bergambar jalur sebagaimana telah dijelaskan pada skenario *use case* sebelumnya (SRS-003-02). Tampilan untuk menu *map* ditunjukan pada Gambar 4.29.



Keterangan :

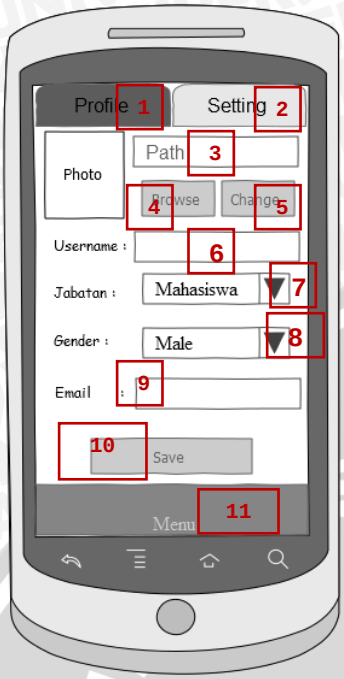
1. *Tab Menu friend profile* yang berisi halaman *profile* teman.
2. *Tab Menu Map* dalam keadaan aktif.
3. *Map View* untuk menampilkan peta berbasis *google map*.
4. Tombol *zoom-out* untuk memperkecil tampilan peta.
5. Tombol *zoom-in* untuk memperbesar tampilan peta.
6. Tombol jalur untuk menampilkan jalur yang dapat dilalui menuju *user* lain.

Gambar 4.29 Tampilan menu *map*

Sumber : [Perancangan]

4.3.4.9 Tampilan Menu *Setting*

User dapat meng-*edit* data pribadinya dengan menggunakan menu *seting*. Menu *setting* berisi fasilitas bagi seorang *user* untuk mengganti gambar *profile*, mengganti biodata, dan mengganti *password* (SRS-005-01). Untuk mengganti gambar profil, *user* dapat menekan tombol *browse* untuk memilih *file* gambar dari media penyimpanan *smartphone* kemudian menekan tombol *change*. *User* dapat mengganti *password* dengan menekan *option menu* dan memilih ganti *password*. Tampilan untuk menu *setting* ditunjukan pada Gambar 4.30.



Keterangan :

1. *Tab Menu Profile* yang berisi halaman *profile*.
2. *Tab Menu Setting* dalam keadaan aktif.
3. *Field* lokasi *file* yang akan secara otomatis terisi oleh hasil penelusuran.
4. Tombol *browse* untuk menelusuri lokasi *file* gambar.
5. Tombol *change* untuk mengganti gambar *profile*.
6. *Field* untuk mengisi *username*.
7. *Combo Box* pilihan jabatan yang berisi mahasiswa atau dosen.
8. *Combo Box* pilihan *gender* (jenis kelamin).
9. *Field* untuk mengisi *email*.
10. Tombol *save* untuk menyimpan perubahan data *user* ke *database*.
11. *Option Menu* untuk mengakses *edit password*.

Gambar 4.30 Tampilan menu *setting*

Sumber : [Perancangan]

BAB V
IMPLEMENTASI

Pada bab ini dibahas mengenai implementasi perangkat lunak berdasarkan hasil yang telah didapatkan dari analisis kebutuhan dan proses perancangan perangkat lunak. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, batasan-batasan dalam implementasi, implementasi tiap *class* pada *file* program, implementasi *database*, implementasi algoritma, dan implementasi antarmuka perangkat lunak.

5.1 Spesifikasi Sistem

Hasil dari analisis kebutuhan dan perancangan perangkat lunak yang telah dijelaskan pada Bab 4 menjadi acuan untuk melakukan implementasi menjadi sebuah aplikasi android yang dapat berfungsi sesuai dengan kebutuhan. Spesifikasi sistem diimplementasikan pada spesifikasi perangkat keras dan perangkat lunak.

5.1.1 Spesifikasi Perangkat Keras

Pengembangan aplikasi jejaring sosial pada *smartphone* android ini menggunakan sebuah komputer dengan spesifikasi prosesor, *memory*, *hardisk*, *mother board*, dan kartu grafis yang dijelaskan pada Tabel 5.1.

Tabel 5.1 Spesifikasi perangkat keras komputer

Nama Komponen	Spesifikasi
Prosesor	Intel® Core™ 2 Duo CPU P8400 @ 2.26*GHz*
Memory (RAM)	4 *GB*
Hardisk	Toshiba MK1652GSX, kapasitas 160 *GB*, 5400 *rpm*
Mother Board	Toshiba Satellite Pro U400
Kartu Grafis	Mobile Intel® GMA 4500MHD

Sumber : [Implementasi]

Dalam proses instalasi dan pengujian, perangkat yang digunakan adalah *smartphone* android dengan spesifikasi perangkat keras ditunjukan pada Tabel 5.2.

Tabel 5.2 Spesifikasi perangkat keras *smartphone*

Nama Komponen	Spesifikasi
Prosesor	1GHz Qualcom Snapdragon
Memory (RAM)	512 MB
Internal Storage	400 MB
External Storage	8 GB
Kartu Grafis (GPU)	Adreno 205 *dual chanel*

Sumber : [Implementasi]

5.1.2 Spesifikasi Perangkat Lunak

Pengembangan aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android ini menggunakan perangkat lunak (*tools*) dengan spesifikasi yang dijelaskan pada Tabel 5.3.

Tabel 5.3 Spesifikasi perangkat lunak komputer

Nama Komponen	Spesifikasi
Sistem operasi	Microsoft Windows® Vista Business *Service Pack* 2 (32 *bit*)
Bahasa pemrograman	Java dan XML
Tools pemrograman	JDK 1.6.0_23, Android SDK (*Software Development Kit*) revisi 15
Lingkungan pemrograman	Java™ 2 Runtime Environment, Standard Edition (build 1.6.0_23-ea-b01) , MySQL dan PHP
IDE (*Integrated Development Environment*)	Eclipse Helios dengan ADT (Android *Development Tools*) *plug in.*

Sumber : [Implementasi]

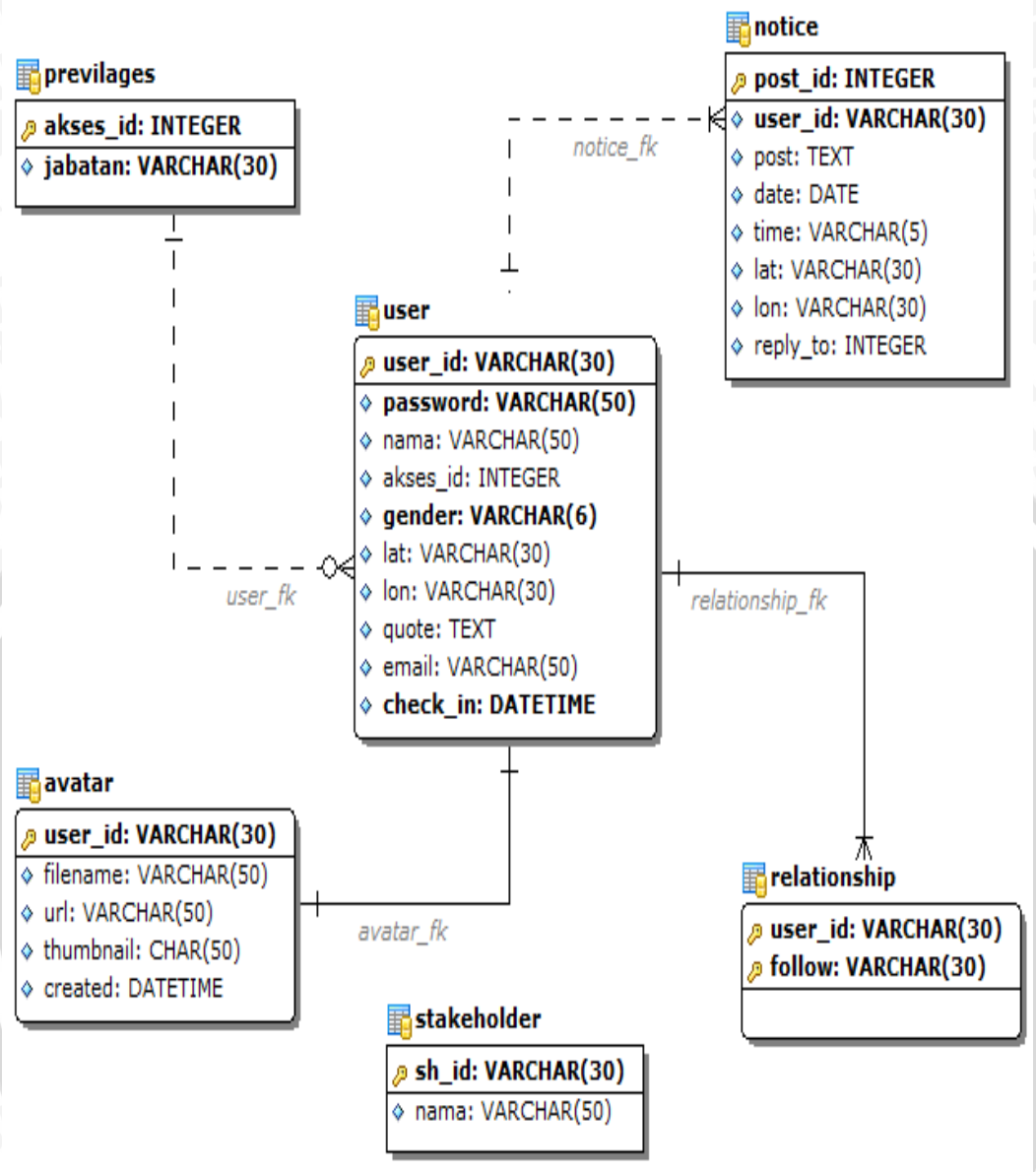
5.2 Batasan-Batasan Implementasi

Beberapa batasan dalam mengimplementasikan sistem adalah sebagai berikut:

1. Aplikasi jejaring sosial kampus berbasis GPS dirancang untuk dijalankan pada *smartphone* android dalam lingkungan yang terhubung dengan *Internet*.
2. Aplikasi ini menggunakan peta yang berbasis *google maps* .
3. Komunikasi data antara aplikasi dengan *server* diimplementasikan menggunakan pertukaran data dengan format JSON.
4. GPS *receiver* yang digunakan adalah GPS dengan teknologi *Asisted*-GPS (A-GPS) yang tertanam pada *smartphone* android.
5. *Web service* yang digunakan untuk komunikasi data dibangun dengan PHP.
6. Pembuatan *layout* antarmuka aplikasi menggunakan bahasa pemrogaman XML.

5.3 Implementasi *Database*

Implementasi penyimpanan data dilakukan dengan *database management system* MySQL. Hasil implementasi penyimpanan data ini berupa *script – script* SQL. Hasil implementasi SQL pada *database* ini dimodelkan dalam diagram konseptual *entity relationship*. Pada konseptual *entity relationship* dapat dilihat hubungan relasi antar tabel seperti *foreign key*. Gambar 5.1 menggambarkan diagram konseptual *entitiy relationship* dari aplikasi jejaring sosial kampus berbasis GPS.



Gambar 5.1 Diagram konseptual *entity relationship* dari sosial kampus

Sumber : [Implementasi]

5.4 Implementasi *Class* pada *File* Program

Setiap *class* yang telah dirancang pada proses perancangan direalisasikan pada sebuah *file* program dengan format java (.java). Tabel 5.4 menjelaskan mengenai hubungan antara *class* dengan *file* program yang digunakan untuk mengimplentasikannya.

Tabel 5.4 Implementasi *class* pada kode program *.java

No.	Paket	Nama *Class*	Nama *File* Program
1	android.campus	Intro	Intro.java
2		Login	Login.java
3		Home	Home.java
4		Profile	Profile.java
5		Frndprofile	Frndprofile.java
6		Setting	Setting.java
7		Search	Search.java
8		Map	Map.java
9		Admin	Admin.java
10		Container	Container.java
11		Help	Help.java
12	android.adapter	TimelineAdapter	TimelineAdapter.java
13		UserAdapter	UserAdapter.java
14		ViewHolder	ViewHolder.java
15	android.logic	Notice	Notice.java
16		User	User.java
17		Base64	Base64.java
18		AutoComplete	AutoComplete.java
19		CustomOverlayItem	CustomOverlayItem.java
20	android.overlay	MapOverlay	MapOverlay.java
21		DirectionOverlay	DirectionOverlay.java
22		BallonItemizedOverlay	BallonItemizedOverlay.java
23		BallonOverlayView	BallonOverlayView.java
24		CustomBallonOverlayView	CustomBallonOverlayView.java

Sumber : [Implementasi]

5.5 Implementasi Algoritma

Aplikasi jejaring sosial ini mempunyai beberapa proses (*method*) utama yang terbagi dalam beberapa *class*. Pada penulisan skripsi ini hanya dicantumkan algoritma dari beberapa proses saja sehingga tidak tidak semua algoritma *method* akan dicantumkan. Algoritma proses yang dicantumkan antara lain adalah proses *login*, mencari teman, melakukan *posting*, melihat *timeline*, menambah teman, melihat lokasi, melihat jalur, dan *edit account*. Implementasi algoritma ini akan direpresentasikan dalam bentuk *pseudocode*.

5.5.1 Algoritma Proses *Login*

Login dilakukan dengan memasukkan *userid* dan *password* pada *EditText* kemudian menekan tombol *Login*. Algoritma *login* akan dijalankan saat tombol *Login* ditekan. Gambar 5.2 merupakan algoritma untuk *login*.

NAMA ALGORITMA : login

DEKLARASI

- Userid adalah String, variabel untuk menampung nilai masukan userid.
- Password adalah String, variabel untuk menampung nilai masukan password.
- baris adalah String, variabel untuk menampung tiap baris reponse dari web service.
- isi adalah String, variabel untuk menampung semua baris response dari web service.

DESKRIPSI

Masukan : userid,password
Proses :

1. Periksa data masukan.
2. Jika data masukan (userid dan password)dari EditText belum diisi maka jalankan method showResult() untuk menampilkan pemberitahuan “userid harus diisi”.
3. Jika sudah, data masukan dijadikan parameter dari webservice untuk mengakses tabel user.
4. Mencoba melakukan beberapa proses yang antara lain:
   a. Koneksi ke web service.
   b. response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris.
   c. Kumpulan baris response di masukkan ke dalam variabel isi.
   d. Notifikasi ditampilkan dengan menjalankan *method* showResult(isi);
   e. Jika isi bernilai “Login Sukses”, maka akan masuk ke halaman home sesuai dengan hak akses yang dimiliki.
   f. Jika isi bernilai “Admin Mode”, maka akan masuk ke halaman admin.
5. Jika gagal maka jalankan method showResult() dengan pesan peringatan “loss connection”

Keluaran : status login.

Gambar 5.2 Implementasi algoritma *login*

Sumber : [Implementasi]

Implementasi algoritma *login* kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `login()` ditunjukan pada Gambar 5.3.

```
1  Public void login() {
2  HttpClient httpClient = new DefaultHttpClient();
3            HttpPost httpPost = new HttpPost("http://"
4  + host+ "/Soscamp/login.php");
5  ArrayList<NameValuePair> param =
6  new ArrayList<NameValuePair>();
7
8  if (user.getText().toString().matches("")&&
9  pass.getText().toString().matches(""))
10 {
11       showResult("Isi Userid !!");
12 } else {
13 param.add(new BasicNameValuePair
14 ("userid",user.getText().toString()));
15 param.add(new BasicNameValuePair
16 ("password", pass.getText().toString()));
17
18       try {
19 httpPost.setEntity(new UrlEncodedFormEntity(param));
20 HttpResponse httpRespose =
21 httpClient.execute(httpPost);
22 HttpEntity httpEntity = httpRespose.getEntity();
23 InputStream in = httpEntity.getContent();
24 BufferedReader read = new BufferedReader(new
25 InputStreamReader(in));
26 String isi = "";
27 String baris = "";
28
29 while ((baris = read.readLine()) != null)
30 {
31    isi += baris;
32 }
33
34 showResult(isi);
35
36 if (isi.equals("Login Sukses")) {
37 finish();
38 Intent home = new Intent(this, Home.class);
39 home.putExtra("sesion_id",user.getText().toString());
40 startActivity(home);
41 }
42 else if (isi.equals("Admin Mode")) {
43 Intent home = new Intent(this, Admin.class);
44 home.putExtra("sesion_id",
45 user.getText().toString());
```

```
46  startActivity(home);
47  }
48
49  } catch (ClientProtocolException e) {
50        showResult ("koneksi server gagal");
51  } catch (IOException e) {
52        showResult ("Loss Connection");
53        }
54
55     }
56  }
```

Gambar 5.3 *Source code* implementasi algoritma *login*

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `login ()` adalah sebagai berikut :

1. Algoritma untuk memeriksa data masukan (proses ke-1) di implementasikan pada *source code* baris 8-9.
2. Algoritma proses ke-2 di implementasikan pada *source code* baris 11 yaitu menjalankan *method* `showResult()`.
3. *Source code* baris 12-16 merupakan implementasi dari algoritma proses ke-3, yaitu memasukkan nilai parameter *web service*.
4. *Source code* baris 18 merupakan implementasi dari algoritma proses ke-4, yaitu mencoba melakukan beberapa proses.
5. *Source code* baris 19-21 merupakan implementasi dari algoritma proses 4a, yaitu koneksi ke *web service.*
6. *Source code* baris 22-33 merupakan implementasi dari algoritma proses 4b-4c, yaitu menangkap *response* dari *web service.*
7. Algoritma proses 4d di implementasikan pada *source code* baris 34 yaitu menjalankan *method* `showResult(isi)`.
8. Algoritma proses 4e di implementasikan pada *source code* baris 36 - 41 yaitu masuk ke halaman *home.*
9. Algoritma proses 4f di implementasikan pada *source code* baris 42- 47 yaitu masuk ke halaman admin.

10. *Source code* baris 49-53 merupakan implementasi dari algoritma proses ke-5, yaitu eksepsi jika terjadi kegagalan.

5.5.2 Algoritma Mencari Teman

Mencari teman dilakukan dengan memasukkan *username* ke dalam *EditText* pada halaman pencarian. Algoritma *searching* akan dijalankan saat tombol *search* ditekan oleh *user*. Gambar 5.5 merupakan algoritma untuk mencari teman.

```
NAMA ALGORITMA : searching

DEKLARASI

    • Username adalah EditText, variabel untuk menampung
      nilai masukan username.
    • adapter adalah object dari Klas UserAdapter.

DESKRIPSI
Masukan : username
Proses :

   1. Jika username kosong, panggil method showResult("isi
      username !!").
   2. Kosongkan adapter.
   3. Jalankan method initAdapter()yang berisi :
        a. Inisialisasi adapter dengan parameter getUser().
        b. Jalankan method setListAdapter(adapter) untuk
           menampilkan view di layar.

Keluaran :  list dari user yang dicari.
```

Gambar 5.4 Implementasi algoritma mencari teman

Sumber : [Implementasi]

Implementasi algoritma mencari teman kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `searching()` ditunjukan pada Gambar 5.5.

```
1  public void searching(View v){
2
3  if(username.getText().toString().equals(""))
4  {
5      showToast("isi username!!");
6  }
7  else
8  {
9        adapter.clear();
10       initAdapter();
11       AnimationUtils.makeInChildBottomAnimation(this);
12        in=AnimationUtils.loadAnimation
13        (this,android.R.anim.fade_in);
14        getListView().startAnimation(in);
15
16  }
17 }
```

Gambar 5.5 *Source code* implementasi algoritma mencari teman

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `searching()` adalah sebagai berikut :

1. Algoritma untuk memeriksa data masukan (proses ke-1) di implementasikan pada *source code* baris 3-6.
2. Algoritma proses ke-2 di implementasikan pada *source code* baris 9 yaitu menjalankan *method* `adapter.clear()`.
3. *Source code* baris 10 -14 merupakan implementasi dari algoritma proses ke-3, yaitu menampilkan *list* dari *user* yang dicari ke layar.

5.5.3 Algoritma Melakukan *Posting*

User melakukan *posting* dengan menambah isi *posting* pada *field* yang ada pada antar muka aplikasi, kemudian menekan tombol *POST*. Algoritma *post* akan dijalankan saat tombol *POST* ditekan oleh *user*. Gambar 5.6 merupakan algoritma untuk melakukan *posting*.

NAMA ALGORITMA : post

DEKLARASI

- Userid adalah String, variabel untuk menampung nilai masukan userid.
- notice adalah String, variabel untuk menampung nilai masukan notice.
- lat adalah String, variabel untuk menampung nilai latitude.
- lon adalah String, variabel untuk menampung nilai langitude.
- reply_to adalah String, variabel untuk menampung nilai id dari posting.
- baris adalah String, variabel untuk menampung tiap baris reponse dari webservice.
- isi adalah String, variabel untuk menampung semua baris response dari webservice.

