SKRIPSI

PEMBUATAN *TWITTER BOT* UNTUK MENCARI JALUR TRANSPORTASI PUBLIK

KEVIN THEODORUS YONATHAN

NPM: 2011730037

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2015

UNDERGRADUATE THESIS

DEVELOPING TWITTER BOT FOR LOCATING PUBLIC TRANSPORT ROUTE

KEVIN THEODORUS YONATHAN

NPM: 2011730037

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2015

LEMBAR PENGESAHAN

PEMBUATAN *TWITTER BOT* UNTUK MENCARI JALUR TRANSPORTASI PUBLIK

KEVIN THEODORUS YONATHAN

NPM: 2011730037

Bandung, 03 Juni 2015

Menyetujui,

Pembimbing Tunggal

Thomas Anung Basuki, Ph.D.

Ketua Tim Penguji	Anggota Tim Penguji
Chandra Wijaya, M.T.	**Luciana Abednego, M.T.**

Mengetahui,

Ketua Program Studi

Thomas Anung Basuki, Ph.D.

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PEMBUATAN *TWITTER BOT* UNTUK MENCARI JALUR TRANSPORTASI PUBLIK

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 03 Juni 2015

Meterai

Kevin Theodorus Yonathan
NPM: 2011730037

ABSTRAK

Transportasi publik sudah banyak digunakan oleh kebanyakan orang di Indonesia. Keuntungan memakai transportasi publik sudah banyak dirasakan yaitu untuk mengatasi kemacetan dan mengurangi pemanasan global. KIRI adalah website yang dapat mencari jalur transportasi publik dari lokasi awal menuju lokasi tujuan. Seiring dengan perkembangan zaman, perkembangan internet di Indonesia sudah semakin maju. Banyak orang sudah menggunakan fasilitas internet untuk berbagai macam kebutuhan terutama untuk jejaring sosial online. Salah satu jejaring sosial online yang sudah banyak digunakan orang-orang adalah Twitter. Twitter adalah salah satu layanan jejaring sosial online yang memungkinkan pengguna memposting pesan berbasis teks hingga 140 karakter.

Pada tugas akhir ini penulis akan membuat perangkat lunak *Twitter bot* untuk mencari jalur transportasi publik. Dalam pembuatan Twitter bot, penulis memanfaatkan KIRI API dan Twitter API. KIRI API digunakan untuk memberi jalur transportasi publik, sedangkan Twitter API digunakan untuk menangkap *tweet* dan membalas *tweet*. *Twitter bot* yang dibangun pada penelitian ini menggunakan bahasa Java dan menggunakan library dari Twitter4J.

Dari hasil pengujian, diperoleh bahwa *Twitter bot* yang dibuat sudah berjalan dengan baik. Pengguna sudah dapat mencari jalur transportasi publik dari suatu lokasi menuju lokasi tujuan dengan melakukan mention kepada akun *Twitter bot*. Lalu *Twitter Bot* melakukan balasan kepada pengguna berupa tweet yang berisikan jalur transportasi publik yang harus ditempuh.

Kata-kata kunci: Twitter, Rute, Transportasi Publik

ABSTRACT

Public transport is widely used by most people in Indonesia. Taking advantage of public transportation has been much felt that to tackle congestion and reduce global warming. KIRI is a website that can search for public transport route from the starting location to the destination location. Along with the times, development of the Internet in Indonesia is more advanced. Many people already use the internet facilities for a variety of needs, especially for online social networking. One of the online social networks are already widely used are Twitter. Twitter is one of the online social networking service that allows users to post text-based messages of up to 140 characters.

In the manufacture Twitter bot, the authors utilize KIRI API and Twitter API. KIRI API is used to provide public transportation route, while the Twitter API is used to capture the tweet and reply tweet. Twitter bot that be built using Java language and use the library from Twitter4J.

From the test results, obtained that the Twitter bot that made already running well. Users are able to search for public transport route from one location to the location of the destination by making mention to a Twitter account bot. Then Twitter Bot reply to users in the form of a tweet that contains the public transportation lines that must be taken.

Keywords: Twitter, Route, Public Transport

Dipersembahkan untuk diri sendiri

KATA PENGANTAR

Puji Syukur penulis kepada Tuhan yang telah memberikan rahmatnya sehingga penulis dapat menyelesaikan skripsi yang berjudul "**Pembuatan witter Bot Untuk Mencari Jalur Transportasi Publik**". Skripsi ini disusun dengan maksud memenihi salah satu prasyarat menyelesaikan pendidikan di Jurusan Teknik Informatika, Fakultas Teknologi Informasi dan Sains, Universitas Katolik Parahyangan. Penulis menyadari bahwa dalam penulisan skripsi ini tidak terlepas dari bantuan dan dukungan berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada:

- Pak Thomas Anung Basuki selaku pembimbing yang telah memberikan banyak masukan untuk skripsi ini sehingga skripsi ini dapat diselesaikan dengan baik.
- Pak Pascal Alfadian atas masukan dan tambahan wawasan untuk skripsi ini sehingga skripsi ini dapat diselesaikan dengan baik.
- Keluarga yang selalu memberi dukungan baik moril maupun materil dan doa.
- Ibu Luciana dan Pak Chandra selaku penguji yang sudah memberikan banyak masukan dalam penyusunan skripsi ini.
- Segenap teman penulis Yohan Sugiyo, Jovan Gunawan, Samuel Christian, Antonio Yaphiar, Steven Christian, Clara, William Wibisono, Calvin Christian, Edbert Jeremiah, teman-teman MWS yang sudah memberi dukungan dan bantuan dalam penyusunan skripsi ini.

Semoga segala bantuan dan dukungan berbagai pihak tersebut mendapat berkat dari Tuhan. Semoga skripsi ini berguna bagi semua orang dan dapat dijadikan bahan pembelajaran. Akhir kata, penulis mohon maaf apabila kesalahan dan kekurangan dalam penulisan skripsi ini.

Bandung, Juni 2015

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Seiring dengan perkembangan zaman, perkembangan internet di Indonesia sudah semakin maju. Banyak orang sudah menggunakan fasilitas internet untuk berbagai macam kebutuhan. Contoh dari penggunaan internet adalah untuk mencari informasi, *email*, bermain jejaring sosial online, *Internet Banking*, *online shop*, dan lain lain. Menurut Kominfo penggunaan internet di Indonesia sudah mencapai 82 juta orang, delapan puluh persen diantaranya adalah remaja[1]. Hal ini menunjukkan bahwa internet sudah tidak asing lagi untuk masyarakat di Indonesia ini. Sebagai informasi tambahan bahwa pengguna internet di Indonesia 95 persennya digunakan untuk sosial media atau jejaring sosial online [2].

Twitter adalah salah satu layanan jejaring sosial online yang memungkinkan pengguna mem*posting* pesan berbasis teks hingga 140 karakter. Pengguna Twitter menyebutnya sebagai *tweet*. *Tweet* ini akan meneruskan pesan singkat yang ditujukan ke semua *follower* suatu akun[3].*Follow* adalah salah satu istilah dalam Twitter yang bertujuan untuk mengikuti aktivitas *tweet* suatu akun. Sedangkan cara seseorang untuk dapat memberi rujukan kepada akun Twitter yang lainnya adalah dengan cara melakukan *reply* atau lebih dikenal dengan nama *mention*[4]. Sebagai contoh, diketahui akun bernama @kviniink mem-*follow* @infobdg untuk mengetahui perkembangan apa saja yang tejadi di Kota Bandung. Lalu akun @kviniink ingin bertanya tentang info *mall* yang sedang ramai dikunjungi di Bandung, maka akun @kviniink membuat *mention tweet* kepada akun @infobdg yang berisikan "@infobdg Halo saya ingin bertanya *mall* apa yang sedang ramai dikunjungi di Bandung yah?". Setelah akun @infobdg membaca *mention* dari @kviniink, akun @infobdg melakukan balasan kepada akun @kviniink untuk membalas pertanyaan yang diajukan oleh akun @kviniink. *Tweet* tersebut berisikan "@kviniink mungkin anda bisa mengunjungi PVJ dan Ciwalk".