DESKRIPSI

Masukan : userid,notice,lat,lon,reply_to
Proses :

1. Mencoba melakukan beberapa proses antara lain:
   a. notice, lat, lon dan reply_to dijadikan parameter dari webservice untuk menambahkan data notice ke database.
   b. Koneksi ke web service.
   c. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris.
   d. Kumpulan baris response di masukkan kedalam variabel isi.
   e. Notifikasi ditampilkan dengan menjalankan *method* showResult(isi);
2. Jika gagal maka jalankan method showResult() dengan pesan peringatan “loss connection”

Keluaran : status posting.

Gambar 5.6 Implementasi algoritma melakukan *posting*

Sumber : [Implementasi]

Implementasi algoritma melakukan *posting* kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `post()` ditunjukan pada Gambar 5.7.

```
 1  public void post(String notice) {
 2  HttpClient httpClient = new DefaultHttpClient();
 3  HttpPost httpPost = new
 4  HttpPost("http://"+host+"/Soscamp/post.php");
 5  ArrayList<NameValuePair> param = new
 6  ArrayList<NameValuePair>();
 7  param.add(new BasicNameValuePair("userid", userid));
 8  param.add(new BasicNameValuePair("post", notice));
 9  param.add(new BasicNameValuePair("lat", lat));
10  param.add(new BasicNameValuePair("lon", lon));
11  param.add(new BasicNameValuePair("replyto",replyto));
12
13    try {
14  httpPost.setEntity(new UrlEncodedFormEntity(param));
15  HttpResponse httpRespose =
16  httpClient.execute(httpPost);
17  HttpEntity httpEntity = httpRespose.getEntity();
18  InputStream in = httpEntity.getContent();
19  BufferedReader read = new BufferedReader(new
20  InputStreamReader(in));
21
22  String isi = "";
23  String baris = "";
24  while ((baris = read.readLine()) != null) {
25            isi += baris;
26  }
27       showResult(isi);
28
29  } catch (ClientProtocolException e) {
30            showResult ("koneksi server gagal");
31  } catch (IOException e) {
32            showResult("Loss Connection");
33     }
34
35  }
```

Gambar 5.7 *Source code* implementasi algoritma *posting*

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `post()` adalah sebagai berikut :

1. Algoritma proses ke-1 di implementasikan pada *source code* baris 13.
2. *Source code* baris 14 merupakan implementasi dari algoritma proses 1a, yaitu memasukkan nilai parameter *web service*.

3. *Source code* baris 16 merupakan implementasi dari algoritma proses 1b, yaitu koneksi ke *web service*.
4. *Source code* baris 17-25 merupakan implementasi dari algoritma proses 1c-1d, yaitu menangkap *response* dari *web service*.
5. Algoritma proses 1e di implementasikan pada *source code* baris 27 yaitu menjalankan *method* `showResult(isi)`.
6. *Source code* baris 29-32 merupakan implementasi dari algoritma proses ke-2, yaitu eksepsi jika terjadi kegagalan.

5.5.4 Algoritma Melihat *Timeline*

Melihat *timeline* dilakukan pada saat *user* berhasil *login* dan masuk ke halaman *home*. Algoritma getTimeline akan dijalankan saat halaman *home* diakses oleh *user*. Gambar 5.8 merupakan algoritma untuk melihat *timeline*.

NAMA ALGORITMA : getTimeline

DEKLARASI

- Userid adalah String, variabel untuk menampung nilai masukan userid (sesion).
- Item adalah object dari Klas Notice.
- Notice_item adalah List dari Notice, variabel untuk menampung data Notice.
- line adalah String, variabel untuk menampung tiap baris reponse dari webservice.
- content adalah String, variabel untuk menampung semua baris response dari webservice.

DESKRIPSI

Masukan : userid
Proses :

1. Sesion Userid dijadikan parameter dari webservice untuk mengakses database kemudian mengirimkan hasilnya ke aplikasi.
2. Mencoba melakukan beberapa proses antara lain:
   a. Koneksi ke web service.
   b. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel line.
   c. Kumpulan baris response di masukkan kedalam variabel content.
   d. Content yang merupakan string dengan format JSON array diubah menjadi data notice kemudian ditampung pada method-method yang ada di objek item.

```
        e. Kumpulan data item di tampung pada list
           Notice_item.
   3. Jika gagal maka jalankan method showResult() dengan
      pesan peringatan “loss connection”.

Keluaran :  nilai dari Notice_item.
```

Gambar 5.8 Implementasi algoritma melihat *timeline*

Sumber : [Implementasi]

Implementasi algoritma melihat *timeline* kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `getTimeline()` ditunjukan pada Gambar 5.9.

```
 1 public List<Notice> getTimeline() {
 2 HttpClient httpClient = new DefaultHttpClient();
 3 HttpPost httpPost = new
 4 HttpPost("http://"+host+"/Soscamp/gettimeline.php");
 5 // parameter
 6 ArrayList<NameValuePair> param = new
 7 ArrayList<NameValuePair>();
 8 param.add(new BasicNameValuePair("userid", userid));
 9 try {
10 // add parameter
11 httpPost.setEntity(new UrlEncodedFormEntity(param));
12 HttpResponse httpRespose =
13 httpClient.execute(httpPost);
14 HttpEntity httpEntity = httpRespose.getEntity();
15 // read content
16 InputStream in = httpEntity.getContent();
17 BufferedReader read = new BufferedReader(new
18 InputStreamReader(in));
19
20 String content = "";
21 String line = "";
22
23 while ((line = read.readLine()) != null)
24 {
25 content += line;
26 }
27
28 // json
29 try {
30 JSONArray jArr = new JSONArray(content);
31 for (int i = 0; i < jArr.length(); i++)
32 {
```

```
33 JSONObject jObj = jArr.getJSONObject(i);
34 Notice item = new Notice();
35 item.setId(jObj.getString("post_id"));
36 item.setUser_id(jObj.getString("user_id"));
37 item.setNama(jObj.getString("nama"));
38 item.setPost(jObj.getString("post"));
39 item.setDate(jObj.getString("date"));
40 item.setTime(jObj.getString("time"));
41 item.setReply_to(jObj.getString("reply_to"));
42 item.setAvatar
43 (getImage("http://"+host+"/Soscamp/"+jObj.getString("
44 url")));
45 notice_item.add(item);
46 }
47 } catch (JSONException e) {
48 e.printStackTrace();
49 }
50 } catch (ClientProtocolException e) {
51 e.printStackTrace();
52 } catch (IOException e) {
53 showResult ("Loss Connection");
54 }
55 return notice_item;
56 }
```

Gambar 5.9 *Source code* implementasi algoritma *getTimeline*

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `getTimeline()` adalah sebagai berikut :

1. *Source code* baris 8 merupakan implementasi dari algoritma proses ke-1, yaitu memasukkan nilai parameter *web service*.
2. *Source code* baris 9 merupakan implementasi dari algoritma proses ke-2, yaitu mencoba melakukan beberapa proses.
3. *Source code* baris 11-13 merupakan implementasi dari algoritma proses 2a, yaitu koneksi ke *web service*.
4. *Source code* baris 14-25 merupakan implementasi dari algoritma proses 2b-2c, yaitu menangkap *response* dari *web service*.
5. Algoritma proses 2d di implementasikan pada *source code* baris 29-45 yaitu memasukkan nilai *response* ke dalam objek item.

6. Algoritma proses 2e di implementasikan pada *source code* baris 46 yaitu menampung kumpulan objek item ke dalam *list* notice_item.
7. *Source code* baris 47-53 merupakan implementasi dari algoritma proses ke-5, yaitu eksepsi jika terjadi kegagalan.

5.5.5 Algoritma Menambah Teman

Menambah teman dilakukan dengan menekan tombol "+" yang ada pada halaman *profile user* lain. Algoritma *followOperation* akan dijalankan saat tombol "+" ditekan oleh *user*. Gambar 5.10 merupakan algoritma untuk menambah teman.

NAMA ALGORITMA : followOperation

DEKLARASI

- Userid adalah String, variabel untuk menampung nilai masukan userid.
- id adalah String, variabel untuk menampung nilai userid dari user lain.
- mode adalah String, variabel untuk menampung nilai mode *follow* atau *unfollow*.
- baris adalah String, variabel untuk menampung tiap baris reponse dari webservice.
- isi adalah String, variabel untuk menampung semua baris response dari webservice.

DESKRIPSI

Masukan : useid,id,mode
Proses :

1. userid, id dan mode dengan nilai + (follow) dijadikan parameter dari webservice untuk mengakses database.
2. Mencoba melakukan beberapa proses antara lain:
   a. Koneksi ke webservice.
   b. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris.
   c. Kumpulan baris response di masukkan kedalam variabel isi.
   d. Jika isi tidak kosong, panggil method showResult(isi)untuk menampilkan notifikasi.
   e. Jika isi kosong, panggil method showResult("gagal").
3. Jika gagal maka jalankan method showResult() dengan pesan peringatan "loss connection"

Keluaran : Status proses penambahan teman.

Gambar 5.10 Implementasi algoritma menambah teman

Sumber : [Implementasi]

Implementasi algoritma menambah teman kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `followOperation()` ditunjukan pada Gambar 5.11.

```
1  public void followOperation() {
2
3  HttpClient httpClient = new DefaultHttpClient();
4  HttpPost httpPost = new
5  HttpPost("http://"+host+"/Soscamp/follow.php");
6  ArrayList<NameValuePair> param = new
7  ArrayList<NameValuePair>();
8  param.add(new BasicNameValuePair("mode",
9  follow.getText().toString()));
10 param.add(new BasicNameValuePair("userid", userid));
11 param.add(new BasicNameValuePair("follow", id));
12 try {
13      httpPost.setEntity(new
14 UrlEncodedFormEntity(param));
15      HttpResponse httpRespose =
16 httpClient.execute(httpPost);
17      HttpEntity httpEntity = httpRespose.getEntity();
18      InputStream in = httpEntity.getContent();
19      BufferedReader read = new BufferedReader(new
20 InputStreamReader(in));
21      String isi = "";
22      String baris = "";
23
24      while ((baris = read.readLine()) != null) {
25                  isi += baris;
26                }
27
28      if (!isi.equals("null")) {
29           showResult(isi);
30           }
31
32      else {
33           showResult("Gagal");
34           }
35
36      } catch (ClientProtocolException e) {
37                showResult("Koneksi server gagal");
38           } catch (IOException e) {
39                showResult("Loss Connection");
40           }
41 }
```

Gambar 5.11 *Source code* implementasi algoritma menambah teman

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `followOperation()` adalah sebagai berikut :

1. Algoritma proses ke-1 di implementasikan pada *source code* baris 6-11, yaitu memasukkan nilai parameter *web service*.
2. *Source code* baris 12 merupakan implementasi dari algoritma proses ke-2, yaitu mecoba melakukan beberapa proses.
3. *Source code* baris 15-16 merupakan implementasi dari algoritma proses 2a, yaitu koneksi ke *web service*.
4. *Source code* baris 17-26 merupakan implementasi dari algoritma proses 2b-2c, yaitu menangkap *response* dari *web service*.
5. Algoritma proses 2d di implementasikan pada *source code* baris 28-30 yaitu menjalankan *method* `showResult(isi)`jika isi tidak kosong.
6. Algoritma proses 2e di implementasikan pada *source code* baris 32-34 yaitu menjalankan *method* `showResult("gagal")`jika isi kosong.
7. *Source code* baris 36-40 merupakan implementasi dari algoritma proses ke-3, yaitu eksepsi jika terjadi kegagalan.

5.5.6 Algoritma Melihat Lokasi

User melihat *lokasi* dari *user* lain dengan mengakses menu *map* pada halaman *user* lain. Algoritma melihat lokasi akan dijalankan saat halaman *map* diakses oleh *user*. Algoritma melihat lokasi merupakan proses menambahkan *overlay marker* ke dalam peta. Gambar 5.12 merupakan algoritma untuk melihat lokasi.

NAMA ALGORITMA : ShowMarker
DEKLARASI

- Myposition dan geopoint adalah Geopoint, variabel untuk menampung nilai koordinat lintang dan bujur.
- icon adalah Drawable, variabel untuk menampung gambar icon sebagai marker.
- mapView adalah MapView, objek dari klas MapView untuk menampilkan peta pada layar.
- User_loc adalah list dari klas User.
- alamatA dan alamatB adalah String, variabel untuk menampung nilai alamat dari koordinat lintang dan bujur.
- item adalah OverlayItem, objek dari klas OverlayItem.
  overlay adalah MapOverlay, object dari klas MapOverlay

DESKRIPSI
Masukan : useid,id
Proses :

1. Jika terdapat overlay, hapus overlay lama.
2. Inisialisasi lokasi dari smartphone ke variabel myposition.
3. Inisialisasi icon untuk sebagai marker lokasi di map.
4. Mencoba untuk mengubah koordinat lokasi menjadi nama tempat kemudian masukan hasilnya ke variable alamatA.
5. Jika gagal isi alamatA dengan “no location found”.
6. membuat objek overlay dengan parameter icon.
7. Membuat objek item dengan parameter informasi lokasi dari device.
8. Tambahkan item ke dalam overlay dengan memanggil method overlay.addItem(item).
9. Tambahkan overlay ke dalam mapview dengan method mapView.getoverlays().add(overlay).
10. Untuk jumlah item yang ada pada user_loc lakukan perulangan proses:
    a. Inisialisasi geopoint dengan koordinat lokasi dari user lain.
    b. Hitung jarak dari locationA ke locationB.
    c. Lakukan seleksi icon sesuai dengan hak akses user lain.
    d. membuat objek overlay dengan parameter icon.
    e. Mencoba untuk mengubah koordinat lokasi menjadi nama tempat kemudian masukan hasilnya ke variable amalatB.
    f. Jika gagal isi alamatB dengan “no location found”.
    g. Kumpulan data user di tampung pada list user_loc.
    h. Membuat objek item dengan parameter informasi lokasi dari user lain.
    i. Tambahkan item ke dalam overlay dengan memanggil method overlay.addItem(item)
11. Pindahkan center dari mapview ke posisi device dengan method mapView.getController().animateTo(myposition).
12. Gambar ulang peta pada mapview dengan memanggil method mapView.postInvalidate().

Keluaran : user berhasil melihat lokasi dari user lain di mapview.

Gambar 5.12 Implementasi algoritma melihat lokasi

Sumber : [Implementasi]

Implementasi algoritma melihat lokasi kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `showMarker()` dapat dilihat pada Gambar 5.13.

```
protected void showMarker(Location newLocation)
{
   List<Overlay> overlays = mapView.getOverlays();
  // remove old overlay
   if (overlays.size() > 0) {
   for (Iterator<Overlay> iterator =
overlays.iterator();iterator.hasNext();)
 {
      iterator.next();
      iterator.remove();
 }
}


Awal =
newLocation.getLatitude()+","+newLocation.getLongitu
de();
// transform the location to a geopoint
GeoPoint geopoint = new GeoPoint((int)
(newLocation.getLatitude() * 1E6),(int)
(newLocation.getLongitude() * 1E6));
GeoPoint myposition = geopoint;
Location locationA = new Location("point A");
Location locationB = new Location("point B");
locationA.setLatitude(geopoint.getLatitudeE6() /
1E6);
locationA.setLongitude(geopoint.getLongitudeE6() /
1E6);

// inisialisasi icon
Drawable icon =
getResources().getDrawable(R.drawable.marker1);
icon.setBounds(0, 0, icon.getIntrinsicWidth(),
icon.getIntrinsicHeight());

// Geocoder
Geocoder geoCoder = new Geocoder(this);
  try {
        List<Address> addresses =
geoCoder.getFromLocation(locationA.getLatitude(),loc
ationA.getLongitude(), 1);
   StringBuilder sb = new StringBuilder();
             if (addresses.size() > 0) {
               Address address = addresses.get(0);
               for (int i = 0; i <
address.getMaxAddressLineIndex(); i++)

sb.append(address.getAddressLine(i)).append("\n");

sb.append(address.getLocality()).append("\n");
sb.append(address.getPostalCode()).append("\n");
               sb.append(address.getCountryName());
```

```
53                }
54                alamatA = sb.toString();
55
56                alamatA = addresses.get(0).toString();
57              } catch (IOException e) {
58
59                  alamatA = "No location found";
60
61              }
62
63
64  // buat overlay dan tampilkan
65
66  overlay = new MapOverlay(icon, this);
67  OverlayItem item = new OverlayItem(geopoint, "My
68  Location", "Lat:"+ locationA.getLatitude() +
69  "\nLng:" + locationA.getLongitude()+ "\n"+alamatA);
70  overlay.addItem(item);
71  mapView.getOverlays().add(overlay);
72
73  for (int i = 0; i < user_loc.size(); i++)
74  {
75
76   tujuan =
77  user_loc.get(i).getLatitude()+","+user_loc.get(i).ge
78  tLongitude();
79  // arah tujuan
80  geopoint = new GeoPoint((int)
81  (Double.parseDouble(user_loc.get(i).getLatitude()) *
82  1E6), (int)
83  (Double.parseDouble(user_loc.get(i).getLongitude())
84  * 1E6));
85  locationB.setLatitude(geopoint.getLatitudeE6() /
86  1E6);
87  locationB.setLongitude(geopoint.getLongitudeE6() /
88  1E6);
89
90  float distance = locationA.distanceTo(locationB);
91  // seleksi icon
92  if (user_loc.get(i).getAkses()==1)
93  {
94  icon =
95  getResources().getDrawable(R.drawable.marker2);
96  }
97  else if (user_loc.get(i).getAkses()==2)
100 {
101 icon =
102 getResources().getDrawable(R.drawable.marker3);
103 }
104 else
105 {
106 icon =
```

```
getResources().getDrawable(R.drawable.marker3);
}


icon.setBounds(0, 0,
icon.getIntrinsicWidth(),icon.getIntrinsicHeight());
overlay = new MapOverlay(icon, this);

// Geocoding
geoCoder = new Geocoder(this, Locale.getDefault());
try {
      List<Address> addresses =
geoCoder.getFromLocation(
geopoint.getLatitudeE6()  / 1E6,
geopoint.getLongitudeE6() / 1E6, 1);
        if (addresses.size() > 0){
for (int
j=0;j<addresses.get(0).getMaxAddressLineIndex();j++)
{
alamatB += addresses.get(0).getAddressLine(j) +
"\n";
 }
 }
}catch (IOException e) {
  e.printStackTrace();
}


item = new OverlayItem(geopoint,
user_loc.get(i).getNama(), "Lat:" +
locationB.getLatitude() + "\nLng:"+
locationB.getLongitude() + "\n\nJarak : " +
distance+" m"/*\n  \n"+alamatB*/);
overlay.addItem(item);
mapView.getOverlays().add(overlay);
}
// ke lokasi kita
mapView.getController().animateTo(myposition)
// gambar ulang map
     mapView.postInvalidate();
}
```

Gambar 5.13 *Source code* implementasi algoritma melihat lokasi

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `showMarker()` adalah sebagai berikut :

1. Algoritma proses ke-1 di implementasikan pada *source code* baris 5-12, yaitu menghapus *overlay* yang lama.
2. *Source code* baris 22 merupakan implementasi dari algoritma proses ke-2, yaitu inisialisasi variabel *myposition*.
3. *Source code* baris 31-34 merupakan implementasi dari algoritma proses 3, yaitu inisialisasi *icon*.
4. *Source code* baris 38-56 merupakan implementasi dari algoritma proses ke-4, yaitu mencoba mengubah koordinat lokasi menjadi nama tempat.
5. Algoritma proses ke-5 di implementasikan pada *source code* baris 57-61 yaitu eksepsi dengan memasukkan pesan "No location found" pada variabel alamatA.
6. Algoritma proses ke-6 di implementasikan pada *source code* baris 66 yaitu membuat objek overlay dan menampilkannya.
7. *Source code* baris 67-69 merupakan implementasi dari algoritma proses ke-7, yaitu membuat objek item.
8. Algoritma proses ke-8 di implementasikan pada *source code* baris 70, yaitu menambahkan item ke dalam *overlay*.
9. Algoritma proses ke-9 di implementasikan pada *source code* baris 71, yaitu menambahkan *overlay* ke dalam *mapview*.
10. Algoritma proses ke-10 di implementasikan pada *source code* baris 73-144, yaitu melakukan pengulangan proses untuk menampilkan lokasi dari *user* lain.
11. *Source code* baris 146 merupakan implementasi dari algoritma proses ke-11, yaitu memindahkan *center* dari peta pada posisi *device*.
12. *Source code* baris 148 merupakan implementasi dari algoritma proses ke-12, yaitu menggambar ulang peta.

5.5.7 Algoritma Melihat Jalur

Melihat jalur dilakukan dengan menekan tombol jalur yang ada pada halaman *map*. Algoritma *showDirection* akan dijalankan saat tombol jalur ditekan oleh *user*. Gambar 5.14 merupakan algoritma untuk melihat jalur.