Transportasi publik sudah banyak digunakan oleh kebanyakan orang di dunia. Bukan hanya di Indonesia saja transportasi publik ini sudah banyak digunakan di luar negeri. Menurut data, angkutan umum di Kota Bandung pada tahun 2013 jumlahnya sudah lebih dari 12000 unit ken-

[1]Kominfo bint005 , Pengguna Internet di Indonesia Capai 82 Juta, http://kominfo.go.id/index.php/content/detail/3980/Kemkominfo%3A+Pengguna+Internet+di+Indonesia+Capai+82+Juta/0/berita_satker, pada tanggal 15 April 2015 pukul 12.58

[2]Kominfo bint005 , Pengguna Internet di Indonesia 63 Juta Orang, http://kominfo.go.id/index.php/content/detail/3415/Kominfo+%3A+Pengguna+Internet+di+Indonesia+63+Juta+Orang/0/berita_satker, pada tanggal 15 April 2015 pukul 13.10

[3]Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, halaman 7

[4]Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, halaman 9

daraan[5]. Keuntungan memakai transportasi publik sudah banyak dirasakan di seluruh dunia yaitu untuk mengatasi kemacetan dan mengurangi pemanasan global. Seiring dengan perkembangan teknologi, menaiki transportasi publik menjadi semakin mudah. Dengan adanya KIRI di Indonesia terutama di Kota Bandung, masyarakat dapat manaiki transportasi publik tanpa harus mengetahui terlebih dahulu jalur yang harus ditempuh pengguna. Pengguna hanya perlu tahu tempat asal dan tempat tujuan untuk menaiki transportasi publik di Kota Bandung.

KIRI API adalah aplikasi pihak ketiga yang memungkinkan pengembang perangkat lunak mendapatkan data tentang info jalur transportasi publik. KIRI API dapat mencari suatu nama lokasi tempat terkenal seperti *mall* dan universitas, selain itu KIRI juga dapat mencari berdasarkan nama jalan seperti jalan astana anyar, jalan pajajaran, jalan kelenteng, dan lain lain. Twitter API adalah aplikasi pihak ketiga yang memungkinkan *programmer* melakukan manipulasi dan pengolahan data di Twitter. Twitter API ini dibagi menjadi dua yaitu REST API dan Streaming API. Dengan memanfaatkan KIRI API dan Twitter API peneliti membuat *Twitter bot* yang dapat membalas *tweet* untuk mencari jalur transportasi publik. *Twitter bot* yang dibuat bersifat *real time* sehingga jika seseorang melakukan mention kepada akun *Twitter bot* maka *Twitter bot* akan menangkapnya dan membalas *mention* tersebut berupa jalur yang harus ditempuh. Contoh dari jalannya *Twitter bot* adalah ketika akun bernama @kviniink melakukan *mention* kepada @kiriupdate untuk bertanya jalur transportasi publik "@kiriupdate bip to ip". Maka Twitter bot @kiriupdate akan menerima *tweet* dari akun @kviniink lalu *tweet* tersebut akan diolah oleh server dan akan di-*reply* dengan tiga buah *tweet* yaitu

1. "@kviniink Walk about 135 meter from your starting point to Jalan Aceh.",
2. "@kviniink Take angkot Ciroyom - Antapani at Jalan Aceh, and alight at Jalan Pajajaran about 3.6 kilometer later.",
3. "@kviniink Walk about 93 meter from Jalan Pajajaran to your destination.".

Dikarenakan *Tweet* memiliki keterbatasan 140 karakter maka *tweet* akan dibagi sesuai dengan instruksi yang dikirimkan dari KIRI API.

Oleh karena itu, dalam penelitian ini dibangun sebuah perangkat lunak yang dapat memudahkan pengguna dalam mencari jalur transportasi publik. Sebuah perangkat lunak yang menggabungkan jejaring sosial online Twitter dengan KIRI API. Pengguna dapat melakukan *tweet* kepada *Twitter bot* dengan format yang sudah ditentukan untuk mendapatkan *tweet* yang berisikan rute jalan yang harus ditempuhnya dengan menaiki transportasi publik.

1.2 Rumusan Masalah

Mengacu kepada deskripsi yang diberikan, maka rumusan masalah pada penelitian ini adalah:

1. Bagaimana membuat *Twitter bot* untuk mencari jalur transportasi publik?
2. Bagaimana membuat *Twitter bot* untuk dapat merespon secara *real time*?

[5]Oris Riswan , Wow... Jumlah Angkot di Bandung hampir 12 Ribu Unit, http://news.okezone.com/read/2013/11/17/526/898175/wow-jumlah-angkot-di-bandung-hampir-12-ribu-unit, pada tanggal 15 April 2015 pukul 13.15

3. Bagaimana mem-*format* petunjuk rute perjalanan dalam keterbatasan *tweet* 140 karakter?

1.3 Tujuan

Tujuan dari penelitian ini adalah:

1. Membuat aplikasi *Twitter bot* untuk mencari jalur transportasi publik.
2. Membuat aplikasi Twitter yang bekerja secara *real time*.
3. Membuat algoritma untuk memecah instruksi dari KIRI API dan mengubahnya ke dalam bentuk *tweet*.

1.4 Batasan Masalah

Pada pembuatan perangkat lunak ini, masalah-masalah yang ada akan dibatasi menjadi:

1. Masukan hanya mencakup Kota Bandung saja.
2. Masukan yang diinputkan harus benar, memiliki asal dan tujuan yang jelas di Kota Bandung.
3. Hasil yang dikeluarkan berupa *tweet* jalur transportasi publik.
4. Media transportasi publik yang digunakan adalah angkutan umum.
5. Pencarian jalur memanfaatkan KIRI API.

1.5 Metode Penelitian

Pada perangkat lunak yang dibuat ini digunakan beberapa metode dalam penyelesaian masalah yang menjadi topik pada penelitian ini, antara lain:

1. Melakukan studi literatur, antara lain:
   - KIRI API,
   - REST API Twitter (https://dev.twitter.com/docs/api/1.1),
   - Streaming API Twitter (https://dev.twitter.com/docs/api/streaming).
2. Mempelajari pembuatan server dalam bahasa Java.
3. Melakukan analisis terhadap teori-teori yang sudah dipelajari, guna membangun perangkat lunak yang dimaksud.
4. Melakukan pengujian terhadap *system* yang sudah dibangun.
5. Membuat dokumen.
6. Membuat kesimpulan.

1.5.1 Sistematika Pembahasan

Sistematika pembahasan dalam penelitian ini berupa:

- Bab Pendahuluan Bab 1 berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, metodologi penelitian, dan sistematika pembahasan.
- Bab Dasar Teori Bab 2 berisi mengenai Twitter, Twitter API, OAuth, JSON, KIRI API, dan Twitter4J.
- Bab Analisis Bab 3 berisi analisis meliputi analisis Twitter API, analisis OAuth, analisis KIRI API, analisis Twitter4J, use case, dan rancangan awal diagram kelas.
- Bab Perancangan Bab 4 berisi tahapan penjelasan diagram kelas lengkap, sequence diagram, dan perancangan antarmuka.
- Bab Implementasi dan Pengujian Bab 5 berisi tahapan implementasi pada perangkat lunak meliputi tampilan antarmuka perangkat lunak, dan pengujian perangkat lunak.
- Bab Kesimpulan dan Saran Bab 6 berisi kesimpulan serta beberapa saran untuk pengembangan lebih lanjut dari penelitian yang dilakukan dan perangkat lunak yang dibangun.

BAB 2

DASAR TEORI

Sebelum bisa membuat *Twitter bot* untuk mencari jalur transportasi publik, berikut diberikan beberapa definisi yang berkaitan dengan pembuatan *Twitter bot*. Bab ini akan menjelaskan Twitter, Twitter API, KIRI, KIRI API, dan Twitter4j.

2.1 Twitter [1]

Twitter adalah salah satu layanan jejaring sosial online yang memungkinkan pengguna melakukan *posting* pesan berbasis teks hingga 140 karakter. Berikut ini adalah daftar istilah umum pada Twitter:

- *Tweet*

  Posting pada Twitter disebut sebagai *tweet*. *Tweet* ini akan meneruskan pesan singkat yang ditujukan ke semua *follower* suatu akun. Contohnya adalah seorang akun @kviniink ingin menuliskan bahwa hari ini cuaca cerah, maka akun @kviniink akan melakukan *tweet* 'Hari ini cerah yah..'. *Tweet* juga bisa menyertakan *link* untuk video, foto, atau media lain di internet selain teks biasa. URL(*Uniform resource locator*) *link* teks termasuk ke dalam 140 batas karakter, namun URL tersebut akan menghabisnya tempat/*space* dari keterbatasan karakter *tweet*. Oleh karena itu, URL akan dibuat versi singkatnya, contohnya pada saat pengguna memasukkan *link* `http://www.chacha.com/gallery/7253/15-movies-that-make-guys-cry`, maka *link* tersebut akan dibuat menjadi `bit.ly/1uRi8vV`.

- *Follow*

  Follow adalah satu istilah dalam Twitter yang bertujuan untuk mengikuti aktivitas *tweet* suatu akun. *Following* adalah ketika sebuah akun mengikuti akun orang lain, dan *Follower* adalah ketika sebuah akun melakukan aksi *follow* kepada akun anda.

- *Reply*

  Reply adalah cara seseorang untuk dapat memberi rujukan kepada akun Twitter yang lainnya atau lebih dikenal dengan nama *mention*. Sebagai contoh, diketahui akun bernama @kviniink mem-*follow* @infobdg untuk mengetahui perkembangan apa saja yang tejadi di Kota Bandung. Lalu akun @kviniink ingin bertanya tentang info *mall* yang sedang ramai dikunjungi di Kota Bandung, maka akun @kviniink membuat *mention tweet* yang berisikan "@infobdg Halo saya ingin bertanya apa saja *mall* yang sedang ramai dikunjungi di Bandung yah?". Setelah akun

@infobdg membaca *mention* dari @kviniink, akun @infobdg melakukan balasan kepada akun @kviniink untuk membalas pertanyaan yang diajukan oleh akun @kviniink. *Tweet* tersebut berisikan "@kviniink mungkin anda bisa mengunjungi PVJ dan Ciwalk".

- *Retweet*

  Retweet ini merupakan salah satu istilah penting dari Twitter. *Retweet* ini berguna ketika pengguna menemukan *tweet* menarik dan ingin berbagi *tweet* tersebut dengan *follower* akun tersebut. *Retweet* ini juga secara tidak langsung mengatakan bahwa "saya menghormati anda dan pesan yang anda buat". Contohnya adalah ketika akun @infobdg melakukan *tweet* tentang penutupan jalan di Bandung dan suatu akun ingin membagian informasi tersebut kepada *follower* mereka. Maka akun tersebut akan melakukan *retweet* terhadap *tweet* yang diposting oleh akun @infobdg.

- *Hashtag*

  Sebuah fitur yang diciptakan oleh Twitter untuk membantu pencarian kata kunci dan penandaan suatu diskusi. Contohnya adalah ketika suatu akun mendatangi suatu acara bernama IXPO, dan akun tersebut ingin memposting *tweet* tentang IXPO tersebut, maka akun tersebut dapat menuliskan #IXPO pada *tweet* yang diposting.

- *Direct Message*(DM)

  Direct message digunakan untuk mengirim pesan yang bersifat *private* antara dua akun Twitter. Syarat agar dapat melakukan *direct message* adalah melakukan aksi *follow* terhadap akun yang akan dikirimkan *direct message*.

- *Timeline*

  Timeline adalah sekumpulan *tweet* dari semua akun yang di-*follow*. *Timeline* ditampilkan di halaman utama.

2.2 Twitter API [2]

Twitter API(*Application programming interface*) adalah aplikasi pihak ketiga yang memungkinkan pengembang perangkat lunak melakukan manipulasi dan pengolahan data di Twitter. Twitter API adalah salah satu bentuk pendekatan dari Twitter yang berfokus pada jaringan dan memungkinkan pengembang perangkat lunak memiliki hak untuk berpikir '*out of the box*' untuk membuat aplikasi yang mereka inginkan[2]. Tetapi tetap akan terjadi keterbatasan yang dimiliki Twitter API, yaitu :

- Hanya bisa melakukan *tweet* 1000 per harinya, baik melalui *handphone*, *website*, API, dan sebagainya.
- Total pesan hanya bisa sebanyak 250 per harinya, pada setiap dan semua perangkat.
- 150 permintaan API per jam.

2.2.1 *Search* API

Twitter *Search* API memungkinkan melakukan pencarian terhadap *tweet* baru ataupun *tweet* populer. Tetapi Twitter *Search* API ini bukan fitur yang tersedia pada *website* Twitter itu sendiri. API ini difokuskan kepada relevansi, bukan terhadap kelengkapan data[2]. Ini berarti bahwa ada beberapa *Tweet* atau akun akan hilang dari hasil pencarian.

Bagaimana cara membuat sebuah *query* Cara terbaik dalam membuat sebuah *query* adalah melakukan percobaan yang valid dan mengembalikan *tweet* yang sesuai. Cara mencobanya dapat dilakukan pada `twitter.com/search`. URL yang ditampilkan pada *browser* akan berisi sintaks *query* yang sesuai agar dapat digunakan kembali pada Twitter API. Berikut adalah contohnya:

1. Melakukan pencarian untuk *tweet* yang di-*mention* kepada akun @twitterapi. Pencarian dilakukan pada `twitter.com/search`.
2. Lakukan pengecekan dan salin URL yang ditampilkan pada browser. Sebagai contoh didapatkan URL seperti `https://twitter.com/search?q=%40twitterapi`.
3. Ganti `https://twitter.com/search` dengan `https://api.twitter.com/1.1/search/tweets.json` dan akan didapatkan `https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi`
4. Eksekusi URL tersebut untuk melakukan pencarian di dalam API.

API v1.1 mewajibkan *request* yang sudah diotentikasi. Perlu diingat juga bahwa hasil pencarian yang dilakukan di `twitter.com` dapat menghasilkan data yang sudah sangat lama, sedangkan *Search* API hanya melayani *tweet* dari seminggu terakhir. Contoh berbagai macam pencarian dapat dilihat pada tabel 2.1:

Dipastikan bahwa pengkodean URL terhadap *query* dilakukan terlebih dahulu sebelum melakukan *request*. Tabel 2.2 memberikan contoh *mapping* dari *search query* ke *query* pengkodean URL.

Additional parameters Terdapat parameter tambahan yang dapat digunakan untuk menghasilkan pencarian yang lebih baik. Berikut adalah penjelasan dari parameter tambahan tersebut :

- ***Result Type***. Seperti hasil yang terdapat pada `twitter.com/search`, parameter *result_ type* memungkinkan hasil pencarian akan berdasarkan *tweet* yang paling baru atau *tweet* yang paling populer atau bahkan gabungan dari keduanya.
- ***Geolocatization***. Pencarian tempat tidak tersedia pada API, tetapi ada beberapa cara yang tepat untuk membatasi *query* dengan cara menggunakan parameter *geocode* lalu menentukan "*latitude, longitude, radius*". Contohnya adalah "37.781157,-122.398720,1mi". Ketika pencarian lokasi, pencarian API akan mencoba menemukan *tweet* yang memiliki *latitude* dan *longitude* yang sudah dimasukkan kedalam *query geocode*, jika tidak berhasil maka API akan mencoba menemukan *tweet* yang dibuat oleh pengguna yang lokasi profilenya terdapat pada *latitude* dan *longitude* tersebut. Kesimpulannya adalah hasil pencarian dapat menerima *tweet* yang tidak mencakup informasi *latidute* atau *longitude*.