NAMA ALGORITMA : showDirection
DEKLARASI

- awal adalah String, variabel untuk menampung nilai koordinat lokasi point awal jalur.
- tujuan adalah String, variabel untuk menampung nilai koordinat lokasi point tujuan jalur.
- mapView adalah MapView, variabel untuk menampilkan Mapview.
- startGP,gp1,gp2 adalah Geopoint, objek untuk menampung koordinat lokasi dalam bentuk geopoint.
- start,line,end adalah objek dari klas DirectionOverlay, start merupakan objek overlay di awal jalur(startGP,startGP), line merupakan objek overlay pada jalur yang dilalui(gp1,gp2), dan end adalah objek dari overlay di akhir jalur(gp2,gp2).

DESKRIPSI
Masukan : awal,tujuan
Proses :

1. inisialisasi koordinat di variable awal menjadi startGP.
2. Tambahkan overlay ke mapView dengan menjalankan method add (start).
3. Inisialisasi gp1 dan gp2.
4. Lakukan perulangan proses sebanyak nilai panjang jalur :
   a. inisialisasi koordinat di variable tujuan menjadi gp2.
   b. inisialisasi koordinat jalur yang dilalui dari gp1 ke gp2.
   c. Tambahkan overlay ke mapView dengan menjalankan method add (line)
5. Tambahkan overlay ke mapView dengan menjalankan method add (end)
6. Menggambar ulang mapView dengan method postInvalidate().
7. jalankan method initLocationManagaer() yang berisi :
   a. panggil method ShowMarker() dan updLocation() ketika koordinat GPS berubah.
   b. Konfigurasi untuk auto update lokasi.

Keluaran : user berhasil melihat jalur menuju lokasi dari user lain.

Gambar 5.14 Implementasi algoritma melihat jalur

Sumber : [Implementasi]

Implementasi algoritma melihat jalur kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `ShowDirection()` dapat dilihat pada Gambar 5.15.

```
 1  Public void ShowDirection(View v)
 2  {
 3
 4  String pairs[] = getDirectionData(awal,tujuan);
 5  String[] lngLat = pairs[0].split(",");
 6
 7   // STARTING POINT
 8  GeoPoint startGP = new GeoPoint(
 9  (int) (Double.parseDouble(lngLat[1]) * 1E6), (int)
10  (Double.parseDouble(lngLat[0]) * 1E6));
11
12   start = new DirectionOverlay(startGP, startGP);
13   mapView.getOverlays().add(start);
14
15   // Gambar Jalur
16   GeoPoint gp1;
17   GeoPoint gp2 = startGP;
18
19   for (int i = 1; i < pairs.length; i++)
20  {
21      lngLat = pairs[i].split(",");
22      gp1 = gp2;
23      gp2 = new GeoPoint((int)
24  (Double.parseDouble(lngLat[1]) * 1E6),(int)
25  (Double.parseDouble(lngLat[0]) * 1E6));
26      line = new DirectionOverlay(gp1, gp2);
27      mapView.getOverlays().add(line);
28
29  }
30
31    // END POINT
32  end = new DirectionOverlay(gp2, gp2);
33  mapView.getOverlays().add(end);
34  mapView.getController().animateTo(gp2);
35
36  }
```

Gambar 5.15 *Source code* implementasi algoritma melihat jalur

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `ShowDirection()` adalah sebagai berikut :

1. Algoritma proses ke-1 di implementasikan pada *source code* baris 6-8, yaitu instansiasi objek startGP..
2. *Source code* baris 10 merupakan implementasi dari algoritma proses ke-2, yaitu menambahkan awal jalur pada *mapview*.

3. *Source code* baris 13-14 merupakan implementasi dari algoritma proses ke-3, yaitu inisialisasi gp1 dan gp2.
4. *Source code* baris 16-25 merupakan implementasi dari algoritma proses ke-4, yaitu melakukan pengulangan proses 4a-4c sepanjang jalur yang dilalui.
5. Algoritma proses ke-5 di implementasikan pada *source code* baris 29 yaitu menambahkan akhir jalur pada *mapview*.
6. Algoritma proses ke-6 di implementasikan pada *source code* baris 33 yaitu menggambar ulang peta.
7. *Source code* baris 35 merupakan implementasi dari algoritma proses ke-7, yaitu menjalankan *method* `initLocationManager()`.

5.5.8 Algoritma *Edit Account*

User melakukan *edit account* dengan mengganti isi dari *EditText* yang ada pada halaman *setting*, kemudian menekan tombol *save*. Algoritma *updateAccount* akan dijalankan saat tombol *save* ditekan oleh *user*. Gambar 5.16 merupakan algoritma pada proses *edit account*.

NAMA ALGORITMA : updateAccount

DEKLARASI

- Userid, jb, gn adalah String, variabel untuk menampung nilai masukan.
- name dan email adalah EditText,variable untuk menampung nilai input dari EditText.

DESKRIPSI

Masukan : userid, name,email,jb,gn,pass,pass1,pass2

Proses :

1. Ketika tombol save ditekan, kirimkan data parameter web service (userid, name, jb, gn, dan email).
2. Mencoba melakukan beberapa proses antara lain:
   a. Koneksi ke web service.
   b. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris.
   c. Kumpulan baris response di masukkan kedalam variabel isi.
   d. Jika isi tidak kosong,Notifikasi ditampilkan dengan menjalankan *method* showResult(isi);
   e. Jika kosong,tampilkan pesan kesalahan dengan menjalankan *method* showResult(“Gagal”);

```
   3. Jika gagal maka jalankan method showResult() dengan
      pesan peringatan “loss connection”

Keluaran : status proses update account..
```

Gambar 5.16 Implementasi algoritma *edit account*

Sumber : [Implementasi]

Implementasi algoritma *edit account* kemudian ditulis dalam *source code* dengan java sebagai *platform* pengembangan. *Source code method* `updateAccount()` ditunjukan pada Gambar 5.17.

```
 1  public void updateAccount(View v) {
 2
 3  HttpClient httpClient = new DefaultHttpClient();
 4  HttpPost httpPost = new
 5  HttpPost("http://"+host+"/Soscamp/upduser.php");
 6  ArrayList<NameValuePair> param = new
 7  ArrayList<NameValuePair>();
 8  param.add(new BasicNameValuePair("userid", userid));
 9  param.add(new BasicNameValuePair("nama",
10  name.getText().toString()));
11  param.add(new BasicNameValuePair("akses", jb));
12  param.add(new BasicNameValuePair("jk", gn));
13  param.add(new BasicNameValuePair("email",
14  email.getText().toString()));
15
16   try {
17  httpPost.setEntity(new UrlEncodedFormEntity(param));
18  HttpResponse httpRespose =
19  httpClient.execute(httpPost);
20  HttpEntity httpEntity = httpRespose.getEntity();
21  InputStream in = httpEntity.getContent();
22  BufferedReader read = new BufferedReader(new
23  InputStreamReader(in));
24  String isi = "";
25  String baris = "";
26
27  while ((baris = read.readLine()) != null)
28  {
29    isi += baris;
30  }
31
32  if (!isi.equals("null"))
33  {
34    showResult(isi);
35  }
```

```
36 else {
37   showResult("Gagal");
38 }
39
40
41 } catch (ClientProtocolException e) {
42           showResult("Koneksi gagal");
43    } catch (IOException e) {
44                 showResult("Loss Connection");
45           }
46      }
```

Gambar 5.17 *Source code* implementasi algoritma *edit account*

Sumber : [Implementasi]

Penjelasan hubungan antara implementasi *source code* dengan *pseodocode* algoritma *method* `updateAccount()` adalah sebagai berikut :

1. Algoritma proses ke-1 di implementasikan pada *source code* baris 6-14, yaitu memasukkan nilai parameter *web service*.
2. *Source code* baris 16-39 merupakan implementasi dari algoritma proses ke-2, yaitu mencoba melakukan beberapa proses.
3. *Source code* baris 42-45 merupakan implementasi dari algoritma proses ke-3, yaitu eksepsi jika terjadi kegagalan.

5.6 Implementasi Antarmuka Aplikasi

Antarmuka aplikasi jejaring sosial kampus terdiri dari dua buah menu utama yaitu menu khusus admin dan menu untuk *user* biasa. Menu khusus admin meliputi menu *login*, menu pencarian, *reset password*, dan *delete user*. Menu untuk *user* meliputi menu registrasi, *login*, *home*, *profile*, *friend profile*, *setting*, *map*, dan *search*.

5.6.1 Menu *Login*

Menu *login* merupakan fasilitas yang disediakan aplikasi untuk masuk ke sistem. Pada saat *login*, pengguna diharuskan mengisi *userid* dan *password*. Pada menu ini juga terdapat menu *help* untuk bantuan penggunaan. *Layout* menu *login* diatur pada login.xml. Pemanggilan login.xml dilakukan pada *method*

`onCreate()` yang terdapat pada Login.java. *Login* dilakukan dengan menyentuh tombol *Login* untuk menjalankan *method* `login()`. Pada menu *Login* terdapat *submenu Sign Up* yang berisi *form* untuk melakuka registrasi. *Submenu help* dapat diakses dengan menekan *keypad* menu pada *smartphone*. Gambar 5.18 menunjukkan tampilan menu *login*.

Gambar 5.18 Tampilan menu *login*

Sumber : [Implementasi]

5.6.2 Menu *Home*

Menu *home* ditampilkan ketika *user* berhasil *login* ke aplikasi. *Layout* menu *home* diatur pada home.xml. Pemanggilan home.xml dilakukan menggunakan *method* `onCreate()` yang terdapat pada Home.java.

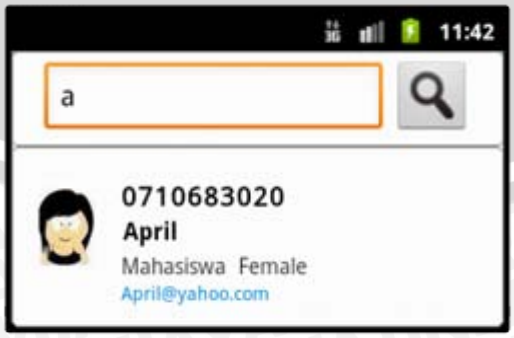
Pada halaman ini terdapat *submenu* untuk menambah *posting*. *Submenu* tersebut dapat diakses dengan menekan *keypad menu* yang terdapat pada *smartphone*. Menambah *posting* dilakukan dengan memilih menu *compose* (*add posting*) yang kemudian akan menjalankan *method* `onOptionMenuSelected()` untuk memanggil antar muka *posting*. Gambar 5.19 menunjukkan tampilan menu *home*.

Gambar 5.19 Tampilan menu *home*

Sumber : [Implementasi]

5.6.3 Menu Pencarian

Layout menu pencarian diatur pada search.xml. Pemanggilan search.xml dilakukan menggunakan *method* `onCreate()` yang terdapat pada Search.java. Pencarian *user* dilakukan dengan memasukkan *username* dari *user* yang dicari pada *EditText* (pada search.xml diberi id: *search*). Ketika tombol cari di tekan, *method* `search(View v)` akan dijalankan dan halaman ini akan menampilkan *list* dari *user* yang *username*-nya sesuai dengan masukan. Tampilan menu pencarian dapat dilihat pada Gambar 5.20.



Gambar 5.20 Tampilan menu pencarian

Sumber : [Implementasi]

5.6.4 Menu Registrasi

Menu Registrasi merupakan fasilitas yang disediakan aplikasi untuk mendaftar ke sistem. Untuk mengakses menu ini, pengguna dapat menekan *sliding drawer Sign Up* yang terdapat pada bagian bawah halaman *login*. Pada saat registrasi, pengguna diharuskan mengisi *userid* yang valid (NIM atau NIP). *Field* yang harus diisi pada halaman ini antara lain adalah *userid*, *username*, jabatan, *gender*, *email* dan *password*. Gambar 5.21 menunjukkan tampilan menu registrasi.

Gambar 5.21 Tampilan menu registrasi

Sumber : [Implementasi]

5.6.5 Menu *Profile*

Menu *profile* berisi biodata dan kumpulan *posting* dari *user*. *Layout* menu *profile* diatur pada profile.xml. Pemanggilan profile.xml dilakukan menggunakan *method* `onCreate()` yang terdapat pada Profile.java.

Pada halaman ini, *user* dapat menghapus *posting* miliknya dengan memilih salah satu *item posting* yang akan dihapus pada *listview* kemudian menekan lama item tersebut sampai *context menu* hapus muncul. Menu *profile* mempunyai *submenu* untuk melihat info dan melihat *conversation*. Sub menu tersebut dapat diakses dengan menekan *keypad menu* yang terdapat pada *smartphone*. Gambar 5.22 menunjukkan tampilan menu *profile*.

Gambar 5.22 Tampilan menu *profile*

Sumber : [Implementasi]

5.6.6 Menu *Friend Profile*

Menu *friend profile* berisi biodata dan kumpulan *posting* dari *teman*. *Layout* menu *profile* diatur pada frnprofile.xml. Pemanggilan frnprofile.xml dilakukan menggunakan *method* `onCreate()` yang terdapat pada FrnProfile.java.

Pada halaman ini, terdapat tombol untuk menghapus atau menambah pertemanan (pada frnprofile.xml diberi id: btnfollow). Ketika tombol tersebut ditekan, maka aplikasi akan menjalankan *method* `FollowOperation()`. Gambar 5.23 menunjukkan tampilan menu *friend profile*.

Gambar 5.23 Tampilan menu *friend profile*

Sumber : [Implementasi]

5.6.7 Menu *Setting*

Menu *setting* merupakan menu yang disediakan aplikasi untuk mengatur *account* dari *user*. Menu ini berisi *form* untuk meng-*edit* biodata dan *avatar* (gambar *profile*). *Layout* menu *profile* diatur pada setting.xml. Pemanggilan setting.xml dilakukan menggunakan *method* `onCreate()` yang terdapat pada Setting.java.

Pada halaman ini, terdapat *submenu* untuk mengganti *password*. *Submenu* tersebut dapat diakses dengan menekan *keypad menu* yang terdapat pada *smartphone*. Ketika *submenu edit password* ditekan, aplikasi akan menjalankan *method* `onOptionMenuSelected()` untuk menampilkan antar muka *edit password*. Gambar 5.24 menunjukkan tampilan menu *setting*.

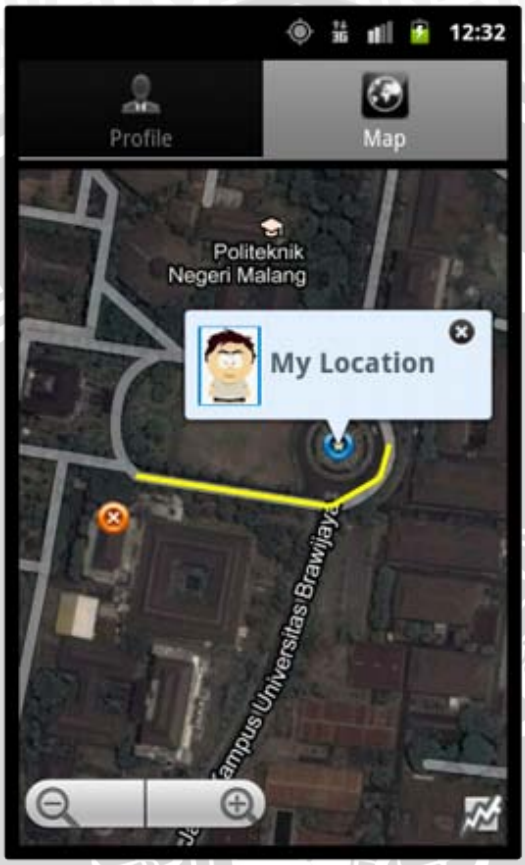
Gambar 5.24 Tampilan menu *setting*

Sumber : [Implementasi]

5.6.8 Menu *Map*

Pada halaman ini terdapat peta yang menampilkan lokasi dari *user* lain. *Layout* menu *map* diatur pada map.xml. Pemanggilan map.xml dilakukan menggunakan *method* `onCreate()` yang terdapat pada Map.java.

User dapat melihat jalur yang harus dilalui dengan menekan tombol jalur (pada map.xml diberi id: btnjalur). Ketika tombol ditekan, aplikasi akan menjalankan *method* `ShowDirection()` untuk menampilkan jalur ke peta. Gambar 5.25 menunjukkan tampilan menu *map*.



Gambar 5.25 Tampilan menu *map*
Sumber : [Implementasi]

5.6.9 Kendala Dalam Implementasi

Proses implementasi mempunyai kendala yaitu terbatasnya alokasi memori untuk DVM (*Davlik Virtual Machine*) dengan *emulator*. Untuk melakukan komunikasi data secara *realtime* dengan *server* dalam jumlah *user* yang banyak dibutuhkan alokasi memori yang banyak. Jika alokasi memori kurang, maka memungkinkan terjadinya *memory leaks* pada aplikasi. Hal ini dapat diatasi dengan cara menambah alokasi memori DVM pada AVD *manager* menggunakan opsi *memory virtual machine* minimal 128. Ukuran ini bisa diubah sesuai dengan keinginan dengan mempertimbangkan RAM komputer.

BAB VI
PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan proses pengujian dan analisis terhadap aplikasi jejaring sosial yang telah dibangun. Proses pengujian dilakukan melalui tiga tahapan (strategi) yaitu pengujian unit, pengujian integrasi dan pengujian validasi. Pada pengujian unit dan pengujian integrasi digunakan teknik pengujian *White Box (White Box Testing)*. Pada pengujian validasi digunakan teknik pengujian *Black Box (Black Box Testing)*. Proses analisis dilakukan dengan menggunakan *apache benchmark* untuk mengetahui performa dari jejaring sosial berbasis gps.

6.1 Pengujian

Proses pengujian dilakukan melalui tiga tahapan (strategi) yaitu pengujian unit, pengujian integrasi dan pengujian validasi.

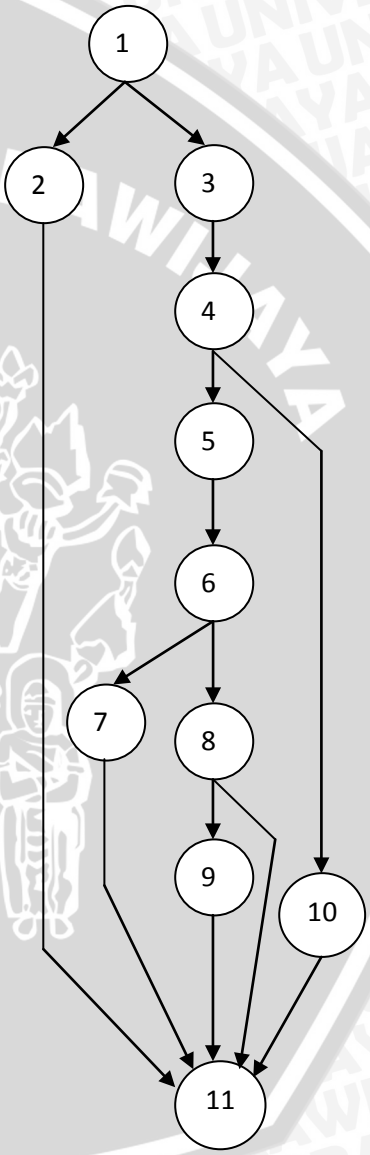
6.1.1 Pengujian Unit

Sistem yang dibangun dengan paradigma pemrograman berorientasi objek menerapkan pengujian unit untuk suatu metode (operasi) dari suatu klas. Pada pengujian unit aplikasi ini digunakan teknik pengujian *White Box (White Box Testing)* dengan teknik *Basis Path Testing*. Pada teknik *Basis Path Testing*, proses pengujian dilakukan dengan memodelkan algoritma pada suatu *flow graph*, menentukan jumlah kompleksitas siklomatis (*cyclomatic complexity*), menentukan sebuah basis set dari jalur independen dan memberikan kasus uji (*test case*) pada setiap basis set yang telah ditentukan. Pada penulisan laporan skripsi ini hanya dicantumkan hasil pengujian unit untuk algoritma dari beberapa metode (operasi) yang ada pada aplikasi jejaring sosial (tidak untuk keseluruhan metode).

6.1.1.1 Pengujian Unit pada Operasi `login()`

Operasi `login()` merupakan operasi untuk autentifikasi *user* ke sistem. Operasi ini ada pada klas Login.java. Tabel 6.1 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `login()`.