Tabel 2.1: Contoh berbagai macam pencarian *tweet*

Operator	**Finds *tweets***
watching now	Mengandung kata "*watching*" dan "*now*".
"*happy hour*"	Mengandung frase "*happy hour*" yang tepat.
love OR hate	Mengandung kata "*love*" atau "*hate*" atau keduanya.
beer -root	Mengandung kata "*beer*" tanpa adanya kata "*root*".
#haiku	Mengandung *hashtag* "*haiku*".
from:alexiskold	Dikirim melalui akun "alexiskold".
to:techcrunch	Dikirimkan kepada akun "techcrunch".
@mashable	Mereferensi kepada akun "mashable".
superhero since:2010-12-27	Mengandung kata "*superhero*" dari tanggal "2010-12-27" (tahun-bulan-hari).
ftw until:2010-12-27	Mengandung kata "*ftw*" sebelum tanggal "2010-12-27".
movie -scary :)	Mengandung kata "*movie*", tanpa adanya kata "*scary*", dengan pencarian yang positif.
flight :(	Mengandung kata "*flight*" dengan pencarian yang negatif.
traffic ?	Mengandung kata "*traffic*" dan mengandung pertanyaan.
hilarious filter:links	Mengandung kata "*hilarious*" yang di sambungkan dengan URL.
news source:twitterfeed	Mengandung kata "*news*" yang di-*posting* melalui *twitterfeed*.

Tabel 2.2: Contoh *mapping* dari *search query* ke *query* pengkodean URL

Search query	***URL encoded query***
#haiku #poetry	%23haiku+%23poetry
"*happy hour*" :)	%22*happy*%20*hour*%22%20%3A%29

- ***Language***. Bahasa dapat dijadikan parameter untuk mencari *tweet* yang sesuai dengan bahasa yang dipilih.
- ***Iterating in a result set***. Parameter seperti *count, until, since_ id, max_ id* memungkinkan untuk melakukan kontrol bagaimana iterasi melalui hasil pencarian.

Rate limits *User* pada saat ini diwakilkan oleh *access tokens* yang dapat membuat 180 *request* per 15 menit. Tetapi pengguna bisa membuat 450 *request* per 15 menit dengan menggunakan *application-only authentication* atas nama sendiri tanpa konteks pengguna.

Contoh Pencarian Ketika anda mengikuti suatu acara yaitu *superbowl*, lalu anda tertarik untuk mencari hal yang sedang terjadi di acara tersebut dengan melihat *tweet* yang paling baru dan menggunakan *hashtag* dari acara tersebut, maka langkah-langkah yang dilakukan adalah:

- Anda ingin mencari *tweet* yang paling baru dengan menggunakan *hashtag* #*superbowl*
- Maka *search* URL akan seperti ini: `https://api.twitter.com/1.1/search/tweets.json?q=%23superbowl&result_type=recent`

Ketika anda ingin mengetahui *tweet* yang datang dari suatu lokasi dengan bahasa yang spesifik, maka langkah-langkah yang dilakukan adalah:

- Anda ingin mencari *tweet* yang paling baru dalam Bahasa Portugal, yang lokasinya dekat Maracana soccer stadium yang terletak di Rio de Janeiro.
- Maka search URL akan seperti ini: https://api.twitter.com/1.1/search/tweets.json?q=&geocode=-22.912214,-43.230182,1km&lang=pt&result_type=recent

  Ketika anda ingin mencari *tweet* yang sedang poluler dari spesifik *user* dan *tweet* tersebut terdapat sebuah hashtag tertentu, maka langkah-langkah yang dilakukan adalah:

  – Anda ingin mencari *tweet* yang populer yang berasal dari *user* @kviniink yang terdapat *hashtag* #nasa.

  – Maka *search* URL akan seperti ini: https://api.twitter.com/1.1/search/tweets.json?q=from%3Akviniink%20%23nasa&result_type=popular

2.2.2 Streaming API

Streaming API adalah contoh *real-time* API. API ini ditujukan bagi para developer dengan kebutuhan data yang intensif. *Streaming* API memungkinkan melacak kata kunci yang ditentukan dalam jumlah besar dan melakukan suatu aksi (seperti *tweet*) secara langsung atau *real-time*.

Twitter menawarkan beberapa *endpoint streaming*, disesuaikan dengan kasus yang dibutuhkan.

- *Public stream*

  Public stream merupakan *streaming* data publik yang mengalir melalui Twitter. *Public stream* Digunakan untuk mengikuti sebuah *user* atau topik tertentu. *Public stream* biasa digunakan untuk *data mining*.

- *User Stream*

  User Stream merupakan *Single-user streams* yang mengandung hampir semua data yang berhubungan dengan satu *user* tertentu.

- *Site Stream*

  Site Stream merupakan versi dari *multi-user stream*. *Site stream* terhubung dengan server yang terkoneksi dengan Twitter atas nama banyak pengguna.

Public Streams *Stream* ini menawarkan sampel data publik yang mengalir melalui Twitter. Ketika aplikasi membuat sambungan ke *streaming endpoint*, perangkat lunak akan mengambil *tweet* tanpa perlu khawatir akan keterbatasan *rate limit*.

Endpoints

- POST statuses / *filter*
- GET statuses / *sample*
- GET statuses / *firehose*

POST statuses/*filter* POST *filter* dapat mengembalikan status publik yang sesuai dengan satu atau lebih predikat yang telah difilter. *Multiple parameter* memungkinkan klien untuk menggunakan koneksi tunggal untuk ke *Streaming* API. Antara GET *request* dan POST *request* keduanya didukung oleh POST statuses / *filter* tetapi untuk GET *request* yang memiliki parameter yang terlalu banyak mungkin akan ditolak karena URL yang terlalu panjang. Gunakanlah POST request untuk menghindari URL yang panjang. *Track, follow*, dan lokasi harus dipertimbangkan untuk dapat digabungkan dengan operator OR. *track*=foo&*follow*=1234 ini mengembalikan *tweet* yang memiliki kata "foo" atau dibuat oleh *user* 1234. Akses standar mengizinkan pencarian hingga 400 kata kunci, dan 5000 *follow userids*. Perintah ini dikembalikan dalam format JSON, memerlukan otentikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk POST statuses/*filter* dapat dilihat pada tabel 2.3

Tabel 2.3: Parameter POST statuses/*filter*

follow	Menentukan pencarian *tweet* dari suatu akun.
track	Kata kunci pencarian untuk di-*track*.
locations	Menentukan lokasi yang dilacak.
delimited	Menentukan apakah pesan harus dibatasi limitnya.
stall_warnings	Menentukan apakah pesan warning harus dikirim atau tidak.

GET *statuses/sample* Mengembalikan *random* sampel dari semua status publik. Jika terdapat dua *client* yang terhubung dengan *endpoint* ini, maka kedua *client* tersebut akan melihat *tweet* yang sama. Perintah ini dikembalikan dalam format JSON, memerlukan otentikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter GET *statuses/sample* dapat dilihat pada tabel 2.4

Tabel 2.4: Parameter GET *statuses/sample*

delimited	Menentukan apakah pesan harus dibatasi limitnya.
stall_warning	Menentukan apakah pesan warning harus dikirim atau tidak.

GET *statuses/firehose* Mengembalikan semua status publik. Beberapa aplikasi membutuhan akses ini. Teknik ini diolah secara kreatif dengan cara menggabungkan sumber informasi yang ada dengan berbagai sumber lainnya untuk dapat memuaskan pengguna. Perintah ini dikembalikan dalam format JSON, memerlukan otentikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter GET *statuses/firehose* dapat dilihat pada tabel 2.5

Tabel 2.5: Parameter GET *statuses/firehose*

count	Kumpulan pesan untuk dijadikan bahan materi
delimited	Menentukan apakah pesan harus dibatasi limitnya.
stall_warning	Menentukan apakah pesan warning harus dikirim atau tidak.

Menggunakan *Streaming* API Proses menggunakan *streaming* API adalah dengan cara menghubungkan *endpoint* yang sudah tercantum di atas dengan parameter yang sudah di-*list* kepada *streaming endpoint* dan juga *request* parameter *streaming* API.

Koneksi Setiap akun hanya dapat membuat satu koneksi yang terhubung dengan *public endpoint* dan jika melakukan koneksi ke *public stream* lebih dari satu kali dengan menggunakan akun yang sama akan menyebabkan koneksi terlama akan putus. Klien yang membuat koneksi secara berlebihan baik berhasil ataupun tidak maka IP mereka otomatis akan di *banned*.

User Streams *User Stream* memberikan aliran(*stream*) data dan event yang spesifik untuk akun yang sudah diotentikasi. Perintah ini dikembalikan dalam format JSON, memerlukan otentikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk parameter ini dapat dilihat pada tabel 2.6

Endpoints

- GET *user*

Tabel 2.6: Parameter GET *user*

delimited	Menentukan apakah pesan harus dibatasi limitnya.
stall_ warnings	Menentukan apakah pesan *warning* harus dikirim atau tidak.
with	Menentukan apakah pesan informasi harus dikembalikan kepada akun yang sudah diotentikasi atau dilakukan pengiriman juga kepada akun yang di-*follow* oleh akun yang sudah diotentikasi tersebut.
replies	Menentukan apakah harus mengembalikan @replies.
follow	Termasuk *tweet publik* tambahan dari daftar yang disediakan untuk ID pengguna.
track	Termasuk *tweet* tambahan yang cocok dengan kata kunci tertentu.
locations	Termasuk *tweet* tambahan yang termasuk dalam batasan lokasi tertentu.
stringify_friend_ ids	Mengirim list teman yang terdiri dari *array of integer* dan *array of string*.

Koneksi Jika suatu perangkat lunak menggunakan *user stream*, maka sebisa mungkin untuk meminimalkan jumlah koneksi suatu perangkat lunak. Setiap akun Twitter terbatas hanya untuk beberapa koneksi *user stream* per otentikasi perangkat lunak, terlepas dari IP(*Internet Protocol*). Setelah mencapai batasnya, maka koneksi tertua atau terlama akan diberhentikan secara otomatis. *User login* dari beberapa instansi dari otentikasi perangkat lunak yang sama akan mengalami siklus koneksi yaitu akan dihubungan dan diputuskan satu sama lain.

Sebuah aplikasi harus dapat mengatasi HTTP(*The Hypertext Transfer Protocol*) 420 *error code* yang memberitahukan bahwa suatu akun sudah terlalu sering melakukan *login*. Oleh karena itu, akun yang seperti itu akan secara otomatis di-*banned* dari *user stream* untuk tingkat *login* yang berlebihan. Perhatikan bahwa setiap perangkat lunak memiliki alokasinya masing-masing, sehingga *login* dari perangkat lunak yang pertama tidak akan mempengaruhi koneksi untuk perangkat lunak yang kedua, begitu juga sebaliknya. Tetapi akan menimbulkan masalah apabila menjalankan terlalu banyak salinan perangkat lunak yang pertama maupun kedua. Jika anda perlu membuat koneksi

atas nama beberapa akun dari perangkat lunak yang sama, maka akan lebih baik jika menggunakan *site stream*.

2.3 OAuth

Dengan semakin berkembangnya *website*, semakin banyak situs yang bergantung pada layanan distribusi dan *cloud computing*. Contohnya adalah menggunakan jejaring sosial dengan menggunakan akun media sosial lainnya seperti Google untuk mencari teman-teman yang sudah tersimpan pada kontak Google. Atau bisa juga menggunakan pihak ketiga yang memanfaatkan API dari beberapa layanan.

OAuth menyediakan suatu metode bagi pengguna untuk memberi akses pihak ketiga untuk *resources* (sumber daya) client tanpa berbagi *password*. Sebagai contoh, seorang pengguna *website* dapat memberikan layanan percetakan untuk mengakses foto pribadinya yang disimpan di layanan berbagi foto tanpa harus memberikan *username* dan *password*nya. Ia akan mengotentikasi langsung dengan layanan berbagi foto tersebut sehingga dapat dibagikan kepada layanan percetakan.

Agar *client* dapat mengakses *resource*, pertama-tama client harus mendapatkan izin dari si pemilik *resource*. Izin ini dinyatakan dalam bentuk token dan juga digunakan untuk mencocokkan *shared-secret*. Tujuan dari token ini adalah untuk membuat pemilik *resource* berbagi kepercayaan mereka kepada *client*. Token dapat dikeluarkan dalam ruang lingkup terbatas, durasi yang terbatas, dan akan dicabut secara independen[1].

Twitter OAuth yang diberikan memiliki fitur :

- *Secure*

  Pengguna tidak harus berbagi *password* mereka dengan aplikasi pihak ketiga untuk meningkatkan keamanan akun.

- *Standard*

  Banyak *library* dan contoh kode yang tersedia dengan implementasi Twitter Oauth.

Twitter API mengijinkan 350 permintaan OAuth per jam.

2.3.1 *Application-only authentication*

Twitter menawarkan aplikasi yang mampu mengeluarkan permintaan otentikasi atas nama aplikasi itu sendiri. Dengan menggunakan *application-only authentication*, perangkat lunak tidak mempunyai konteks dari otentikasi pengguna dan ini berarti setiap *request* API untuk endpoint akan membutuhkan konteks pengguna, seperti memposting *tweet* tidak akan bekerja. *Application-only authentication* dapat melakukan berbagai macam aktivitas, seperti :

- melihat *timeline*,
- mengakses *following* dan *follower* dari suatu *akun*,

[1] Hueniverse Documentation , OAuth, http://hueniverse.com/oauth/guide/intro/, pada tanggal 20 Agustus 2014 pukul 12.58

- mencari *tweet*,
- mengambil informasi dari akun Twitter manapun.

Tetapi *application-only authentication* tidak dapat melakukan :

- Posting *tweet*
- Melakukan koneksi dengan *Streaming endpoint*
- Mencari akun seseorang
- Menggunakan *geo endpoint*
- Mengakses *Direct Message*

OAuth Flow Langkah-langkah dari *application-only auth* terdiri dari : Sebuah aplikasi dikodekan berdasarkan *consumer key* dan *secret* ke dalam satu set khusus yang dikodekan secara kredensial. aplikasi membuat *request* kepada POST OAuth2/*token endpoint* untuk mengubah kredensial tersebut menjadi *token bearer*. Ketika mengakses REST API, aplikasi menggunakan *token bearer* untuk melakukan otentikasi.

Tentang *Application-only Authentication* Token adalah *password*. *Consumer key* dan *secret, bearer token credential*, dan *the bearer token* memberikan akses untuk membuat permintaan atas nama aplikasi itu sendiri. Poin-poin ini harus dianggap sensitif layaknya *password* dan tidak boleh dibagikan atau didistribusikan kepada pihak yang tidak dipercaya atau tidak berkepentingan.

SSL(*Secure Sockets Layer*) sangat dibutuhkan karena *SSL* merupakan cara otentikasi yang aman. Oleh karena itu, semua *request* (baik untuk mendapatkan atau menggunakan token) harus menggunakan *endpoint* HTTPS, yang juga merupakan syarat untuk menggunakan API.

Request yang dibuat atas nama pengguna tidak akan menguras ketersediaan *rate limit*, begitu juga dengan *request*. Request tidak akan menguras batas penggunaan *limit* dalam *user-based auth*.

2.3.2 3-*legged authorization*

Tahap awal dari cara kerja dari *3-legged authorization* adalah dengan memberikan *access token*. Pengambilan *access token* dilakukan dengan cara melakukan *redirect* akun dengan Twitter. Lalu Twitter memberikan akun sebuah otentikasi dari aplikasi yang telah dibuat. Terdapat dua pengecualian dalam cara kerja dari *3-legged authorization*, yaitu :

- GET *oauth endpoint* digunakan sebagai pengganti GET *oauth*,
- akun akan selalu diminta untuk mengotentikasi perangkat lunak.