Tabel 6.1 Pemodelan algoritma `login()` ke dalam *flow graph*

Node	Operasi	Flow graph
1 2 3 4 5	**NAMA ALGORITMA** : login **DEKLARASI** • Userid adalah String, variabel untuk menampung nilai masukan userid. • Password adalah String, variabel untuk menampung nilai masukan password. • baris adalah String, variabel untuk menampung tiap baris reponse dari webservice. • isi adalah String, variabel untuk menampung semua baris response dari webservice. **DESKRIPSI** Masukan : userid,password Proses : 1. periksa data masukan. 2. Jika data masukan (userid dan password)dari EditText belum diisi maka jalankan *method* showResult() untuk menampilkan pemberitahuan “userid harus diisi”. 3. Jika sudah, data masukan dijadikan parameter dari webservice untuk mengakses tabel user. 4. Mencoba melakukan beberapa proses yang antara lain: a. Koneksi ke web service. b. response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris. c. Kumpulan baris response di masukkan ke dalam variabel isi. d. Notifikasi ditampilkan dengan menjalankan *method* showResult(isi);	Node (N) = 11 Edge (E) = 14 

```
6       e. Jika isi menyatakan
           "Login Sukses", maka
           akan masuk ke
7          halaman home sesuai
           dengan hak akses
           yang dimiliki.
8       f. Jika isi menyatakan
           "Admin", maka akan
9          masuk ke halaman
           admin.
10   5. Jika gagal maka jalankan
        method showResult()
        dengan pesan peringatan
        "loss connection"

11  Keluaran : status login.
```

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `login()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2

= 14 – 11+ 2

= 5

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan lima buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-11

Jalur 2 : 1-3-4-5-6-7-11

Jalur 3 : 1-3-4-5-6-8-9-11

Jalur 4 : 1-3-4-5-6-8-11

Jalur 5 : 1-3-4-10-11

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.2.

Tabel 6.2 *Test case* untuk pengujian unit operasi `login()`

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	*Field Userid dan password* kosong ketika tombol *login* ditekan.	Aplikasi menampilkan peringatan bahwa *field userid* dan *password* harus diisi.	Aplikasi menampilkan peringatan bahwa *field userid* dan *password* harus diisi.
2	*Userid dan password* valid dengan hak akses sebagai *user*.	*User* berhasil *login* dan masuk ke halaman *home*.	*User* berhasil *login* dan masuk ke halaman *home*.
3	*Userid dan password* valid dengan hak akses sebagai admin.	Admin berhasil *login* dan masuk ke halaman admin.	Admin berhasil *login* dan masuk ke halaman admin.
4	*Userid dan password* tidak valid.	*User* tidak berhasil *login*.	*User* tidak berhasil *login*.
5	Aplikasi tidak dapat mengakses *server* atau eksepsi.	Data tidak diproses dan akan ditampilkan pesan *error*.	Data tidak diproses dan akan ditampilkan pesan *error*.

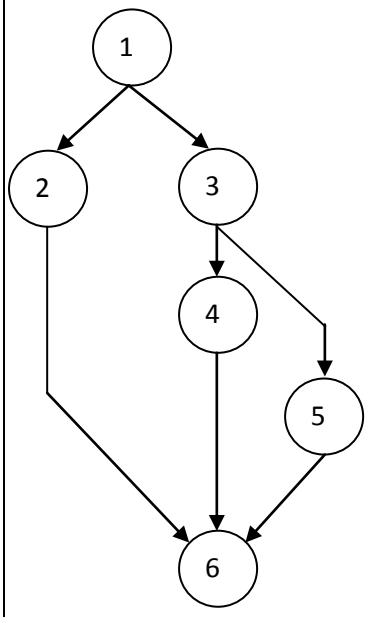
Sumber: [Pengujian]

6.1.1.2 Pengujian Unit pada Operasi `post()`

Operasi `post()` merupakan operasi untuk melakukan *posting* ke dalam aplikasi. Operasi ini ada pada klas Home.java. Tabel 6.3 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `post()`.

Tabel 6.3 Pemodelan `post()` ke dalam *flow graph*

Node	Operasi	Flow graph
	NAMA ALGORITMA : post **DEKLARASI** • Userid adalah String, variabel untuk menampung nilai masukan userid. • notice adalah String, variabel untuk menampung nilai masukan notice. • lat adalah String, variabel untuk menampung	Node (N) = 6 Edge (E) = 7

	nilai latitude. • lon adalah String, variabel untuk menampung nilai langitude. • reply_to adalah String, variabel untuk menampung nilai id dari posting. • baris adalah String, variabel untuk menampung tiap baris reponse dari webservice. • isi adalah String, variabel untuk menampung semua baris response dari webservice. **DESKRIPSI** Masukan : userid,notice,lat,lon,reply_to Proses : 1. Memeriksa data notice pada EditText. 2. Jika variable notice kosong, panggil method showResult("Empty post"). 3. Mencoba melakukan beberapa proses : a. notice, lat, lon dan reply_to dijadikan parameter dari webservice untuk menambahkan data notice ke database. b. Koneksi ke web service. c. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris. d. Kumpulan baris response di masukkan kedalam variabel isi. e. Notifikasi ditampilkan dengan menjalankan *method* showResult(isi); 4. Jika gagal maka jalankan method showResult() dengan pesan peringatan "loss connection" Keluaran : status posting.	

1 2 3 4 5 6

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `post()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2
= 7 – 6 + 2
= 3

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan tiga buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-6
Jalur 2 : 1-3-4-6
Jalur 3 : 1-3-5-6

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.4.

Tabel 6.4 *Test case* untuk pengujian unit operasi `post()`

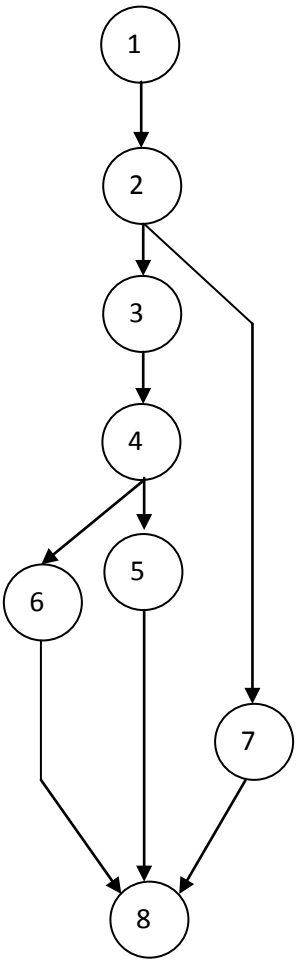
Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	*Field posting* belum diisi ketika tombol *POST* ditekan.	Data tidak diproses dan muncul pesan "*empty post*".	Data tidak diproses dan muncul pesan "*empty post*".
2	*User mengisi posting* dan tombol *POST* ditekan.	Data di-*posting* dan aplikasi menampilkan notifikasi bahwa proses berhasil.	Data di-*posting* dan aplikasi menampilkan notifikasi bahwa proses berhasil.
3	Aplikasi tidak dapat mengakses *server* atau eksepsi.	Data tidak diproses dan akan ditampilkan pesan *error*.	Data tidak diproses dan akan ditampilkan pesan *error*.

Sumber: [Pengujian]

6.1.1.3 Pengujian Unit pada Operasi `followOperation()`

Operasi `followoperation()` merupakan operasi untuk menambah *user* lain sebagai teman. Operasi ini ada pada klas FrnProfile.java. Tabel 6.5 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `followOperation()`.

Tabel 6.5 Pemodelan `followOperation()` ke dalam *flow graph*

Node	Operasi	Flow graph
	NAMA ALGORITMA : followOperation **DEKLARASI** • Userid adalah String, variabel untuk menampung nilai masukan userid. • id adalah String, variabel untuk menampung nilai userid dari user lain. • mode adalah String, variabel untuk menampung nilai mode *follow* atau *unfollow*. • line adalah String, variabel untuk menampung tiap baris reponse dari webservice. • content adalah String, variabel untuk menampung semua baris response dari webservice. **DESKRIPSI** Masukan : useid,id,mode Proses : 1. userid, id dan mode dengan nilai + (follow) dijadikan parameter dari webservice untuk mengakses database. 2. Mencoba melakukan beberapa proses antara lain: a. Koneksi ke webservice. b. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel line. c. Kumpulan baris	Node (N) = 8 Edge (E) = 9 

1
2
3

	response di masukkan kedalam variabel content.	
4	d. Jika content tidak kosong, panggil method showResult(content)untuk menampilkan notifikasi.	
5		
6	e. Jika content kosong, panggil method showResult(“gagal”)untuk menampilkan peringatan kesalahan.	
	3. Jika gagal maka jalankan method showResult() dengan pesan peringatan “loss connection”	
7		
8	Keluaran : Status proses penambahan teman.	

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `followOperation()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2

= 9 – 8 + 2

= 3

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan tiga buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-3-4-5-8

Jalur 2 : 1-2-3-4-6-8

Jalur 3 : 1-2-7-8

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.6.

Tabel 6.6 *Test case* untuk pengujian unit operasi `followOperation()`

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Proses menambah teman berhasil.	Data pertemanan ditambah dan muncul notifikasi bahwa proses berhasil.	Data pertemanan ditambah dan muncul notifikasi bahwa proses berhasil.
2	*Web service* tidak memberikan *response*	Data tidak diproses dan aplikasi menampilkan peringatan bahwa proses gagal.	Data tidak diproses dan aplikasi menampilkan peringatan bahwa proses gagal.
3	Aplikasi tidak dapat mengakses *server* atau eksepsi.	Data tidak diproses dan akan ditampilkan pesan *error*.	Data tidak diproses dan akan ditampilkan pesan *error*.

Sumber: [Pengujian]

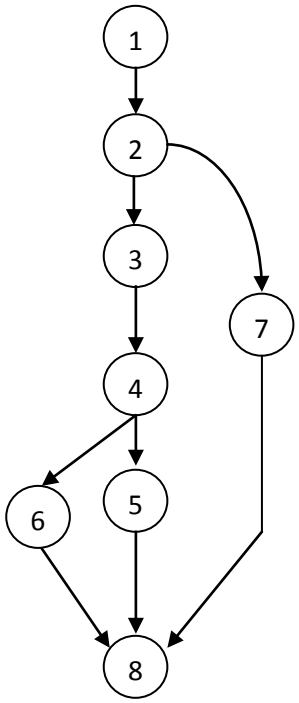
6.1.1.4 Pengujian Unit Operasi `updateAccount()`

Operasi `updateAccount()` merupakan operasi untuk melakukan *update* data *user* di dalam aplikasi. Operasi ini ada pada klas Setting.java. Tabel 6.7 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `updateAccount()`.

Tabel 6.7 Pemodelan `updateAccount()` ke dalam *flow graph*

Node	Operasi	Flow graph
	NAMA ALGORITMA : updateAccount **DEKLARASI** • Userid, jb, gn adalah String, variabel untuk menampung nilai masukan. • name dan email adalah EditText,variable untuk menampung nilai input dari EditText. **DESKRIPSI** Masukan :userid, name, email, jb, gn, pass, pass1, pass2	Node (N) = 8 Edge (E) = 9

1 2 3 4 5 6 7 8	Proses : 1. Ketika tombol save ditekan, kirimkan data parameter web service (userid, name, jb, gn, dan email). 2. Mencoba melakukan beberapa proses antara lain: a. Koneksi ke web service. b. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel baris. c. Kumpulan baris response di masukkan kedalam variabel isi. d. Jika isi tidak kosong,Notifikasi ditampilkan dengan menjalankan *method* showResult(isi); e. Jika kosong,tampilkan pesan kesalahan dengan menjalankan *method* showResult(“Gagal”); 3. Jika gagal maka jalankan *method* showResult() dengan pesan peringatan “loss connection” Keluaran : status proses update account.



Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `updateAccount()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2

= 9 – 8 + 2

= 3

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan tiga buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-3-4-5-8

Jalur 2 : 1-2-3-4-6-8

Jalur 3 : 1-2-7-8

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.8.

Tabel 6.8 *Test case* untuk pengujian unit operasi `updateAccount()`

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Data masukan valid dan *web service* memberikan *response*	Data diproses dan muncul notifikasi "*updated*" yang menyatakan bahwa proses berhasil.	Data diproses dan muncul notifikasi "*updated*" yang menyatakan bahwa proses berhasil.
2	*Web service* tidak memberikan *response.*	Data tidak diproses dan aplikasi menampilkan peringatan bahwa proses gagal.	Data tidak diproses dan aplikasi menampilkan peringatan bahwa proses gagal.
3	Aplikasi tidak dapat mengakses *server* atau eksepsi.	Data tidak diproses dan akan ditampilkan pesan *error*.	Data tidak diproses dan akan ditampilkan pesan *error*.

Sumber: [Pengujian]

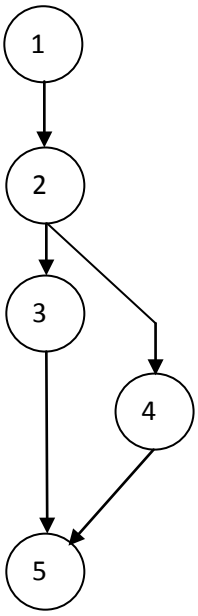
6.1.2 Pengujian Integrasi

Pengujian integrasi diterapkan pada proses yang mengintegrasikan fungsionalitas dari beberapa *class* untuk melakukan sebuah operasi tertentu. Pada pengujian integrasi yang dijadikan sebagai objek uji adalah *class-class* yang menggabungkan kinerja dari *class-class* yang lain. Pengujian integrasi yang dilakukan terhadap aplikasi ini menggunakan strategi *bottom-up*, dimana modul-modul yang diintegrasikan masing-masing diuji terlebih dahulu dalam pengujian unit dan kemudian bergerak menuju ke pengujian modul-modul kontrol yang mengintegrasikannya.

6.1.2.1 Pengujian Integrasi pada Operasi `getTimeline()`

Operasi `getTimeline()` merupakan operasi untuk mendapatkan data *notice* dari *database*. Operasi ini ada pada klas Home.java. Tabel 6.9 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `getTimeline()`.

Tabel 6.9 Pemodelan `getTimeline()` ke dalam *flow graph*

Node	Operasi	Flow graph
	NAMA ALGORITMA : getTimeline **DEKLARASI** • Userid adalah String, variabel untuk menampung nilai masukan userid (sesion). • Item adalah object dari Klas Notice. • Notice_item adalah List dari Notice, variabel untuk menampung data Notice. • line adalah String, variabel untuk menampung tiap baris reponse dari webservice. • content adalah String, variabel untuk menampung semua baris response dari webservice. **DESKRIPSI** Masukan : userid Proses : 1. Sesion Userid dijadikan parameter dari webservice untuk mengakses database. 2. Mencoba melakukan beberapa proses antara lain: a. Koneksi ke web service. b. Response dari webservice diubah menjadi string kemudian tiap baris response ditampung pada variabel line.	Node (N) = 5 Edge (E) = 5 

3	c. Kumpulan baris response di masukkan kedalam variabel content. d. Content yang merupakan string dengan format JSON array diubah menjadi data notice kemudian ditampung pada *method-method* yang ada di objek item. e. Kumpulan data item di tampung pada list Notice_item.	
4	3. Jika gagal maka jalankan *method* showResult() dengan pesan peringatan "loss connection"	
5	Keluaran : nilai dari Notice_item.	

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `getTimeline()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node)*.

V(G) = E – N + 2

= 5 – 5 + 2

= 2

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan dua buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-3-5

Jalur 2 : 1-2-4-5

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.10.

Tabel 6.10 *Test case* untuk pengujian integrasi operasi `getTimeline()`

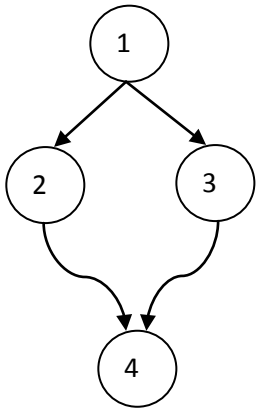
Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Parameter sesuai dan *request* data ke *web service* berhasil.	Data *posting* dari teman muncul di *timeline*.	Data *posting* dari teman muncul di *timeline*.
2	Aplikasi tidak dapat mengakses *server* atau eksepsi.	Data tidak diproses dan akan ditampilkan pesan *error*.	Data tidak diproses dan akan ditampilkan pesan *error*.

Sumber: [Pengujian]

6.1.2.2 Pengujian Integrasi pada Operasi `searching()`

Operasi `searching()` merupakan operasi untuk melakukan pencarian *user* lain dengan parameter *username*. Operasi ini ada pada klas Search.java. Tabel 6.11 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `searching()`.

Tabel 6.11 Pemodelan `searching()` ke dalam *flow graph*

Node	Operasi	Flow graph
1 2 3	**NAMA ALGORITMA** : searching **DEKLARASI** • Username adalah EditText, variabel untuk menampung nilai masukan username. • adapter adalah object dari Klas UserAdapter. **DESKRIPSI** Masukan : username Proses : 1. Jika username kosong, panggil *method* showResult(“isi username !!”). 2. Jika username sudah diisi, Kosongkan adapter. 3. Jalankan *method* initAdapter()yang berisi : a. Inisialisasi adapter dengan parameter getUser().	Node (N) = 4 Edge (E) = 4 

4	b. Jalankan *method* setListAdapter(adapter) untuk menampilkan view di layar. Keluaran : list dari user yang dicari.	

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `searching()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2
= 4 – 4 + 2
= 2

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan dua buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-4

Jalur 2 : 1-3-4

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.12.

Tabel 6.12 *Test case* untuk pengujian integrasi operasi `searching()`

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	*Field username* belum diisi ketika tombol search ditekan.	menampilkan peringatan bahwa *field username* harus diisi.	menampilkan peringatan bahwa *field username* harus diisi.
2	*Username* diisi dan tombol *search* ditekan.	Aplikasi menampilkan daftar *user* yang sesaui dengan *username* yang dimasukan.	Aplikasi menampilkan daftar *user* yang sesaui dengan *username* yang dimasukan.

Sumber: [Pengujian]

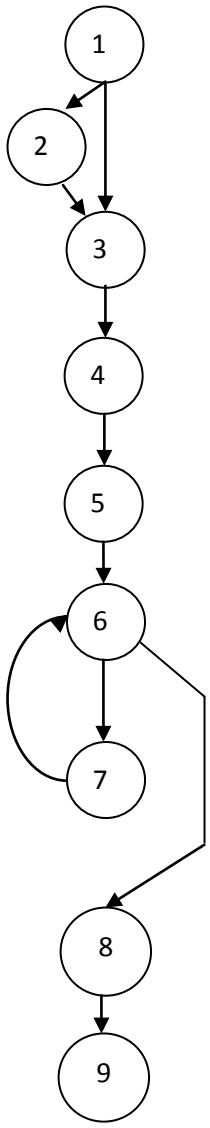
6.1.2.3 Pengujian Integrasi pada Operasi `showMarker()`

Operasi `showMarker()` adalah *method* yang terdapat pada *class* Map dan merupakan implementasi dari algoritma untuk melihat lokasi. Tabel 6.13 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `showMarker()`.

Tabel 6.13 Pemodelan `showMarker()` ke dalam *flow graph*

Node	Operasi	Flow graph
	NAMA ALGORITMA : showMarker **DEKLARASI** • Myposition dan geopoint adalah Geopoint, variabel untuk menampung nilai koordinat lintang dan bujur. • icon adalah Drawable, variabel untuk menampung gambar icon sebagai marker. • mapView adalah MapView, objek dari klas MapView untuk menampilkan peta pada layar. • User_loc adalah list dari klas User. • alamatA dan alamatB adalah String, variabel untuk menampung nilai alamat dari koordinat lintang dan bujur. • item adalah CustomOverlayItem, objek dari klas CustomOverlayItem. • overlay adalah MapOverlay, object dari klas MapOverlay **DESKRIPSI** Masukan : useid,id Proses : 1. Jika terdapat overlay, hapus overlay lama. 2. Inisialisasi lokasi dari smartphone ke variabel myposition. 3. Inisialisasi icon untuk sebagai marker lokasi di map. 4. Mencoba untuk mengubah koordinat lokasi menjadi nama tempat kemudian masukan hasilnya ke variable alamatA. 5. Jika gagal isi alamatA dengan	Node (N) = 9 Edge (E) = 10

1
2
3
4

	“no location found”.	
5	6. membuat objek overlay dengan parameter icon. 7. Membuat objek item dengan parameter informasi lokasi dari device. 8. Tambahkan item ke dalam overlay dengan memanggil method overlay.addItem(item). 9. Tambahkan overlay ke dalam mapview dengan *method* mapView.getoverlays().add(overlay).	
6	10. Untuk jumlah item yang ada pada user_loc lakukan perulangan proses:	
7	a. Inisialisasi geopoint dengan koordinat lokasi dari user lain. b. Hitung jarak dari locationA ke locationB. c. Lakukan seleksi icon sesuai dengan hak akses user lain. d. membuat objek overlay dengan parameter icon. e. Mencoba untuk mengubah koordinat lokasi menjadi nama tempat kemudian masukan hasilnya ke variable amalatB. f. Jika gagal isi alamatB dengan “no location found”. g. Kumpulan data user di tampung pada list user_loc. h. Membuat objek item dengan parameter informasi lokasi dari user lain. i. Tambahkan item ke dalam overlay dengan memanggil *method* overlay.addItem(item)	
8	11. Pindahkan center dari mapview ke posisi device dengan *method* mapView.getController(). animateTo(myposition). 12. Gambar ulang peta pada mapview dengan memanggil *method* mapView.postInvalidate().	
9	Keluaran : user berhasil melihat lokasi dari user lain di mapview.	

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `ShowMarker()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2
= 10 – 9 + 2
= 3

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan tiga buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-3-4-5-6-7-...

Jalur 2 : 1-2-3-4-5-6-7-...

Jalur 3 : 1-2-3-4-5-6-8-9

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.14.

Tabel 6.14 *Test case* untuk pengujian integrasi operasi `showMarker()`

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Belum ada *overlay marker* yang ditampilkan pada peta.	Aplikasi menampilkan *overlay* koordinat posisi *user* pada peta.	Aplikasi menampilkan *overlay* koordinat posisi *user* pada peta.
2	Sudah ada *overlay marker* yang ditampilkan pada peta.	Aplikasi menghapus *overlay* lama dan kemudian menampilkan *overlay* koordinat yang baru pada peta.	Aplikasi menghapus *overlay* lama dan kemudian menampilkan *overlay* koordinat yang baru pada peta.
3	Semua jumlah item pada user_loc sudah dihitung.	Aplikasi memindahkan pusat tampilan dari peta ke koordinat yang baru.	Aplikasi memindahkan pusat tampilan dari peta ke koordinat yang baru.

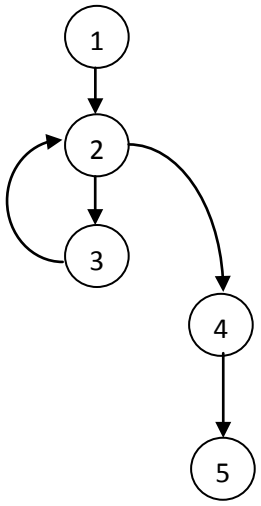
Sumber: [Pengujian]

6.1.2.4 Pengujian Unit pada Operasi `showDirection()`

Operasi `showDirection()` merupakan operasi untuk menampilkan jalur menuju lokasi dari *user* lain. Operasi ini ada pada klas Map.java. Tabel 6.15 menunjukkan proses pemodelan dalam *flow graph* pada algoritma *method* `showDirection()`.

Tabel 6.15 Pemodelan `showDirection()` ke dalam *flow graph*

Node	Operasi	Flow graph
	NAMA ALGORITMA : showDirection **DEKLARASI** • awal adalah String, variabel untuk menampung nilai koordinat lokasi point awal jalur. • tujuan adalah String, variabel untuk menampung nilai koordinat lokasi point tujuan jalur. • mapView adalah MapView, variabel untuk menampilkan Mapview. • startGP,gp1,gp2 adalah Geopoint, objek untuk menampung koordinat lokasi dalam bentuk geopoint. • start,line,end adalah objek dari klas DirectionOverlay, start merupakan objek overlay di awal jalur(startGP,startGP), line merupakan objek overlay pada jalur yang dilalui(gp1,gp2), dan end adalah objek dari overlay di akhir jalur(gp2,gp2). **DESKRIPSI** Masukan : awal,tujuan Proses : 1. inisialisasi koordinat di variable awal menjadi startGP. 2. Tambahkan overlay ke mapView dengan menjalankan *method* add (start). 3. Inisialisasi gp1 dan gp2. 4. Lakukan perulangan proses sebanyak nilai panjang jalur :	Node (N) = 5 Edge (E) = 5
1		
2		

3	a. inisialisasi koordinat di variable tujuan menjadi gp2. b. inisialisasi koordinat jalur yang dilalui dari gp1 ke gp2. c. Tambahkan overlay ke mapView dengan menjalankan *method* add (line)	
4	5. Tambahkan overlay ke mapView dengan menjalankan *method* add (end) 6. Menggambar ulang mapView dengan *method* postInvalidate(). 7. jalankan *method* initLocationManagaer() yang berisi : a. panggil *method* ShowMarker() dan updLocation() ketika koordinat GPS berubah. b. Konfigurasi untuk auto update lokasi.	
5	Keluaran : user berhasil melihat jalur menuju lokasi dari user lain.	

Sumber: [Pengujian]

Pemodelan ke dalam *flow graph* yang telah dilakukan terhadap operasi `showDirection()` menghasilkan jumlah kompleksitas siklomatis *(cyclomatic complexity)* melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi (garis penghubung antar *node*) dan N merupakan simpul *(node).*

V(G) = E – N + 2

= 5 – 5 + 2

= 2

Dari hasil perhitungan nilai *cyclomatic complexity* di atas, dapat ditentukan dua buah *basis set* dari jalur *independent*, yaitu:

Jalur 1 : 1-2-3-…

Jalur 2 : 1-2-4-5

Penentuan kasus uji untuk masing-masing jalur dan hasil eksekusi untuk masing-masing kasus uji dijelaskan pada Tabel 6.16.

Tabel 6.16 *Test case* untuk pengujian integrasi operasi `showDirection()`

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Tombol jalur ditekan dan panjang jalur lebih dari "0" (nol).	Jalur yang harus dilalui dari lokasi kita ke lokasi *user* lain muncul di peta.	Jalur yang harus dilalui dari lokasi kita ke lokasi *user* lain muncul di peta.
2	Ketika nilai jarak jalur yang dilalui sudah sama dengan panjang jalur.	Berhenti menggambar jalur.	Berhenti menggambar jalur.

Sumber: [Pengujian]

6.1.3 Pengujian Validasi

Pengujian validasi digunakan untuk mengetahui apakah sistem yang dibangun sudah benar sesuai dengan yang dibutuhkan. Item - item yang telah dirumuskan dalam daftar kebutuhan dan merupakan hasil analisis kebutuhan akan menjadi acuan untuk melakukan pengujian validasi. Pengujian validasi menggunakan metode pengujian *Black Box*, karena tidak diperlukan konsentrasi terhadap alur jalannya algoritma program dan lebih ditekankan untuk menemukan konformitas antara kinerja sistem dengan daftar kebutuhan.

6.1.3.1 Kasus Uji Validasi

Pada skripsi ini dilakukan pengujian validasi terhadap aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android. Untuk mengetahui kesesuaian antara kebutuhan dengan kinerja sistem, pada setiap kebutuhan (*requirement*) sistem dilakukan proses pengujian dengan kasus uji masing-masing yang ditunjukan dalam tabel-tabel di bawah.

Tabel 6.17 Kasus uji untuk pengujian validasi *Register*

Nama Kasus Uji	*Register*
Objek Uji	SRS-001-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas registrasi bagi *user*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman registrasi 2. Calon *user* mengisi data yang diperlukan pada *form* yang disediakan. 3. Calon *user* menekan tombol *register*.
Hasil yang diharapkan	Aplikasi memeriksa data masukan, jika data yang dimasukan valid maka data akan ditambahkan ke *database* dan menampilkan notifikasi bahwa *user* baru sudah terdaftar. Jika data yang diisikan tidak valid atau masih ada *form* yang belum diisi, aplikasi akan menampilkan pesan kesalahan.

Sumber: [Pengujian]

Tabel 6.18 Kasus uji untuk pengujian validasi *login* sah

Nama Kasus Uji	Kasus Uji *Login* Sah
Objek Uji	SRS-001-02 dan SRS-006-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas *login* bagi *user* dan admin.
Prosedur Uji	1. *User* dan admin membuka aplikasi. 2. *Form login* ditampilkan. 3. *User* dan admin memasukkan *userid* dan *password* di dalam *form login*. 4. *User* dan admin menekan tombol *Login*.
Hasil yang diharapkan	Aplikasi dapat melakukan penyeleksian kondisi *login* pada *database* berdasar data yang dimasukkan dan jika penyeleksian kondisi ini benar, maka akan masuk ke sistem sesuai dengan hak aksesnya.

Sumber: [Pengujian]

Tabel 6.19 Kasus uji untuk pengujian validasi *login* tidak sah

Nama Kasus Uji	Kasus Uji *Login* Tidak Sah
Objek Uji	SRS-001-02 dan SRS-006-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas *login* bagi *user* dan admin.
Prosedur Uji	1. *User* dan admin membuka aplikasi. 2. *Form login* ditampilkan. 3. *User* dan admin memasukkan *userid* dan *password* di dalam *form login*. 4. *User* dan admin menekan tombol *Login*.
Hasil yang diharapkan	Aplikasi dapat melakukan penyeleksian kondisi *login* pada *database* berdasar data yang dimasukkan dan jika penyeleksian kondisi ini salah, maka tidak akan masuk ke sistem dan aplikasi menampilkan pesan kesalahan.

Sumber: [Pengujian]

Tabel 6.20 Kasus uji untuk pengujian validasi *logout*

Nama Kasus Uji	Kasus Uji Logout
Objek Uji	SRS-001-03
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas *login* bagi *user* dan admin.
Prosedur Uji	1. Aplikasi dijalankan pada keadaan sudah *login* 2. *User* menekan tombol *Logout*.
Hasil yang diharapkan	Aplikasi melakukan penghapusan session yang aktif dan menampilkan halaman *login*.

Sumber: [Pengujian]

Tabel 6.21 Kasus uji untuk pengujian validasi melakukan *posting*

Nama Kasus Uji	Kasus Uji Melakukan *Posting*
Objek Uji	SRS-002-01

Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk melakukan *posting* bagi *user*
Prosedur Uji	1. Aplikasi dijalankan pada halaman *home.* 2. *User* memilih menu add. 3. *User* memasukan data *posting* pada *form posting.* 4. *User* menekan tombol Post.
Hasil yang diharapkan	Aplikasi dapat melakukan penambahan data *posting* pada *timeline.*

Sumber: [Pengujian]

Tabel 6.22 Kasus uji untuk pengujian validasi melihat *timeline*

Nama Kasus Uji	Kasus Uji Melihat Timeline
Objek Uji	SRS-002-02
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk melihat *timeline* dari posting yang dilakukan *user*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman *home.* 2. *User* melihat *timeline posting*.
Hasil yang diharapkan	Aplikasi dapat menampilkan data *posting* pada *timeline.*

Sumber: [Pengujian]

Tabel 6.23 Kasus uji untuk pengujian validasi menghapus *posting*

Nama Kasus Uji	Kasus Uji Menghapus Posting
Objek Uji	SRS-002-03
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas bagi *user* untuk menghapus *posting* miliknya.

Prosedur Uji	1. Aplikasi dijalankan pada halaman *home atau profile.* 2. *User* memilih *posting* yang akan dihapus pada *timeline*. 3. *User* memilih context menu delete. 4. Aplikasi menampilkan konfirmasi untuk menghapus *posting*. 5. *User* menekan tombol OK
Hasil yang diharapkan	Aplikasi dapat melakukan penghapusan data *posting* dan menampilkan notifikasi bahwa proses berhasil.

Sumber: [Pengujian]

Tabel 6.24 Kasus uji untuk pengujian validasi melihat lokasi

Nama Kasus Uji	Kasus Uji Melihat Lokasi
Objek Uji	SRS-003-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk melihat informasi lokasi dari *user* dalam bentuk peta (*map*).
Prosedur Uji	1. *User* mengakses halaman *map.* 2. Aplikasi menampilkan peta dengan *marker* sebagai penanda lokasi dari *user*.
Hasil yang diharapkan	Aplikasi dapat menampilkan peta yang berisi informasi dari lokasi *user* lain.

Sumber: [Pengujian]

Tabel 6.25 Kasus uji untuk pengujian validasi melihat jalur

Nama Kasus Uji	Kasus Uji Melihat Jalur
Objek Uji	SRS-003-02
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan komponen di dalam peta (*map*) untuk melihat jalur yang harus dilaluioleh seorang *user* menuju lokasi *user* lain.

Prosedur Uji	1. *User* mengakses halaman *map*. 2. Aplikasi menampilkan peta dengan *marker* sebagai penanda lokasi dari *user*. 3. *User* menekan tombol jalur.
Hasil yang diharapkan	Aplikasi dapat menampilkan jalur yang dapat dilalui untuk menuju lokasi *user* lain pada peta.

Sumber: [Pengujian]

Tabel 6.26 Kasus uji untuk pengujian validasi mencari teman

Nama Kasus Uji	Kasus Uji Mencari Teman
Objek Uji	SRS-004-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk mencari *user* berdasarkan nama *user*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman *search.* 2. *User* memasukan *username* pada *EditText*. 3. *User* menekan tombol cari.
Hasil yang diharapkan	Aplikasi dapat menampilkan daftar *user* yang sesuai dengan parameter *username* masukan. Jika tidak ditemukan maka aplikasi akan menampilkan notifikasi.

Sumber: [Pengujian]

Tabel 6.27 Kasus uji untuk pengujian validasi melihat info

Nama Kasus Uji	Kasus Uji Melihat Info
Objek Uji	SRS-004-02
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk melihat info dari seorang *user*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman *profile teman.* 2. *User* memenekan *keypad* menu pada *smartphone*. 3. *User* menekan menu info pada aplikasi.

Hasil yang diharapkan	Aplikasi dapat menampilkan informasi dari *user* yang sesuai.

Sumber: [Pengujian]

Tabel 6.28 Kasus uji untuk pengujian validasi menambah teman

Nama Kasus Uji	Kasus Uji Menambah Teman
Objek Uji	SRS-004-03
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk menambah *user* lain sebagai teman
Prosedur Uji	1. Aplikasi dijalankan pada halaman *profile user lain.* 2. *User* menekan tomol "+" untuk menambah sebagai teman.
Hasil yang diharapkan	Aplikasi dapat menambahkan data pertemanan ke dalam *database* dan menampilkan notifikasi saat proses berhasil dijalankan.

Sumber: [Pengujian]

Tabel 6.29 Kasus uji untuk pengujian validasi menghapus teman

Nama Kasus Uji	Kasus Uji Menghapus Teman
Objek Uji	SRS-004-04
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk menghapus teman
Prosedur Uji	1. Aplikasi dijalankan pada halaman *profile teman.* 2. *User* menekan tomol "-" untuk memenghapus pertemanan. 3. Aplikasi menampilkan konfirmasi penghapusan. 4. *User* menekan tombol OK untuk menghapus teman.

Hasil yang diharapkan	Aplikasi dapat menghapus data pertemanan dari *database* dan menampilkan notifikasi saat proses berhasil dijalankan.

Sumber: [Pengujian]

Tabel 6.30 Kasus uji untuk pengujian validasi *edit account*

Nama Kasus Uji	Kasus Uji *Edit Account*
Objek Uji	SRS-005-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk *user* dapat mengganti data pribadinya seperti biodata dan *password*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman *setting*. 2. *User* mengubah data yang diinginkan pada *form setting*. 3. *User* menekan tombol *Save*.
Hasil yang diharapkan	Aplikasi dapat meng-*update* data *user* dari *database* dan menampilkan notifikasi saat proses berhasil dijalankan.

Sumber: [Pengujian]

Tabel 6.31 Kasus uji untuk pengujian validasi mencari *user*

Nama Kasus Uji	Kasus Uji Mencari *User*
Objek Uji	SRS-006-01
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas untuk mencari *user* berdasarkan *userid*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman admin. 2. Admin memasukan *userid* pada *EditText*. 3. Admin menekan tombol cari.
Hasil yang diharapkan	Aplikasi dapat menampilkan daftar *user* yang sesuai dengan parameter *userid* masukan. Jika tidak ditemukan maka aplikasi akan menampilkan notifikasi.

Sumber: [Pengujian]

Tabel 6.32 Kasus uji untuk pengujian validasi *reset password*

Nama Kasus Uji	Kasus Uji *Reset Password*
Objek Uji	SRS-006-02
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas bagi admin untuk mengeset ulang (*reset*) *password* pengguna.
Prosedur Uji	1. Aplikasi dijalankan pada halaman admin. 2. Admin telah melakukan pencarian *user*. 3. Admin memilih *user* pada list *user* yang ditemukan. 4. Admin menekan *context menu reset password*. 5. Aplikasi menampilkan *form* untuk *reset password* pengguna. 6. Admin mengisi *password* baru pada *form* yang disediakan. 7. Admin menekan tombol *reset.*
Hasil yang diharapkan	Aplikasi dapat me-*reset password* pengguna dan menampilkan notifikasi saat proses berhasil.

Sumber: [Pengujian]

Tabel 6.33 Kasus uji untuk pengujian validasi hapus *user*

Nama Kasus Uji	Kasus Uji Hapus *User*
Objek Uji	SRS-006-04
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa aplikasi dapat memenuhi kebutuhan fungsional dalam menyediakan fasilitas bagi admin untuk menghapus *user*.
Prosedur Uji	1. Aplikasi dijalankan pada halaman admin. 2. Admin telah melakukan pencarian *user*. 3. Admin memilih *user* pada *list user* yang ditemukan. 4. Admin menekan *context menu Delete user* 5. Aplikasi menampilkan konfirmasi penghapusan. 6. Admin menekan tombol *OK*.
Hasil yang diharapkan	Aplikasi dapat menghapus data *user* dari *database* dan menampilkan notifikasi saat proses berhasil.

Sumber: [Pengujian]

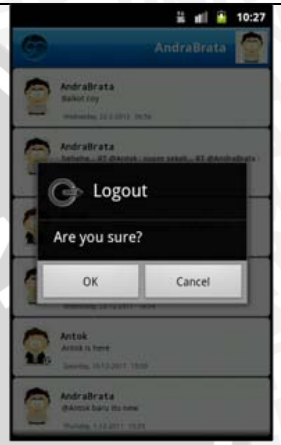
6.1.3.2 Hasil Pengujian Validasi

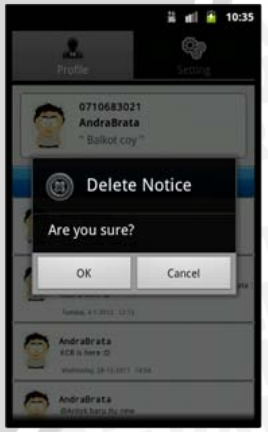
Dari kasus uji yang telah dilaksanakan sesuai dengan prosedur pengujian pada sub bab sebelumnya, maka didapatkan hasil seperti ditunjukan pada Tabel 6.34.