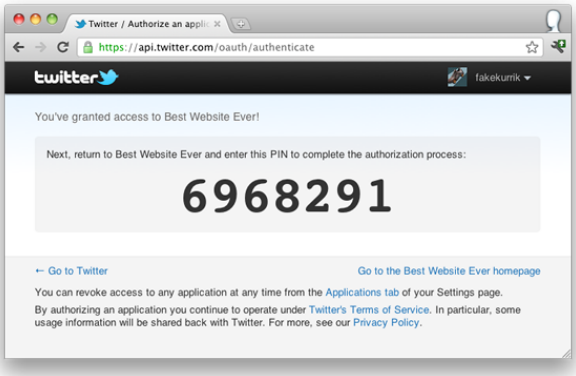
2.3.3 *PIN-based authorization*

PIN-based authorization ditujukan untuk perangkat lunak yang tidak bisa mengakses atau menanamkan *web browser* untuk mengarahkan akun kepada *authorization endpoint*. Contohnya adalah perangkat lunak yang bersifat *command-line*, *embedded systems*, *game* konsol, dan beberapa jenis aplikasi *mobile*.

Implementasi Implementasi *PIN-based authorization* ini memiliki cara kerja yang sama seperti *3-legged authorization*. Perbedaan antara *PIN-based authorization* dengan 3-*legged authorization* terletak pada nilai dari *oauth_callback* yang harus di-set menjadi *oob* saat proses pemanggilan *POST oauth* atau *request_token*.

Setelah perangkat lunak telah mendapatkan *GET oauth/authenticate* atau *GET oauth/authorize URL*, aplikasi akan memberi URL kepada pengguna. URL tersebut dimasukkan oleh pengguna menggunakan *web browser* untuk mengakses URL tersebut.

Ketika *callback oob* diminta, pengguna tidak akan dipindahkan secara otomatis ke perangkat lunak setelah menyetujui akses seperti yang dilakukan 3-*legged authorization*. Akan tetapi jika menggunakan *PIN-based authorization*, pengguna akan melihat kode PIN untuk dikembalikan kepada perangkat lunak dengan cara memasukkan nilai dari kode PIN yang sudah diberikan. Gambar 2.1 merupakan contoh nilai kode yang diberikan.



Gambar 2.1: Contoh PIN-based authorization

Perangkat lunak harus dirancang agar memungkinkan pengguna untuk memasukkan *PIN code*. Nilai dari *PIN code* harus lolos sebagai *oauth_verifier* untuk *POST oauth/access_token request*. Semua *request* akan berjalan normal kedepannya.

2.4 JSON[3]

JSON(*JavaScript Object Notation*) adalah suatu format pertukaran data komputer berbasis teks. JSON digunakan untuk merepresentasikan struktur data sederhana karena formatnya mudah dibaca oleh manusia. Selain itu juga format JSON mudah digunakan oleh suatu perangkat lunak untuk melakukan *parse* dan *generate*. JSON didasarkan pada subset bahasa pemrograman *JavaScript*. JSON dianggap sebagai format data yang tak tergantung pada suatu bahasa. Beberapa bahasa pemograman telah menyediakan *library* untuk pengolahan data JSON.

JSON dibangun dengan dua struktur, yaitu:

1. Nama. Nama ini direalisasikan sebagai objek, *record*, kamus, tabel *hash*, *keyed list*, atau *associative array*.

2. Nilai. Nilai ini direalisasikan sebagai *array*, vektor, *list*, atau *sequence*.

Listing 2.1 menunjukkan representasi JSON untuk suatu objek yang mendeskripsikan seseorang mahasiswa UNPAR.

Listing 2.1: Data Seorang Mahasiswa UNPAR

```
{
    {
     "namaDepan": "Kevin",
     "namaBelakang": "Theodorus",
          "NPM" : "2011730037",
          "Jurusan" : "Teknik Informatika"
     "alamat": {
          "namaJalan": "Jl. Mekar Sederhana 17",
          "kota": "Bandung",
          "kodePos": 40237
     },
     "nomerTelepon": [
          "089676821932",
                  "5233901"
     ]
        }
}
```

Listing 2.1 menunjukan data seorang mahasiswa UNPAR. Berikut ini adalah keterangan dari data seorang mahasiswa UNPAR tersebut:

- Index 0 berisikan nama depan dari mahasiswa UNPAR. Nilai dari nama depan adalah Kevin.
- Index 1 berisikan nama belakang dari mahasiswa UNPAR. Nilai dari nama belakang adalah Theodorus.
- Index 2 berisikan NPM(Nomor Pokok Mahasiswa) dari mahasiswa UNPAR. Nilai dari NPM adalah 2011730037.
- Index 3 berisikan alamat dari mahasiswa UNPAR. Terdapat tiga objek di dalam alamat yaitu nama jalan, kota, dan kode pos. Nama jalan memiliki nilai Jl. Mekar Sederhana 17, kota memiliki nilai Bandung, dan kode pos memiliki nilai 40237.
- Index 4 berisikan nomor telepon dari mahasiswa UNPAR. Nomor telepon memiliki nilai *array*, nilai yang pertama adalah 089676821932 dan nilai yang ke dua adalah 5233901.

2.5 KIRI API[4]

KIRI API adalah aplikasi pihak ketiga yang memungkinkan pengembang perangkat lunak mendapatkan data tentang info jalur transportasi publik. Semua *request* harus berisi API *key* yang dapat diambil melalui KIRI API *Management Dashboard*. Berikut adalah spesifikasi dari KIRI API :

- *Routing Web Service*
- *Search Place Web Service*
- *Nearest Transports Web Service*

2.5.1 *Routing Web Service*

Routing Web Service adalah salah satu KIRI API yang digunakan untuk mendapatkan langkah perjalanan dari lokasi awal menuju lokasi tujuan.

Berikut ini adalah parameter *request* yang diperlukan:

Tabel 2.7: Parameter *Routing Web Service*

Parameter	*Valid values*	*Description*
version	2	Memberitahukan bahwa layanan yang dipakai adalah protokol versi 2
mode	"*findroute*"	Menginstruksikan layanan untuk mencari rute
locale	"en" or "id"	Respons bahasa yang digunakan
start	lat,lng (*both are decimal values*)	Titik awal *Latitude* dan *longitude*
finish	lat,lng (*both are decimal values*)	Titik akhir *Latitude* dan *longitude*
presentation	"*mobile*" or "*desktop*"	Menentukan tipe presentasi untuk hasil keluaran.
apikey	16-digit *hexadecimals*	API *key* yang digunakan

Listing 2.2: kode respon pencarian rute

```
1  {
2      "status": "ok" or "error"
3      "routingresults": [
4          {
5              "steps": [
6                  [
7                      "walk" or "none" or others,
8                      "walk" or vehicle_id or "none",
9                      ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
10                     "human readable description, dependant on locale",
11                     URL for ticket booking or null (future)
12                 ],
13                 [
14                     "walk" or "none" or others,
15                     "walk" or vehicle_id or "none",
16                     ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
17                     "human readable description, dependant on locale",
18                     URL for ticket booking or null (future)
19                 ]
20             ],
21             "traveltime": any text string, null if and only if route is not found.
22         } ,
23         {
24             "steps": [ ... ],
25             "traveltime": "..."
26         } ,
27         {
28             "steps": [ ... ],
29             "traveltime": "..."
30         } ,
31         ...
32     ]
33 }
```

Listing 2.2 menunjukan hasil yang akan diberikan dari pencarian rute. Ketika pencarian rute berhasil, maka status yang diberikan akan bernilai "ok" seperti pada baris 2. Kemudian server harus memberikan hasil dari rute yang berisi langkah-langkah yang disimpan di dalam *array*. Berikut ini adalah keterangan dari *array* tersebut:

- *Index* 0 berisikan "*walk*" atau "*none*" atau "*others*". "*Walk*" berarti jalan kaki, "*none*" berarti rute jalan tidak ditemukan, dan "*others*" berarti menggunakan kendaraan.
- *Index* ke 1 merupakan *detail* dari *index* ke 0 yang memiliki arti:
  - Jika berisikan "*walk*" berarti *index* ini pun harus berisikan "*walk*",
  - Jika berisikan "*none*" maka *index* ini pun harus berisikan "*none*",

  - Selain itu, maka *field* ini berisikan id kendaraan yang dapat digunakan untuk menampilkan gambar dari id kendaraan tersebut.