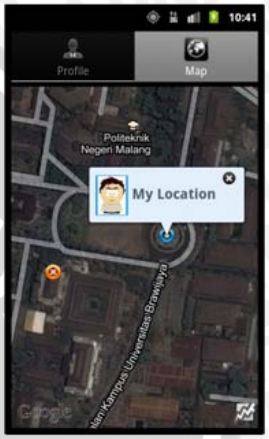
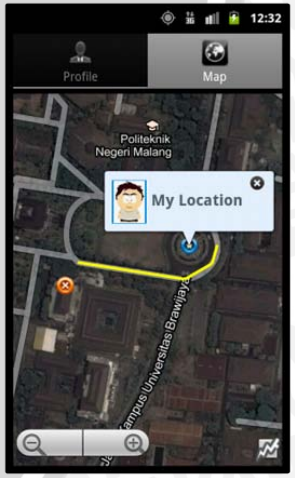
Tabel 6.34 Hasil pengujian validasi

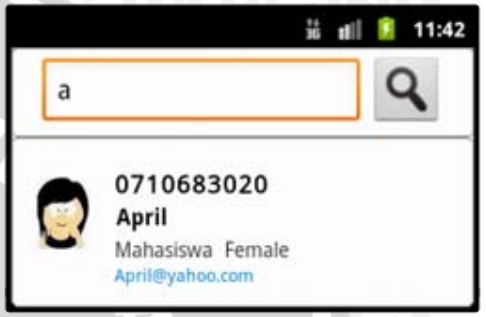
No.	Nama Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan	Tampilan Validitas
1.	*Register*	Aplikasi memeriksa data masukan, jika data yang dimasukan valid maka data akan ditambahkan ke *database* dan menampilkan notifikasi bahwa *user* baru sudah terdaftar. Jika data yang diisikan tidak valid atau masih ada *form* yang belum diisi, aplikasi akan menampilkan pesan kesalahan.	Aplikasi memeriksa data masukan, jika data yang dimasukan valid maka data akan ditambahkan ke *database* dan menampilkan notifikasi bahwa *user* baru sudah terdaftar. Jika data yang diisikan tidak valid atau masih ada *form* yang belum diisi, aplikasi akan menampilkan pesan kesalahan.	 Valid

2.	*Login* Sah	Aplikasi dapat melakukan penyeleksian kondisi *login* pada *database* berdasar data yang dimasukkan dan jika penyeleksian kondisi ini benar, maka akan masuk ke sistem sesuai dengan hak aksesnya.	Aplikasi dapat melakukan penyeleksian kondisi *login* pada *database* berdasar data yang dimasukkan dan jika penyeleksian kondisi ini benar, maka akan masuk ke sistem sesuai dengan hak aksesnya.	 Valid
3.	*Login* Tidak Sah	Aplikasi dapat melakukan penyeleksian kondisi *login* pada *database* berdasar data yang dimasukkan dan jika penyeleksian kondisi ini salah, maka tidak akan masuk ke sistem dan aplikasi menampilkan pesan kesalahan.	Aplikasi dapat melakukan penyeleksian kondisi *login* pada *database* berdasar data yang dimasukkan dan jika penyeleksian kondisi ini salah, maka tidak akan masuk ke sistem dan aplikasi menampilkan pesan kesalahan.	 Valid

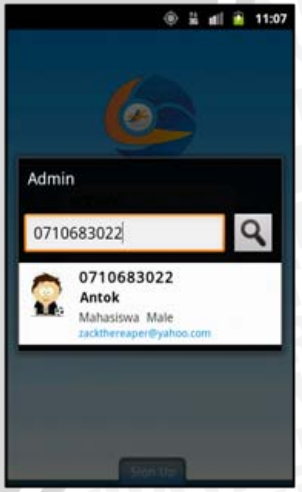
4.	*Logout*	Aplikasi melakukan penghapusan session yang aktif dan menampilkan halaman *login*.	Aplikasi melakukan penghapusan session yang aktif dan menampilkan halaman *login*.	 Valid
5.	Melakukan *Posting*	Aplikasi dapat melakukan penambahan data *posting* pada *timeline*.	Aplikasi dapat melakukan penambahan data *posting* pada *timeline*.	 Valid

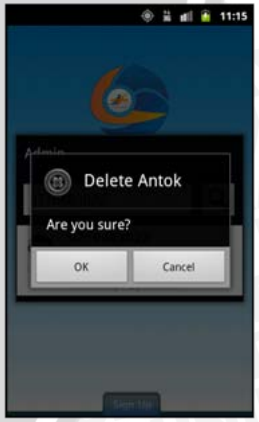
6.	Melihat *Timeline*	Aplikasi dapat menampilkan data *posting* pada *timeline.*	Aplikasi dapat menampilkan data *posting* pada *timeline.*	 Valid
7.	Menghapus *Posting*	Aplikasi dapat melakukan penghapusan data *posting* dan menampilkan notifikasi bahwa proses berhasil.	Aplikasi dapat melakukan penghapusan data *posting* dan menampilkan notifikasi bahwa proses berhasil.	 Valid

8.	Melihat Lokasi	Aplikasi dapat menampilkan peta yang berisi informasi dari lokasi *user* lain.	Aplikasi dapat menampilkan peta yang berisi informasi dari lokasi *user* lain.	 Valid
9.	Melihat Jalur	Aplikasi dapat menampilkan jalur yang dapat dilalui untuk menuju lokasi *user* lain pada peta.	Aplikasi dapat menampilkan jalur yang dapat dilalui untuk menuju lokasi *user* lain pada peta.	 Valid

10.	Mencari Teman	Aplikasi dapat menampilkan daftar *user* yang sesuai dengan parameter *username* masukan. Jika tidak ditemukan maka aplikasi akan menampilkan notifikasi.	Aplikasi dapat menampilkan daftar *user* yang sesuai dengan parameter *username* masukan. Jika tidak ditemukan maka aplikasi akan menampilkan notifikasi.	 Valid
11.	Melihat Info	Aplikasi dapat menampilkan informasi dari *user* yang sesuai.	Aplikasi dapat menampilkan informasi dari *user* yang sesuai.	 Valid

12.	Menambah Teman	Aplikasi dapat menambahkan data pertemanan ke dalam *database* dan menampilkan notifikasi saat proses berhasil dijalankan.	Aplikasi dapat menambahkan data pertemanan ke dalam *database* dan menampilkan notifikasi saat proses berhasil dijalankan.	 Valid
13.	Menghapus Teman	Aplikasi dapat menghapus data pertemanan dari *database* dan menampilkan notifikasi saat proses berhasil dijalankan.	Aplikasi dapat menghapus data pertemanan dari *database* dan menampilkan notifikasi saat proses berhasil dijalankan.	 Valid

14.	*Edit Account*	Aplikasi dapat meng-*update* data *user* dari *database* dan menampilkan notifikasi saat proses berhasil dijalankan.	Aplikasi dapat meng-*update* data *user* dari *database* dan menampilkan notifikasi saat proses berhasil dijalankan.	 Valid
15.	Mencari *User*	Aplikasi dapat menampilkan daftar *user* yang sesuai dengan parameter *userid* saat aplikasi berada dalam menu admin. Jika tidak ditemukan maka aplikasi akan menampilkan notifikasi.	Aplikasi dapat menampilkan daftar *user* yang sesuai dengan parameter *userid* saat aplikasi berada dalam menu admin. Jika tidak ditemukan maka aplikasi akan menampilkan notifikasi.	 Valid

16.	*Reset Password*	Aplikasi dapat me-*reset password* pengguna saat aplikasi berada dalam menu admin dan menampilkan notifikasi saat proses berhasil.	Aplikasi dapat me-*reset password* pengguna saat aplikasi berada dalam menu admin dan menampilkan notifikasi saat proses berhasil.	 Valid
17.	Hapus *User*	Aplikasi dapat menghapus data *user* dari *database* saat aplikasi berada dalam menu admin dan menampilkan notifikasi saat proses berhasil.	Aplikasi dapat menghapus data *user* dari *database* saat aplikasi berada dalam menu admin dan menampilkan notifikasi saat proses berhasil.	 Valid

Sumber: [Pengujian]

6.1.4 Pengujian Performa

Pengujian performa dilakukan untuk mengetahui bagaimana performa *server* dari aplikasi jejaring sosial dalam mengatasi banyaknya *request* dalam satuan waktu yang di lakukan secara bersamaan dan dalam jumlah tertentu. Pengujian performa dilakukan menggunakan *tool* yang disebut apache *benchmark.*

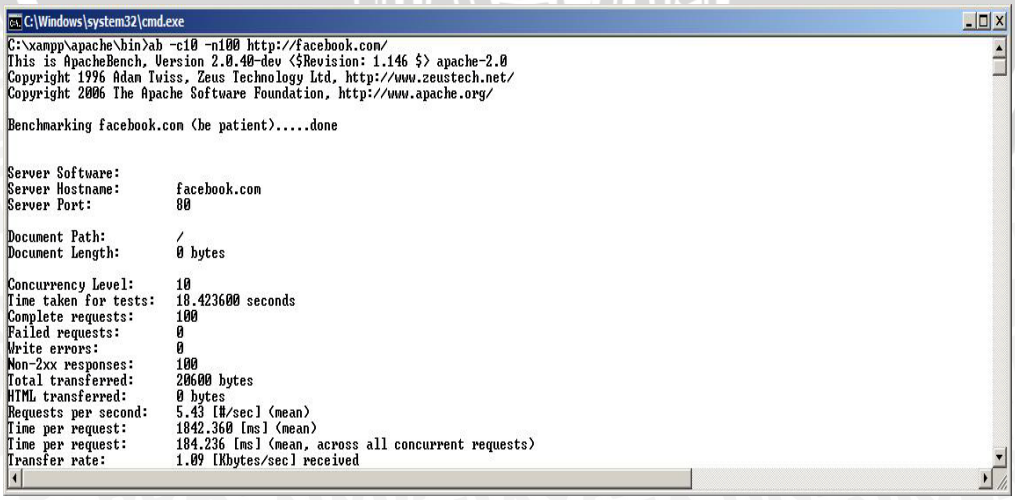
Apache *benchmark* adalah *tool* untuk mengukur performa dari apache *server* dalam melayani *request* dari *client*. Adapun sintak yang digunakan dalam apache *benchmark* ditunjukan pada Gambar 6.1.

```
ab -c 10 -n 100 [url]
```

Gambar 6.1 Sintak apache *benchmark*

Sumber : [APA-12]

Sintak pada Gambar 6.1 memiliki beberapa parameter antara lain `ab`, `-c`, `-n`, dan `url`. Parameter `ab` merupakan singkatan dari *Apache Benchmark*. Parameter `-c` 10 adalah jumlah *concurrent connection*, sebanyak 10 koneksi secara bersamaan yang dibuat dalam satu waktu. Parameter `-n` 100 adalah jumlah *request* yang dibuat ke server tujuan, sebanyak 100 *request.* Parameter terakhir adalah `url`, alamat halaman yang akan diproses oleh apache *benchmark*. Gambar 6.2 menunjukkan antarmuka apache *benchmark* saat dijalankan.



Gambar 6.2 Tampilan apache *benchmark* saat dijalankan

Sumber : [Pengujian]

Pengujian dilakukan dalam dua tahap yaitu pengujian *request per second* dan pengujian jumlah maksimal *request*. Spesifikasi *hadware* komputer yang digunakan sebagai *server* dalam jaringan *intranet* dalam pengujian ini dijelaskan pada Tabel 6.35.

Tabel 6.35 Spesifikasi perangkat keras *server* dalam percobaan

Nama Komponen	Spesifikasi
Prosesor	Intel® Core™ 2 Duo CPU P8400 @ 2.26*GHz*
Memory (RAM)	4 *GB*
Hardisk	Toshiba MK1652GSX, kapasitas 160 *GB*, 5400 *rpm*
Mother Board	Toshiba Satellite Pro U400
Kartu Grafis	Mobile Intel® GMA 4500MHD

Sumber : [Pengujian]

6.1.4.1 Pengujian *Request per Second*

Pengujian *request per second* dilakukan pada jaringan yang dimiliki oleh Teknik Informatika Universitas Brawijaya melalui dua skenario, yaitu pengujian performa *server* yang berada pada jaringan *intranet* dan pada *server* yang di-*hosting* secara *online*. Dalam pengujian ini objek yang digunakan adalah *file web service* gettimeline.php yang ada pada *server* dengan alamat IP 172.21.4.222 dan pada *server online* dengan alamat soscamp.malangku.net, sehingga nanti akan diketahui *throughput* dari *request* ke *web service* tersebut. Percobaan dilakukan dengan menggunakan jumlah *request* yang tetap namun dengan jumlah *concurrency* yang berbeda.

Pada pengujian yang pertama, percobaan dilakukan dengan menggunakan 100 *request* dengan jumlah *concurrency* yang berbeda yaitu mulai dari 5 hingga 80 dengan interval 2n pada lingkungan jaringan Teknik Informatika Universitas Brawijaya. Tabel 6.36 menunjukkan hasil pengujian apache *benchmark* dengan 100 *request* pada jaringan *intranet*.

Tabel 6.36 Hasil percobaan apache *benchmark* dengan 100 *request pada intranet*

Request	*Concurrency*	*Request / Second*	*Transfer Rate*
100	5	110,38	160,04 KBps

100	10	112,87	163,66 KBps
100	20	120,34	174,49 KBps
100	40	128,45	187,88 KBps
100	80	140,06	203,08 KBps
Rata-rata		122,42	177,83 KBps

Sumber: [Pengujian]

Berdasarkan kasus uji pertama pada Tabel 6.36 didapatkan hasil sebagai berikut :

a. Nilai *transfer rate* minimum sebesar 160,04 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 110,38 *request*.
b. Nilai *transfer rate* maksimum sebesar 203,08 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 140,06 *request*.
c. Dari kelima percobaan yang dilakukan didapatkan nilai *transfer rate* rata – rata adalah sebesar 177,83 KBps dengan dengan *request per second* rata-rata sebesar 122,42 *request*.

Pengujian dilanjutkan dengan mengukur performa *server* yang ada pada *hosting online*, percobaan dilakukan dengan menggunakan 100 *request* dengan jumlah *concurrency* yang berbeda-beda. Tabel 6.37 menunjukkan hasil pengujian apache *benchmark* dengan 100 *request* pada jaringan *Internet*.

Tabel 6.37 Hasil percobaan apache *benchmark* dengan 100 *request pada Internet*

Request	*Concurrency*	*Request / Second*	*Transfer Rate*
100	5	5,13	1,44 KBps
100	10	10,77	3,02 KBps
100	20	22,84	6,39 KBps
100	40	35,52	9,95 KBps
100	80	47,19	13,21 KBps
Rata-rata		24,29	6,8 KBps

Sumber: [Pengujian]

Berdasarkan kasus uji pada Tabel 6.37 didapatkan hasil sebagai berikut :

a. Nilai *transfer rate* minimum sebesar 1,44 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 5,13 *request*.
b. Nilai *transfer rate* maksimum sebesar 13,21 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 47,19 *request*.
c. Dari kelima percobaan yang dilakukan didapatkan nilai *transfer rate* rata – rata adalah sebesar 6,8 KBps dengan dengan *request per second* rata-rata sebesar 24,29 *request*.

Kasus uji selanjutnya adalah dengan menggunakan jumlah *request* yang lebih banyak yaitu 200 *request* dengan jumlah *concurrency* yang berbeda yaitu mulai dari 5 hingga 80 dengan interval 2n. Tabel 6.38 menunjukkan hasil pengujian *apache benchmark* dengan 200 *request* pada jaringan *intranet*.

Tabel 6.38 Hasil percobaan apache *benchmark* dengan 200 *request* pada *intranet*

Request	*Concurrency*	*Request / Second*	*Transfer Rate*
200	5	116,14	168,41 KBps
200	10	123,76	179,46 KBps
200	20	130,29	188,93 KBps
200	40	139,28	201,95 KBps
200	80	144,82	209,99 KBps
Rata-rata		130,86	189,75 KBps

Sumber: [Pengujian]

Berdasarkan kasus uji kedua pada Tabel 6.38 didapatkan hasil sebagai berikut :

a. Nilai *transfer rate* minimum sebesar 168,41 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 116,14 *request*.
b. Nilai *transfer rate* maksimum sebesar 209,99 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 144,82 *request*.

c. Dari kelima percobaan yang dilakukan, didapatkan nilai *transfer rate* rata – rata adalah sebesar 189,75 KBps dengan dengan *request per second* rata-rata sebesar 130,86 *request*.

Pengujian dilanjutkan dengan mengukur performa *server* yang ada pada *hosting online*, percobaan dilakukan dengan menggunakan 200 *request* dengan jumlah *concurrency* yang berbeda-beda. Tabel 6.39 menunjukkan hasil pengujian apache *benchmark* dengan 200 *request* pada jaringan *Internet*.

Tabel 6.39 Hasil percobaan apache *benchmark* dengan 200 *request* pada *Internet*

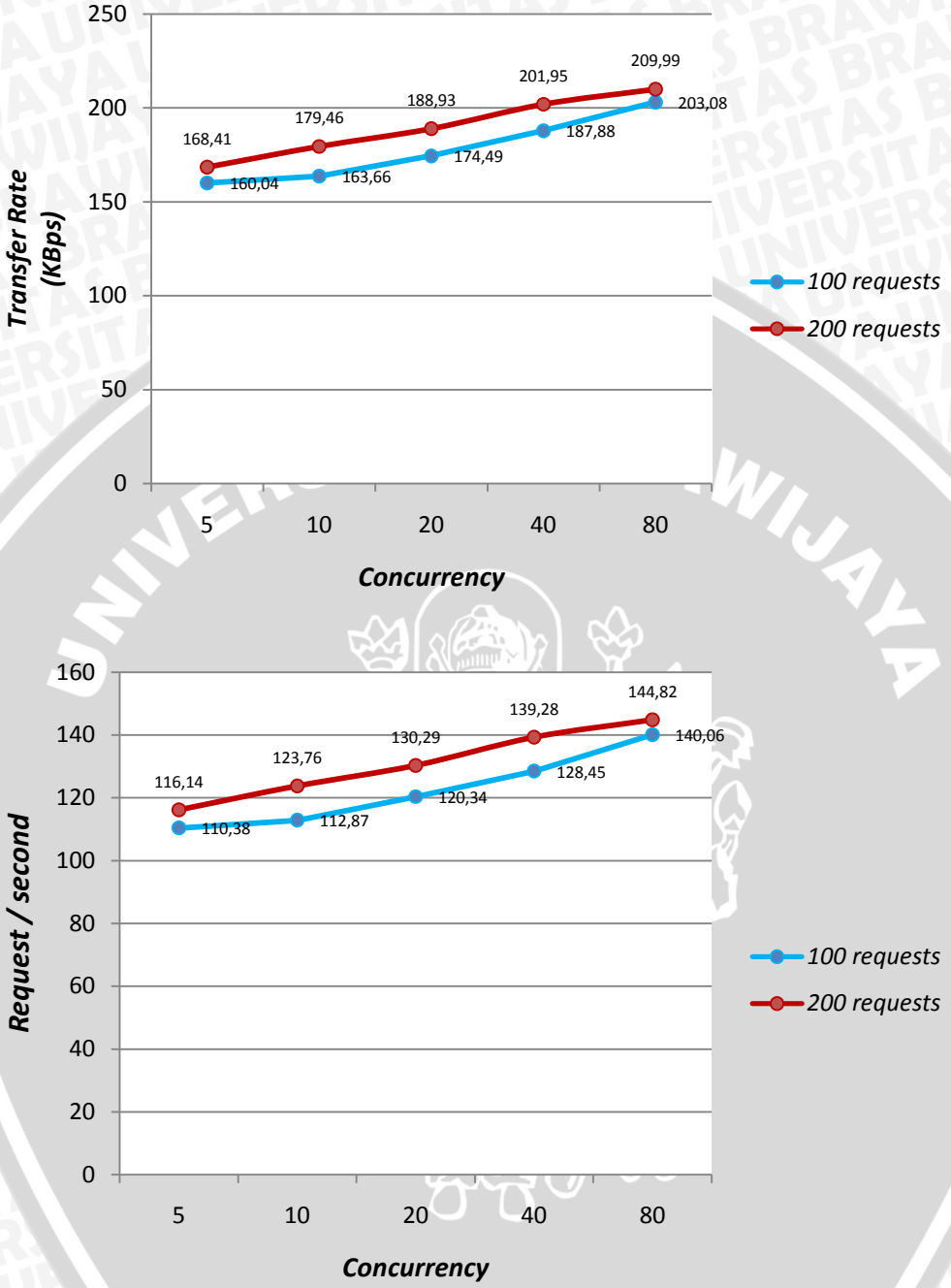
Request	*Concurrency*	*Request / Second*	*Transfer Rate*
200	5	7,29	2,08 KBps
200	10	14,46	4,12 KBps
200	20	28,37	8,09 KBps
200	40	47,63	13,57 KBps
200	80	69,93	19,93 KBps
Rata-rata		33,54	9,56 KBps

Sumber: [Pengujian]

Berdasarkan kasus uji pada Tabel 6.39 didapatkan hasil sebagai berikut :

a. Nilai *transfer rate* minimum sebesar 2,08 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 7,29 *request*.
b. Nilai *transfer rate* maksimum sebesar 19,93 KBps dengan jumlah *request per second* yang dapat dilayani *server* sebesar 69,93 *request*.
c. Dari kelima percobaan yang dilakukan, didapatkan nilai *transfer rate* rata – rata adalah sebesar 9,56 KBps dengan dengan *request per second* rata-rata sebesar 33,54 *request*.

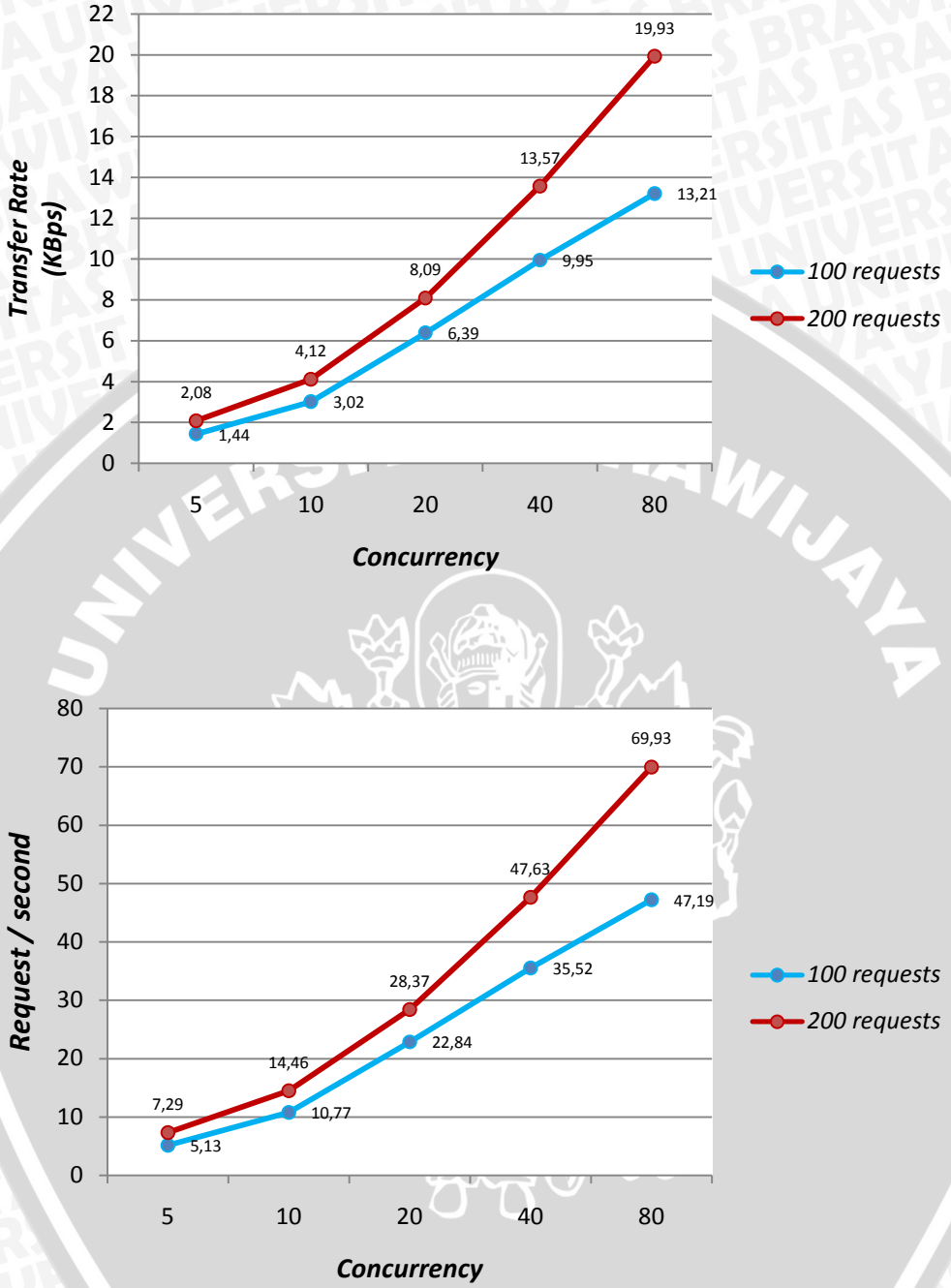
Dari percobaan yang dilakukan dalam lingkungan jaringan Teknik Informatika Universitas Brawijaya, hasil yang didapatkan dapat dideskripsikan ke dalam grafik yang menunjukkan hubungan antara *concurrency* dengan *transfer rate* dan *concurrency* dengan *request per second*. Gambar 6.3 menunjukkan grafik performa *server* pada jaringan *intranet*.



Gambar 6.3 Grafik hasil pengujian pada jaringan *intranet*

Sumber : [Pengujian]

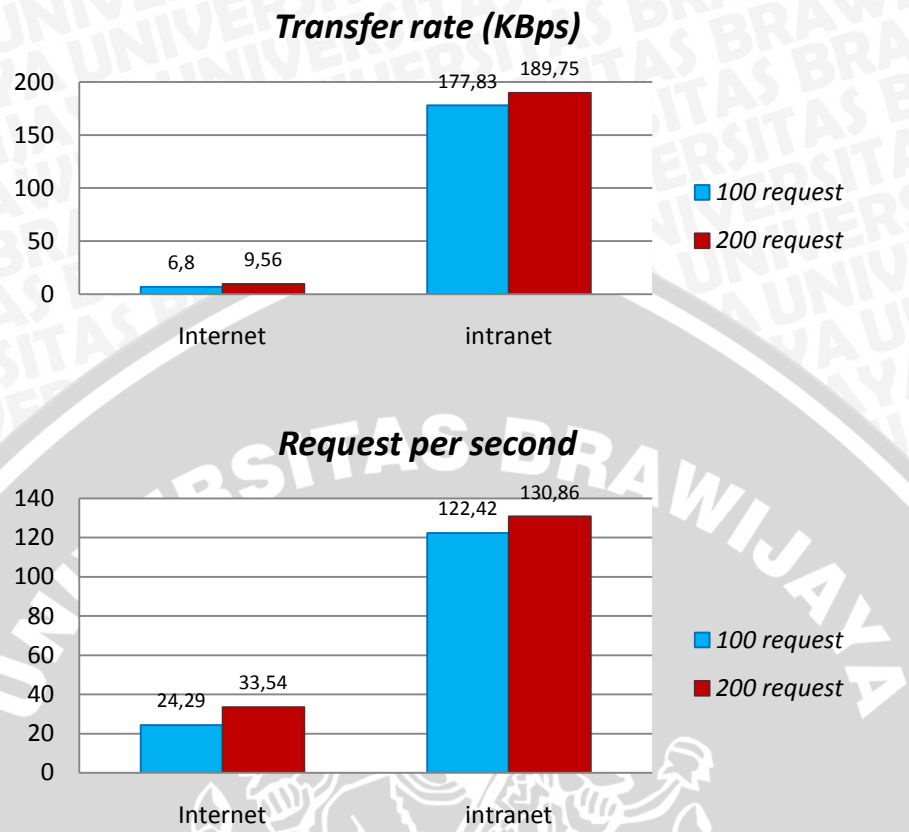
Grafik dari hasil pengujian yang dilakukan pada *server* yang di *hosting* secara *online* dan diakses dari jaringan *Internet* Teknik Informatika Universitas Brawijaya dapat dilihat pada Gambar 6.4.



Gambar 6.4 Grafik hasil pengujian pada jaringan *Internet*

Sumber : [Pengujian]

Diagram dari rata-rata performa *server* pada jaringan *intranet* dan *Internet* dapat dilihat pada Gambar 6.5.



Gambar 6.5 Diagram rata-rata hasil pengujian

Sumber : [Pengujian]

Dari hasil pengujian dapat ditarik kesimpulan bahwa nilai *transfer rate* dan jumlah *request per second* tidak linier terhadap banyaknya koneksi (*concurrency*) karena tidak memenuhi persamaan linier y = ax+b. Ketidaklinieran tersebut disebabkan oleh *transfer rate* yang berbeda untuk jumlah koneksi yang berbeda dalam suatu jaringan, sehingga mempengaruhi jumlah *request per second* yang dapat dilayani oleh *server*. Semakin tinggi nilai *transfer rate* maka *request* yang dapat dilayani oleh *server* dalam satu detik juga akan semakin banyak. *Transfer rate* sendiri selain dipengaruhi oleh jumlah *request* yang dikirimkan, juga dipengaruhi oleh *traffic* jaringan yang digunakan. Perbedaan performa *server* yang berada pada jaringan *intranet* dan jaringan *Internet* cukup signifikan, hal ini disebabkan oleh jumlah *router* yang dilalui menuju ke alamat *server* pada *hosting online* lebih banyak daripada *server* dalam jaringan *intranet*.

6.1.4.2 Pengujian Jumlah Maksimum *Request*

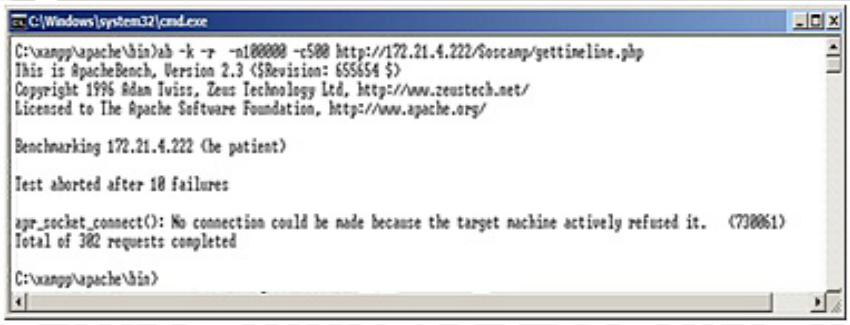
Pengujian performa *server* dilanjutkan untuk mengetahui jumlah *request* maksimal yang dapat dilayani oleh *server* yang terdapat pada jaringan *intranet* Teknik Informatika Universitas Brawijaya dengan alamat IP 172.21.4.222. Pengujian dilakukan dengan memberikan *apache benchmark* nilai *concurrency* yang tetap yaitu 500 koneksi secara bersamaan dan jumlah *request* yang berbeda yaitu mulai dari 10.000 kemudian ditingkatkan dengan interval 10.000 hingga *server* tidak mampu lagi melayani *request* dari *client*. Tabel 6.40 menunjukkan hasil dari pengujian jumlah *request* maksimum yang dapat dilayani *server*.

Tabel 6.40 Hasil pengujian jumlah *request* maksimum *server intranet*

Request	*Concurrency*	Hasil Pengujian
10000	500	Dapat dilayani
20000	500	Dapat dilayani
30000	500	Dapat dilayani
40000	500	Dapat dilayani
50000	500	Dapat dilayani
60000	500	Dapat dilayani
70000	500	Dapat dilayani
80000	500	Dapat dilayani
90000	500	Dapat dilayani
100000	500	Tidak dapat dilayani

Sumber : [Pengujian]

Hasil pengujian menunjukkan bahwa *server* mengalami kegagalan dalam melayani *request client* saat *concurrency* bernilai 500 dan jumlah *request* mencapai nilai 100.000 *request*. Tampilan apache *benchmark* saat *server* gagal melayani *request* dari *client* dapat dilihat pada Gambar 6.6.



Gambar 6.6 Tampilan apache *benchmark* saat *server* gagal melayani *request*

Sumber : [Pengujian]

BAB VII
PENUTUP

7.1. Kesimpulan

Berdasarkan hasil perancangan, implementasi dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut :

1. Aplikasi jejaring sosial pada *smartphone* android yang memiliki layanan *autogeotagging* sesuai dengan perancangan telah dibuat dan dapat digunakan sebagai salah satu media hubungan sosial antara sesama mahasiswa maupun dosen pada Teknik Informatika Universitas Brawijaya.
2. Koneksi data antara *web server* dan aplikasi pada *smartphone* android telah berhasil diimplementasikan dengan metode pengiriman data dari *web server* ke aplikasi *mobile* di android menggunakan JSON.
3. Hasil pengujian nilai *throughput* data pada *server* yang dilakukan menggunakan *apache benchmark* dengan 100 *request* pada kasus uji pertama dan 200 *request* pada kasus uji kedua serta jumlah *concurrency* yang berbeda menunjukkan bahwa nilai *transfer rate* dan jumlah *request per second* tidak linier terhadap banyaknya koneksi (*concurrency*). Ketidaklinieran tersebut disebabkan oleh *transfer rate* yang berbeda untuk jumlah koneksi yang berbeda sehingga mempengaruhi jumlah *request per second* yang dapat dilayani oleh *server*.
4. Pengujian performa dengan 100 *request* yang dikirimkan pada *server* menunjukan nilai rata-rata *transfer rate* 6,8 KBps dengan *request per second* sebesar 24,29 *request* pada jaringan *Internet* dan *transfer rate* pada jaringan *intranet* sebesar 177,83 KBps dengan *request per second* 122,42. Pada pengujian performa dengan 200 *request* diperoleh nilai *transfer rate* rata-rata sebesar 9,56 KBps dengan *request per second* sebesar 33,54 pada jaringan *Internet* dan nilai *transfer rate* pada jaringan *intranet* sebesar 189,75 KBps dengan *request per second* 130,86 *request*.

5. Semakin tinggi *transfer rate* maka *request* yang dapat dilayani oleh *server* dalam satu detik juga akan semakin banyak.
6. *Transfer rate* selain dipengaruhi oleh jumlah *request* yang dikirimkan, juga dipengaruhi oleh *traffic* jaringan yang digunakan.
7. Pengujian jumlah maksimum *request* menunjukkan bahwa jumlah *request* maksimum yang dapat dilayani dengan 500 koneksi secara bersamaan adalah 100.000 *request*.

7.2. Saran

Saran yang dapat diberikan untuk pengembangan aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android ini antara lain adalah :

1. Untuk pengembangan lebih lanjut, aplikasi ini dapat dikembangkan dengan membuat *client* berbasis *web* dengan *Java Server Pages* (JSP).
2. Informasi untuk *user* pada aplikasi ini masih sederhana, sehingga penambahan data *user* seperti tanggal lahir, nomor *hand phone*, alamat url dan status pertemanan dapat memperkaya informasi yang dimiliki.
3. Aplikasi ini dapat dikembangkan lebih lanjut dengan menambahkan fitur-fitur jejaring sosial yang lebih memudahkan pengguna dalam berinteraksi satu sama lain seperti layanan untuk *private message* antar pengguna, menu pencarian dengan parameter yang lebih beragam, layanan untuk notifikasi via email dan layanan untuk menampilkan notifikasi jika ada teman yang lokasinya dekat dengan kita.
4. *Web service* yang terdapat pada sistem mendukung pengembangan aplikasi jejaring sosial ini pada semua jenis *platform smartphone* seperti Blackberry OS dan iOS sehingga *client* aplikasi ini tidak terbatas pada *platform* android saja.
5. Aplikasi ini dapat dikembangkan untuk penggunaan secara global tidak hanya sebatas area kampus.

DAFTAR PUSTAKA

[ABL-11] Ableson, Frank W; Robi Sen and Chris King. 2011. *Android in Action*. Manning.

[AND-11] Android Developer Official Site. 2011. *Android Development Guide.* <http://developer.android.com/ > diakses tanggal : 17 Juni 2011.

[APA-11] Apache Official Site. 2012. *Benchmarking with apache benchmark.* <http://ab.apache.org/ > diakses tanggal : 21 April 2012.

[BUR-10] Burnette, Ed. 2010. *Hello, Android : Introducting Google's Mobile Development Platform, Third Edition*. The Pracmatic Bookself.

[BOY-08] Boyd, M. Danah and Nicole B. Ellison. 2008. *Social Network Sites : Definition, History, and Scholarship*. Journal of Computer-Mediated Communication 13, No.210-230

[CTI-11] Cellular Telecommunications Industry Association. 2011. *Glossary*. <http://www.ctia.org/content/index.cfm/AID/10409> diakses tanggal : 13 Juni 2011

[DAC-05] Daconta, M.C.; Obrst L.J.; and Smith, K.T. 2005. *The Semantic Web: A Guide to the Future of XML,Web Services, and Knowledge Management*. John Willey.

[DHA-06] Dharwiyanti, Sri. 2006. *Pengantar Unified Modeling Language (UML)*. IlmuKomputer.Com

[DIG-09] Diggelen, Frank van. 2009. *A-GPS: Assisted GPS, GNSS, and SBAS*. Artech House.

[ELR-06] El-Rabbany, Ahmed. 2006.*Introduction to GPS: The Global Positioning System, second edition*. Norwood: Artech House.

[JSO-11] Json.org. 2011. *Pengenalan JSON.*<http://json.org> diakses tanggal : 25 Juli 2011.

[KAP-06] Kaplan, Elliot and Christopher Hegarty. 2006. *Understanding GPS : Principles and Applications, Second Edition.* Norwood: Artech House , Inc.

[KUS-11] Kushwaha, Amit and Vineet Kushwaha. 2011. *Location Based Services using Android Mobile Operating System*. International Journal of Advances in Engineering and Technology, Mar 2011. Vol.1 ISSN:2231-1963.

[LAN-08] Langer, Arthur M. 2008. *Analysis and Design of Information Systems: Third Edition*. Springer.

[LAR-11] Larson, Martha; Soleymani, Mohammad; and Serdyukov, Pavel. 2011. *Automatic Tagging and Geotagging in Video Collections and Communities*. International Conference on Multimedia Retrieval, ICMR, Apr 2011 Trento, Italy. ACM. ISBN 978-1-4503-0336-1.

[MCN-04] McNamara, Joel. 2004. *GPS For Dummies*®.Wiley Publishing, Inc.

[MEI-10] Meier, Reto. 2010. *Profesional Android 2 Application Development, Second Edition*. Wiley Publishing , Inc.

[MIL-09] Miller, Frederic P; Agnes F Vandome; John McBrewster. 2009. *Geotagging*. VDM Publishing House Ltd.

[MUR-10] Murphy, Mark L. 2010. *Begining Android 2*. Apress.

[MUR-11] Murphy, Mark L. 2011. *Begining Android 3*. Apress.

[PET-12] Peter dan Joe. 2012. *National Marine Electronics Association (NMEA) Data*. <http://www.gpsinformation.org/dale/nmea.htm> diakses tanggal : 20 Januari 2012.

[PHP-11] PHP team. 2011. *JSON Book Manual*. <http://php.net/manual> diakses tanggal : 27 Juli 2011.

[PRE-10] Pressman, Roger S. 2010. *Software Engineering: a practitioner's approach,7th edition*. Mc Graw Hill.

[RAN-12] Rani, Radhika; Praveen Kumar; Adarsh; Krishna Mohan; and K.V Kiran. 2012. *Location Based Service in Android*. International Journal of Advances in Engineering and Technology, Mar 2012. Vol.3 ISSN: 2231-1963.

[ROB-09] Robert, Albert R and Gilbert. 2009. *Buku Pintar Pekerja Sosial - Jilid 2 (Social workers' Desk References).* Oxford University Press, Inc.

[SMI-11] Smith, Dave and Jeff Friesen. 2011. *Android Recipes, A Problem-Solution Approach.* Apress

[SOM-11] Sommerville, Ian. 2011. *Software engineering, 9th edition.* Addison Wesley Publishers.

[W3C-11] W3C. 2011. *Web Services Tutorial*. <http://www.w3schools.com/webservices/default.asp> diakses tanggal : 11 Juli 2011.

[WIR-11] Wireless Week. 2011. *Google Looks to Dominate Mobile*. <www.wirelessweek.com/Article-Google-Dominate-Mobile.aspx> diakses tanggal :17 Maret 2011.

Lampiran 1. Sampel Data Dosen

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS BRAWIJAYA
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
Gedung A PTIIK Lt. 1
JL. Veteran No.8, Malang, 65145, Indonesia
Telp. : +62-341-577911; Fax : +62-341-577911
http://ptiik.ub.ac.id E-mail : ptiik@ub.ac.id

Nomor : 0736 /UN10.36/AK/2012
Hal : Permohonan data skripsi

Yth. Ketua Program Studi Teknik Informatika
Program Teknologi Informasi dan Ilmu Komputer
Universitas Brawijaya

Sebagai salah satu persyaratan untuk menyelesaikan studi di Program Teknologi Informasi dan Ilmu Komputer, mahasiswa harus menyusun skripsi. Untuk itu kami mohonkan ijin bagi mahasiswa kami :

Nama : Komang Candra Brata
NIM : 0710683021
Jurusan : Informatika

Guna memperoleh informasi permasalahan/data untuk skripsi mahasiswa tersebut, jenis data yang diperlukan dan rencana waktu pelaksanaan adalah :

Data : 1. Nama dan NIP/NIK Dosen PTIIK.
2. Mahasiswa Program Studi Teknik Informatika Angkatan 2007.

Waktu : 6 Juli s/d 13 Juli 2012

Atas bantuan serta perhatiannya kami ucapkan terima kasih.

a.n. Ketua PTIIK
Wakil Ketua I Bidang Akademik,

Ir. Heru Nurwarsito, M.Kom
NIP196504021990021001

Lampiran 1. Sampel Data Dosen (Lanjutan)

No	Nama	NIK
1	Ismiarta Aknuranda, ST,M.Sc.,Ph.D	740719 06 1 1 0079
2	Eko Sakti Pramukantoro, S.Kom	860805 06 1 1 0252
3	Indriati, ST	831013 06 1 2 0035
4	Aswin Suharsono, ST., MT	840919 06 1 1 0251
5	Welly Purnomo, ST	810117 06 1 1 0026
6	Eriq M Adams J, ST	850410 06 1 1 0027
7	Fariz Andri Bakhtiar, ST	850314 06 1 1 0028
8	Eko Satiawan, ST	870610 06 1 1 0256
9	Rekyan Regasari MP, ST., MT	770414 06 1 2 0253
10	Budi Darma Setiawan, S.Kom., M.Sc	841015 06 1 1 0090

Lampiran 2. Sampel Data Mahasiswa

NO	NIM	NAMA
1	0710683001	DETA PRATAMA AGUNG E
2	0710683002	RESTU ATMAJA R A
3	0710683003	DENISA NOVINDA A P
4	0710683004	HARFINDA AVRILIDA F
5	0710683005	RATIH KARTIKA DEWI
6	0710683006	DIMAS MUHAMMAD ISA
7	0710683007	SURYA WIRAWAN
8	0710683008	ANZA ANSORI
9	0710683009	BIAS AKIANOM
10	0710683010	MAHARDEKA TRI ANANTA
11	0710683011	WISNU ADITYA
12	0710683012	IRWAN HIKMAH BAYU P
13	0710683013	LUKMAN AL KAUTSAR I
14	0710683014	ALDIAN L MANUPUTTY
15	0710683015	FITRY W
16	0710683016	DONNY RAHMADIEN M P
17	0710683017	KUKUH HERU IRAWAN
18	0710683018	FAIZA ALIF FAKHRINA
19	0710683019	NOVIANA PUTRI P
20	0710683020	ARIANTI SILVIA
21	0710683021	KOMANG CANDRA BRATA
22	0710683022	FAIZAL DWI PRASETYO
23	0710683023	R WAHYU PARWITAYASA
24	0710683024	FEBRIAN ARIS PRADIKA
25	0710683025	PRAYOGA DWI YUNATA
26	0710683026	BUCE TRIAS HANGGARA
27	0710683027	ALDILA PRATIWI
28	0710683028	ADITYA CAHYA PUTRA
29	0710683029	LUTFI FANANI
30	0710683030	NYOMAN WIRA PRASETYA
31	0710683031	SATIVANDI PUTRA
32	0710683032	SINGGIH PRIMA A
33	0710683033	PRITA S A F
34	0710683034	FEDRICK ZULVASIUS
35	0710683035	TAMTOWIL MUSTOFA
36	0710683036	ADITYO NUGROHO
37	0710683037	BENNY SESARIO
38	0710683038	ARIF MUDJAHIDAH
39	0710683039	FARID ANGGA PRIBADI
40	0710683040	ADAM HENDRA BRATA
41	0710683041	RADITYA N
42	0710683042	DANI SURYAWAN
43	0710683043	WIDHI YAHYA
44	0710683044	MAYDANIL ARKHAM
45	0710683045	FIRMAN JATI P
46	6077410337	TAUFIQ RIZALDI
47	6077410403	KURNIASARI TRI AMANU
48	6077410670	DIRGA SUGIASTU F
49	6077410765	IRZA NOVA WIWEKA
50	6077411206	REDY ERDIYANTO

Lampiran 3. Daftar perubahan (*change log*) aplikasi jejaring sosial kampus berbasis GPS pada *smartphone* android.