- *Index* ke 2 berisikan *array of string*, yang berisikan jalur dalam format "lat,lon". Lat adalah *latitude*, dan lon adalah *longitude* yaitu titik awal dan titik akhir.

- *Index* ke 3 merupakan bentuk yang dapat dibaca oleh manusia lalu akan ditampilkan kepada pengguna. Informasi tersebut dapat berupa:

  - %*fromicon* = sebuah *icon* penanda yang menunjukkan titik awal atau "*from*". Biasanya digunakan untuk mode presentasi perangkat bergerak.
  - %*toicon* = sebuah *icon* penanda yang menunjukkan titik akhir atau "*to*". Biasanya digunakan untuk mode presentasi perangkat bergerak.

- *Index* ke 4 berisi URL untuk pemesanan tiket untuk travel jika tersedia. Jika tidak ada maka nilai dari *index* ini bernilai null.

2.5.2 *Search Place Web Service*

Search Place Web Service berguna untuk menemukan rute perjalanan berdasarkan *latitute* dan *longitude* koordinat. Layanan *Search Place Web Service* ini membantu mengubah string teks untuk *latitude* dan *longitude*. Untuk dapat melakukan permintaan rute, berikut parameter *request* yang diperlukan:

Tabel 2.8: Tabel parameter *Search Place Web Service*

version	2	Memberitahukan bahwa layanan yang dipakai adalah protokol versi 2
mode	"*searchplace*"	menginstruksikan layanan untuk mencari tempat
region	"cgk" or "bdo" or "sub"	kota yang akan dicari tempatnya
querystring	text apa saja dengan minimum text satu karakter	*query string* yang akan dicari menggunakan layanan ini
apikey	16-digit *hexadecimals*	API *key* yang digunakan

Berikut format kembalian dari KIRI API:

Listing 2.3: code *respond* pencarian lokasi

```
{
    "status": "ok" or "error"
    "searchresult": [
        {
            "placename": "place name"
            "location": "lat,lon"
        },
        {
            "placename": "place name"
            "location": "lat,lon"
        },
        ...
    ]
    "attributions": [
        "attribution_1", "attribution_2", ...
    ]
}
```

Ketika *request find place* berhasil, *server* akan mengembalikan *place result*. Hasil dari *place result* merupakan *array* dari langkah-langkah perjalanan, berikut adalah contoh dari hasil *place result*:

- *searchresult* - berisi *array* dari hasil objek:
  - *placename* - nama dari suatu tempat
  - *location* : *latitude* dan *longitude* dari suatu tempat
- *attributions* - berisi *array string* dan atribut tambahan yang akan ditampilkan

2.5.3 *Nearest Transports Web Service*

Nearest Transports Web Service digunakan untuk menemukan rute transportasi terdekat dengan titik yang diberikan.

Berikut parameter *request* yang diperlukan berikut penjelasanya:

Tabel 2.9: Tabel parameter *Nearest Transports Web Service*

version	2	Memberitahukan bahwa layanan yang dipakai adalah protokol versi 2
mode	"*nearbytransports*"	menginstruksikan layanan untuk mencari rute transportasi terdekat
start	*latitude* dan *longitude* (keduanya menggunakan nilai desimal)	kota yang akan dicari tempatnya
apikey	16-digit *hexadecimals*	API *key* yang digunakan

Berikut format kembalian dari KIRI API:

Listing 2.4: code *respond* menemukan lokasi terdekat

```
{
    "status": "ok" or "error"
    "nearbytransports": [
        [
            "walk" or "none" or others,
            "walk" or vehicle_id or "none",
            text string,
            decimal value
        ],
        [
            "walk" or "none" or others,
            "walk" or vehicle_id or "none",
            text string,
            decimal value
        ],
        ...
    ]
}
```

Pencarian akan memberitahukan status berhasil ("*ok*") atau tidak ("*error*"). Ketika pencarian sukses, maka respon akan mengembalikan array yang berisikan transportasi terdekat yang diurutkan dari yang paling dekat ke yang paling jauh. Berikut keterangan dari setiap *array* tersebut:

- *Index* ke 0 dapat berisi "*walk*" atau "*none*" atau "*others*". Artinya jika isi dari *array* tersebut "*walk*" berarti berjalan kaki, "*none*" jika rute tidak ditemukan dan "*others*" berarti menggunakan kendaraan.
- *Index* ke 1 merupakan detail dari *index* 0. Artinya jika *index* 0 "*walk*" berarti *index* 1 harus "*walk*", "*none*" berarti *index* 1 harus "*none*" dan selain itu menyatakan id kendaraan yang mana bisa dipakai untuk ditampilkan gambarnya.
- *Index* ke 2 berisi nama kendaraan yang dapat dibaca oleh pengguna.

- *Index* ke 3 berisi jarak dalam satuan kilometer.

2.6 Twitter4J[5]

Twitter4J merupakan *Java Library* untuk Twitter API. Dengan adanya Twitter4J ini, pengguna dapat dengan mudah mengintegrasikan aplikasi Java dengan Twitter *service*. Twitter4J memiliki fitur-fitur sebagai berikut :

- 100% menggunakan Bahasa Java.
- Tersedia untuk *Android platform* dan *Google App Engine*.
- Tidak adanya dependensi, tidak memerlukan *jar* tambahan.
- Mendukung sistem OAuth.
- Kompatibel dengan Twitter API 1.1

Dalam pembuatan aplikasi yang akan penulis buat, penulis membutuhkan beberapa kelas yang telah diberikan oleh Twitter4J. Berikut adalah kelas-kelas yang diberikan Twitter4J :

2.6.1 Twitter

Merupakan kelas *interface* untuk Twitter.

- *Constant*
  - public interface Twitter extends java.io.Serializable, OAuthSupport, OAuth2Support, TwitterBase, TimelinesResources, TweetsResources, SearchResource, DirectMessagesResources, FriendsFollowersResources, UsersResources, SuggestedUsersResources, FavoritesResources, ListsResources, SavedSearchesResources, PlacesGeoResources, TrendsResources, SpamReportingResource, HelpResources
- *Methods*
  - TimelinesResources timelines()
    Merupakan *method* yang digunakan untuk mengembalikan *interface* TimelinesResources.
  - TweetsResources tweets()
    Merupakan *method* yang digunakan untuk mengembalikan *interface* tweets.
  - SearchResource search()
    Merupakan *method* yang digunakan untuk mengembalikan *interface* search.
  - DirectMessagesResources directMessages()
    Merupakan *method* yang digunakan untuk mengembalikan *interface* directMessages.
  - FriendsFollowersResources friendsFollowers()
    Merupakan *method* yang digunakan untuk mengembalikan *interface* friendsFollowers.

- UsersResources users()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* users.
- SuggestedUsersResources suggestedUsers()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* suggestedUsers.
- FavoritesResources favorites()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* favorites.
- ListsResources list()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* list.
- SavedSearchesResources savedSearches()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* savedSearches.
- PlacesGeoResources placesGeo()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* placesGeo.
- TrendsResources trends()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* trends.
- SpamReportingResource spamReporting()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* spamReporting.
- HelpResources help()
  Merupakan *method* yang digunakan untuk mengembalikan *interface* help.

Tidak ada penjelasan yang diberikan oleh Twitter4J

2.6.2 TwitterFactory

Merupakan kelas *final* untuk Twitter Factory.