Increment 1 (*Prototype*).

Daftar kebutuhan fungsional *user*.

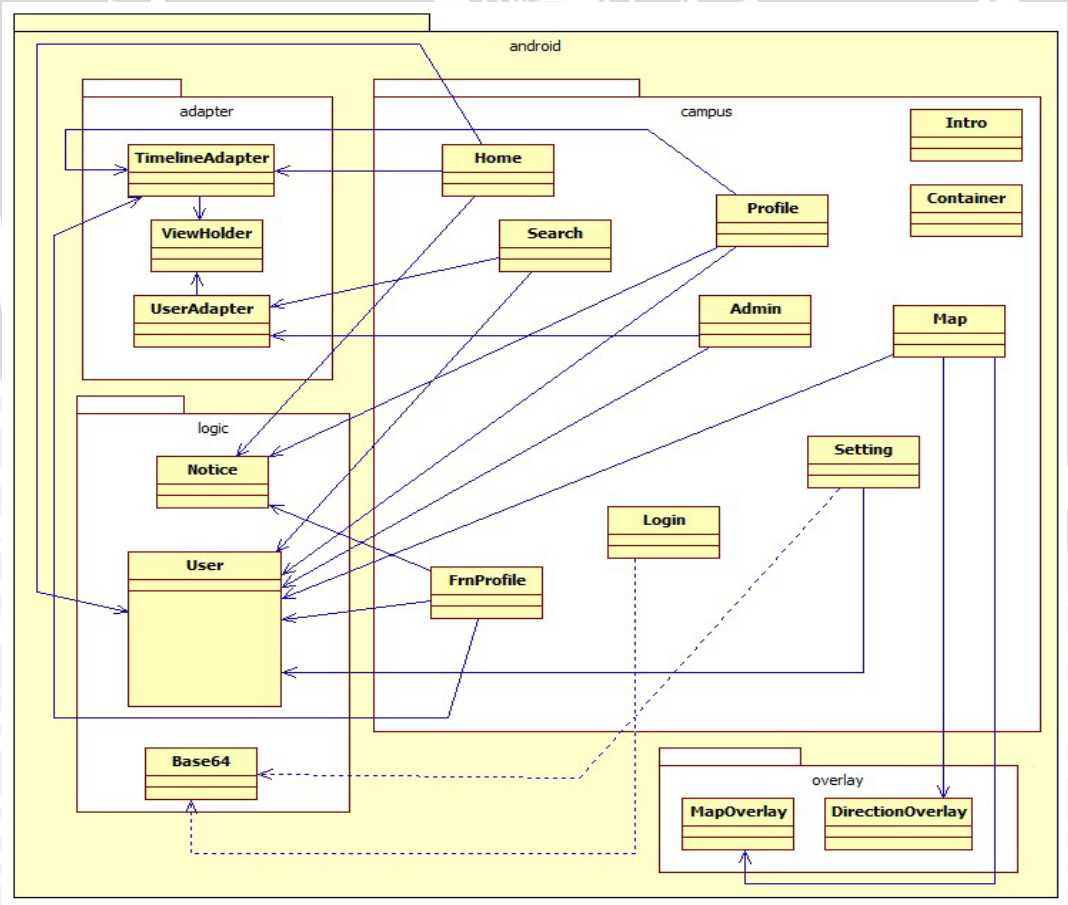
SRS-Id	Deskripsi	*Use Case*
SRS-001-01	Sistem harus menyediakan fasilitas untuk membuat *account* di sistem.	Registrasi
SRS-001-02	Sistem harus menyediakan fasilitas untuk melakukan autentifikasi ke sistem.	*Login*
SRS-001-03	Sistem harus menyediakan komponen untuk user yang telah *login* dapat keluar dari sistem.	*Logout*
SRS-002-01	Sistem harus menyediakan komponen untuk melakukan *posting* ke sistem.	Melakukan *Posting*
SRS-002-02	Sistem harus menyediakan fasilitas untuk melihat *timeline* dari daftar *posting*.	Melihat *Timeline*
SRS-002-03	Sistem harus menyediakan komponen untuk menghapus *posting* yang dilakukan.	Menghapus *Posting*
SRS-003-01	Sistem harus menyediakan fasilitas untuk melihat informasi lokasi dari *user* dalam bentuk peta (*map*).	Melihat Lokasi
SRS-003-02	Sistem harus menyediakan komponen di dalam peta (*map*) untuk melihat jalur yang dapat dilalui oleh seorang *user* menuju lokasi *user* lain.	Melihat Jalur
SRS-004-01	Sistem harus menyediakan fasilitas untuk mencari *user* berdasarkan nama *user*.	Mencari Teman
SRS-004-02	Sistem harus menyediakan fasilitas untuk melihat info dari seorang *user*.	Melihat Info
SRS-004-03	Sistem harus menyediakan fasilitas untuk seorang *user* dapat menambah *user* lain sebagai teman.	Menambah Teman
SRS-004-04	Sistem harus menyediakan fasilitas untuk menghapus teman.	Menghapus Teman
SRS-005-01	Sistem harus menyediakan fasilitas untuk *user* dapat mengganti biodata dan *password*.	*Edit Account*

Lampiran 3. (Lanjutan)

Daftar kebutuhan fungsional admin.

SRS-Id	Deskripsi	*Use Case*
SRS-006-01	Sistem harus menyediakan fasilitas bagi admin untuk melakukan autentifikasi ke sistem.	*Login*
SRS-006-02	Sistem harus menyediakan fasilitas bagi admin untuk mencari *user* berdasarkan *user id*.	Mencari *User*
SRS-006-03	Sistem harus menyediakan fasilitas bagi admin untuk mengeset ulang (*reset*) *password* pengguna.	*Reset Password*
SRS-006-04	Sistem harus menyediakan fasilitas bagi admin untuk menghapus *user*.	Menghapus *User*

***Class diagram* aplikasi pada *increment* 1.**



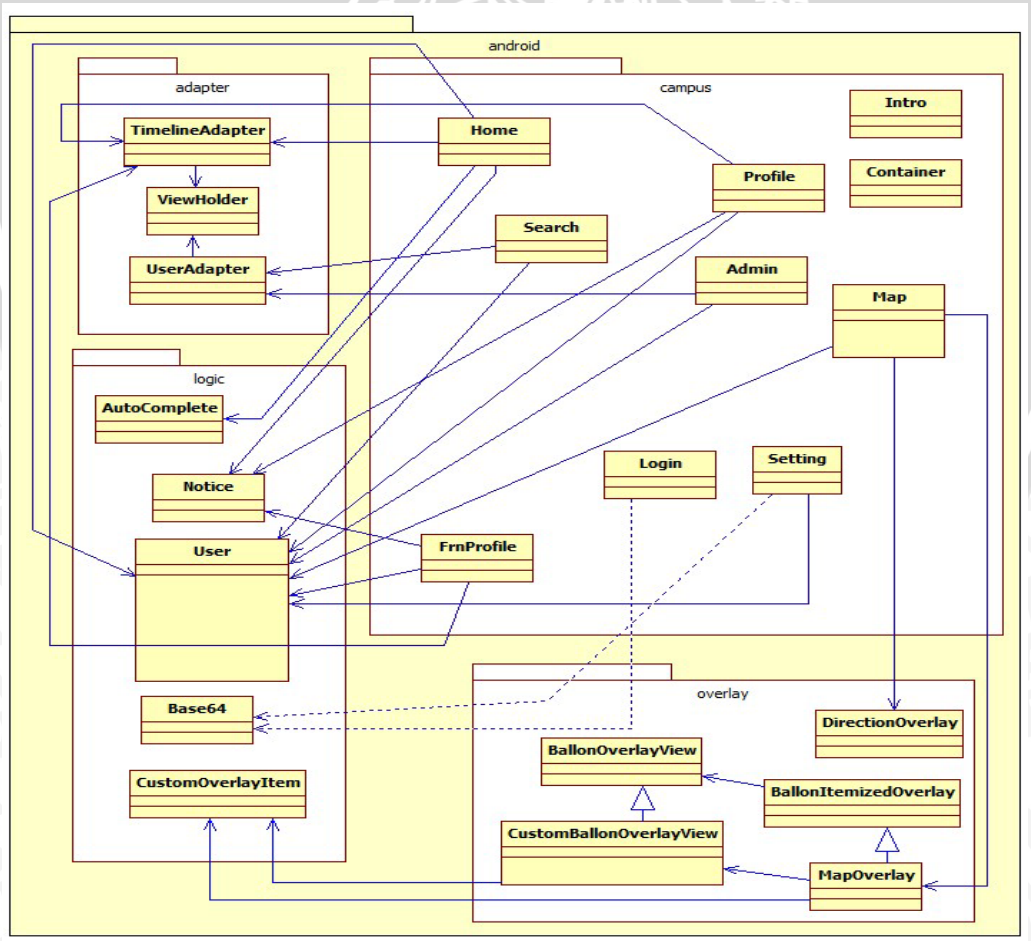
Lampiran 3. (Lanjutan)

Increment 2.

Daftar Perubahan

Revisi	Deskripsi
SRS-002-01	Penambahan fitur untuk melakukan *mention* saat menuliskan @[nama teman] pada *form posting*.
SRS-003-01	Penambahan fitur untuk membuat *marker* lebih interaktif dengan *user* seperti fasilitas untuk menampilkan *avatar* (gambar profil) dan nama *user* saat *marker* peta ditekan dan menampilkan informasi lokasi saat *marker* ditekan untuk kedua kalinya.

***Class diagram* aplikasi pada *increment* 2.**



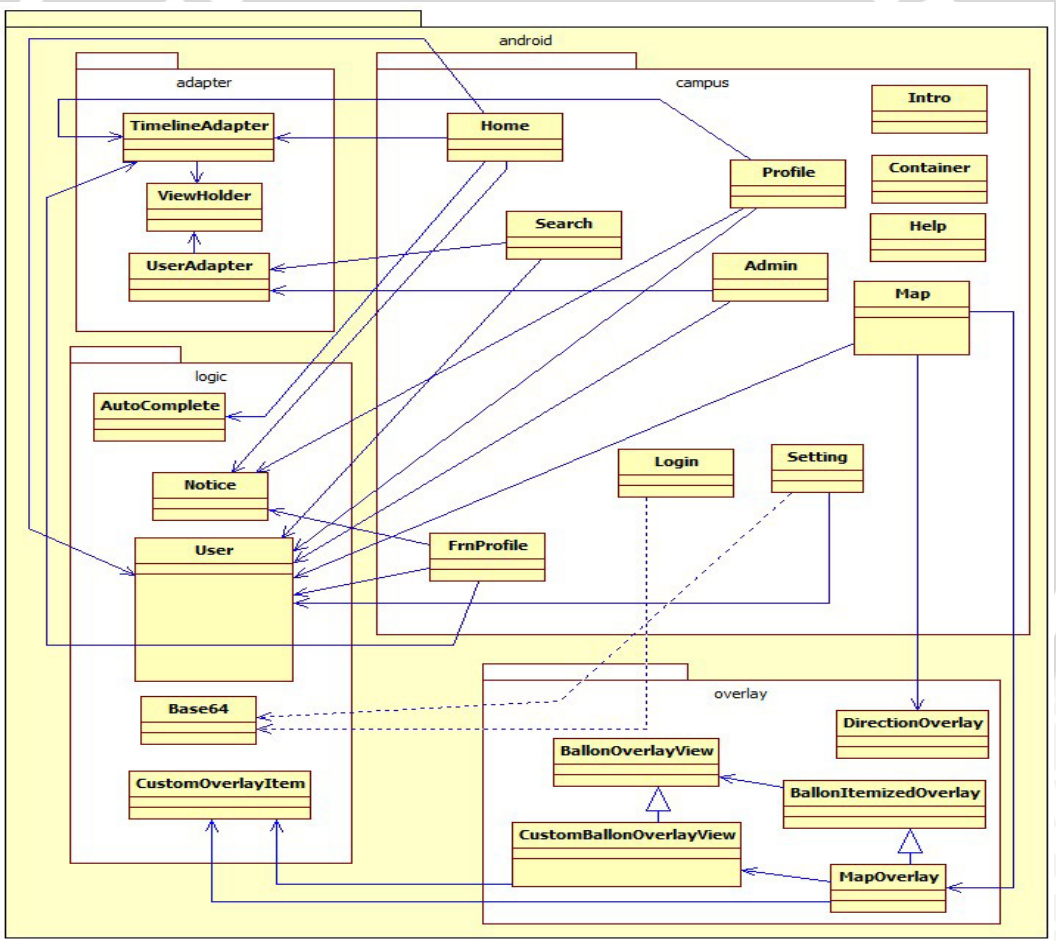
Lampiran 3. (Lanjutan)

Increment 3.

Daftar Perubahan

Revisi	Deskripsi
A	Penambahan menu bantuan penggunaan (*help*) untuk memudahkan pengguna aplikasi.

***Class diagram* aplikasi pada *increment* 3.**



Lampiran 4. Kartu Kendali Pembimbingan Skripsi

KEMENTERIAN PENDIDIKAN NASIONAL
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK INFORMATIKA
Jalan MT Haryono 167 Telp & Fax. 0341 554 166 Malang-65145

KODE
PS 14-10

KARTU KENDALI PEMBIMBINGAN SKRIPSI

Nama Mahasiswa	: KOMANG CANDRA B
NIM	: 0710683021
Judul Skripsi	: RANCANG BANGUN APLIKASI JEJARING SOSIAL
	KAMPUS BERBASIS GPS PADA SMART PHONE ANDROID
Tanggal Pengajuan	:
Batas Akhir Penyelesaian	:

No	Tanggal	Uraian Materi Pembimbing	Paraf	
			Pembimbing 1	Pembimbing 2
1	25-10-2011	Revisi pembuatan kalimat	7	
2	02-11-2011	Struktur Penulisan kalimat	7	
3	08-11-2011	Penulisan Sitasi	7	
4	18-11-2011	Konsultasi Project		✓
5	25-11-2011	BAB I, Latar belakang	7	
6	2-12-2011	Bab II teori pendukung	7	
7	16-12-2011	Bab III diperbaiki	7	
8	30-12-2011	Bab III Konsultasi DF		
9		Perancangan / Blok Diagram	7	
10		Analisa -	7	
11	6-1-2012	Bab III OK, Bab II		
12		II diperbaiki		
13		Demo Project		✓
14	20/1-2012	Lanjut Bab IV	7	
15		Teori Bab Data, Desain Interface		
16	3/2-2012	Bab Analisa	7	

Catatan:

1. *Skripsi disahkan tanggal:*
2. *Konsultasi bimbingan min. 1x/minggu (16x/semester), kolom tanggal ditulis terlebih dahulu*
3. *Kartu ini sebagai syarat untuk pelaksanaan seminar hasil dan ujian skripsi*

Mengetahui, Ketua Program Studi,	Pembimbing 1	Menyetujui, Pembimbing 2
Ir. Sutrisno, MT	Arief Andy Soebroto, ST., M.Kom	Issa Arwani, S.Kom., M.Sc
NIP. 19570325 198701 1 001	NIP. 19720425 199603 1 002	NIP. 830922 06 1 10074

Lampiran 4. Kartu Kendali Pembimbingan Skripsi (Lanjutan)

	KEMENTERIAN PENDIDIKAN NASIONAL UNIVERSITAS BRAWIJAYA FAKULTAS TEKNIK **PROGRAM STUDI TEKNIK INFORMATIKA** Jalan MT Haryono 167 Telp & Fax. 0341 554 166 Malang-65145	**KODE** **PS 14-10**

KARTU KENDALI PEMBIMBINGAN SKRIPSI

Nama Mahasiswa	: KOMANG CANDRA B
NIM	: 0710683021
Judul Skripsi	: RANCANG BANGUN APLIKASI JEJARING SOSIAL
	KAMPUS BERBASIS GPS PADA SMART PHONE ANDROID
Tanggal Pengajuan	:
Batas Akhir Penyelesaian	:

No	Tanggal	Uraian Materi Pembimbing	Paraf	
			Pembimbing 1	Pembimbing 2
1	10-2-2012	Sequence diagram		
2	24/2-2012	[illegible] → profile		
3	7/3-2012	[illegible] di tabel		
4	16/3-2012	[illegible] topik skripsi		
5		[illegible] relasi class		
6	21/3-2012	Bab IV - Interface di		
7	30/3 '12	perbaiki		
8		Bab V - [illegible] source		
9		Code di [illegible]		
10	3/4 '12	layout implementasi		
11	4/5 12	Interface Program		
12	11/5 12	Pengujian Program		
13	15/5 '12	- Implementasi interface pengujian [illegible] grafik		
14		- [illegible] paper.		
15		- [illegible]		
16	25/5 '12	Perbaikan [illegible] hasil		

Catatan:

1. *Skripsi disahkan tanggal:*
2. *Konsultasi bimbingan min. 1x/minggu (16x/semester), kolom tanggal ditulis terlebih dahulu*
3. *Kartu ini sebagai syarat untuk pelaksanaan seminar hasil dan ujian skripsi*

Mengetahui, Ketua Program Studi,	Pembimbing 1	Menyetujui, Pembimbing 2
Ir. Sutrisno, MT **NIP. 19570325 198701 1 001**	Arief Andy Soebroto, ST., M.Kom **NIP.** 19720925 199603 1 002	Issa Arwani, S.Kom., MSc **NIP.** 830922 06 11 0079

Lampiran 4. Kartu Kendali Pembimbingan Skripsi (Lanjutan)

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS BRAWIJAYA
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
PROGRAM STUDI TEKNIK INFORMATIKA
Gedung A PTIIK UB
JL. Veteran No.8, Malang, 65145, Indonesia
Telp. : +62-341-577911 Fax : +62-341-577911
http://ptiik.ub.ac.id E-mail : ptiik@ub.ac.id

KARTU KENDALI PEMBIMBINGAN SKRIPSI

KODE 13-08

Nama Mahasiswa	: KOMANG CANDRA B
NIM	: 0710683021
Judul Skripsi	: RANCANG BANGUN APLIKASI JEJARING SOSIAL
	KAMPUS BERBASIS GPS PADA SMART PHONE ANDROID
Tanggal Pengajuan	:
Batas Akhir Penyelesaian	:

No	Tanggal	Uraian Materi Pembimbing	Paraf	
			Pembimbing 1	Pembimbing 2
1	1-6-2012	- Data test berpola - Kesimpulan pada abstrak.		
2	7-6-2012	Konsultasi Paper		✓
3	8/6-12	Dilanjutkan Seminar Hasil		
4		Batasan ⊕ Stp rumusan		
5		Jurnal.		
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				

Catatan:

1. *Skripsi disahkan tanggal:*
2. *Konsultasi bimbingan min. 1x/minggu (16x/semester), kolom tanggal ditulis terlebih dahulu*
3. *Kartu ini sebagai syarat untuk pelaksanaan seminar hasil dan ujian skripsi*

Mengetahui,
Ketua Jurusan,

Drs. Marji, MT.
NIP. 19670801 199203 1 001

Pembimbing 1

Arief Andy Soebroto, ST., M.Kom
NIP. 19720925 199603 1002

Menyetujui,
Pembimbing 2

Issa Arwani, S.Kom., M.Sc
NIP. 830922 06 1100 79