- *Constant*
  - public final class TwitterFactory extends java.lang.Object implements java.io.Serializable
    Sebuah *factory class* untuk Twitter
- *Constructor*
  - TwitterFactory()
    Membuat TwitterFactory dengan konfigurasi dari sumber.
  - TwitterFactory(Configuration conf)
    Membuat TwitterFactory dengan konfigurasi yang diberikan.
  - TwitterFactory(java.lang.String configTreePath)
    Membuat TwitterFactory yang berasal dari *config tree* yang spesifik.
- *Methods*

  - public Twitter getInstance()
    mengembalikan contoh yang terkait dengan konfigurasi.
  - public Twitter getInstance(AccessToken accessToken)
    mengembalikan OAuth yang sudah otentikasi.
  - public Twitter getInstance(Authorization auth)
  - public static Twitter getSingleton()
    Mengembalikan *singleton* standar Twitter *instance*.

2.6.3 TwitterStream

Merupakan kelas *interface* untuk melakukan *streaming*.

- *Constant*
  - public interface TwitterStream extends OAuthSupport, TwitterBase
    Sebuah *factory class* untuk Twitter
- *Methods*
  - void addConnectionLifeCycleListener(ConnectionLifeCycleListener listener)
    Menambahkan *ConnectionLifeCycleListener*
  - void addListener(StreamListener listener)
    Menambahkan listener.
  - void removeListener(StreamListener listener)
    Menghilangkan listerner.
  - void clearListeners()
    Menghilangkan *status listener*.
  - void replaceListener(StreamListener toBeRemoved,StreamListener toBeAdded)
    Menimpa listener yang sudah ada sebelumnya.
  - void firehose(int count)
    Mendengarkan semua status publik.
  - void links(int count)
    Mendengarkan semua status publik yang mengandung link.
  - void retweet()
    Mendengarkan semua *retweet*.
  - void sample()
    Mendengarkan status publik secara acak.
  - void user()
    User Streams menyediakan update dari semua data secara *real-time*.

- void user(java.lang.String[] track)
  User Streams menyediakan update dari semua data secara *real-time*. Parameter track merupakan kata kunci untuk kata yang akan ditampilkan.
- StreamController site(boolean withFollowings, long[] follow)
  Menerima update secara *real-time* untuk sejumlah pengguna tanpa perlu kerepotan dalam mengelola REST API *rate limits*.
- void filter(FilterQuery query)
  Menerima status publik yang telah di *filter* dari satu atau lebih kata kunci.
- void cleanUp()
  Menon-aktifkan penggunaan *thread stream*.
- void shutdown()
  Menon aktifkan *dispatcher thread* bersama dengan semua instansi TwitterStream.

2.6.4 TwitterStreamFactory

Merupakan kelas final untuk Twitter *Stream Factory*.

- *Constant*
  - public final class TwitterStreamFactory extends java.lang.Object implements java.io.Serializable
    Sebuah *factory class* untuk Twitter. Instansi dari kelas ini memiliki thread yang aman dan digunakan secara berkala lalu dapat digunakan kembali.
- *Constructor*
  - TwitterStreamFactory() Membuat TwitterStreamFactory dengan konfigurasi dari sumber.
  - TwitterStreamFactory(Configuration conf) Membuat TwitterStreamFactory dengan konfigurasi yang diberikan.
  - TwitterStreamFactory(java.lang.String configTreePath) Membuat TwitterStreamFactory yang berasal dari *config tree* yang spesifik.
- *Methods*
  - public TwitterStream getInstance()
    Mengembalikan contoh yang terkait dengan konfigurasi.
  - public TwitterStream getInstance(AccessToken accessToken)
    Mengembalikan OAuth yang sudah diotentikasi.
  - public TwitterStream getInstance(Authorization auth)
    Mengembalikan *instance*.
  - private TwitterStream getInstance(Configuration conf, Authorization auth)
    Mengembalikan *instance* dengan konfigurasi dan autorisasi yang sesuai.
  - public static Twitter getSingleton()
    Mengembalikan *singleton* standar Twitter *instance*.

2.6.5 StatusListener

Merupakan kelas *interface* untuk *status listener*

- *Constant*
  - public interface StatusListener extends StreamListener
- *Methods*
  - void onStatus(Status status)
  - void onDeletionNotice(StatusDeletionNotice statusDeletionNotice)
    Method untuk memberitahukan notifikasi deletionNotice.
  - void onTrackLimitationNotice(int numberOfLimitedStatuses)
    Method untuk memberitahukan bahwa predikat terlalu luas.
  - void onScrubGeo(long userId, long upToStatusId)
    Method untuk memberitahukan *location deletion*.
  - void onStallWarning(StallWarning warning)
    Method untuk memberitahukan pesan *warning*.

2.6.6 StatusUpdate

Merupakan kelas untuk melakukan *update* status

- *Constant*
  - public final class StatusUpdate extends java.lang.Object implements java.io.Serializable
- *Field*
  - private boolean displayCoordinates
  - private long inReplyToStatusId
  - private GeoLocation location
  - private java.io.InputStream mediaBody
  - private java.io.File mediaFile
  - private long[] mediaIds
  - private java.lang.String mediaName
  - private java.lang.String placeId
  - private boolean possiblySensitive
  - private static long serialVersionUID
  - private java.lang.String status
- *Methods*

- private void appendParameter(java.lang.String name, double value, java.util.List<HttpParameter> params)
- private void appendParameter(java.lang.String name, long value, java.util.List<HttpParameter> params)
- private void appendParameter(java.lang.String name, java.lang.String value, java.util.List<HttpParamet params)
- boolean equals(java.lang.Object o)
- long getInReplyToStatusId()
  Merupakan *getter* untuk atribut inReplyToStatusId.
- GeoLocation getLocation()
  Merupakan *getter* untuk atribut location.
- java.lang.String getPlaceId()
  Merupakan *getter* untuk atribut placeId.
- java.lang.String getStatus()
  Merupakan *getter* untuk atribut status.
- boolean isDisplayCoordinates()
  Merupakan *getter* untuk atribut displayCoordinates.
- void setDisplayCoordinates(boolean displayCoordinates)
  Merupakan *setter* untuk atribut displayCoordinates.
- void setInReplyToStatusId(long inReplyToStatusId)
  Merupakan *setter* untuk atribut inReplyToStatusId.
- void setLocation(GeoLocation location)
  Merupakan *setter* untuk atribut location.
- void setMedia(java.io.File file)
  Merupakan *setter* untuk atribut mediaFile.
- void setMedia(java.lang.String name, java.io.InputStream body)
  Merupakan *setter* untuk atribut mediaFile.
- void setMediaIds(long[] mediaIds)
  Merupakan *setter* untuk atribut mediaIds.
- void setPlaceId(java.lang.String placeId)
  Merupakan *setter* untuk atribut placeId.
- void setPossiblySensitive(boolean possiblySensitive)
  Merupakan *setter* untuk atribut possiblySensitive.
- java.lang.String toString()
  Merupakan *method* untuk mengubah *status update* ke dalam bentuk *string*

Tidak ada penjelasan yang diberikan oleh Twitter4J

2.6.7 TweetsResources

- *Constant*
  - public interface TweetsResources
- *Methods*
  - ResponseList<Status> getRetweets(long statusId) throws TwitterException
    Mengembalikan sampai dengan 100 *retweet* pertama yang diberikan.
  - IDs getRetweeterIds(long statusId, long cursor) throws TwitterException
    Mengembalikan sampai dengan 100 ID pengguna yang telah melakukan *retweet* oleh parameter ID tertentu
  - IDs getRetweeterIds(long statusId, int count, long cursor) throws TwitterException
    Mengembalikan sampai dengan "*count*" ID pengguna yang telah melakukan *retweet* oleh parameter ID tertentu
  - Status showStatus(long id) throws TwitterException
    Mengembalikan *single status* yang ditentukan oleh parameter ID yang telah ditentukan.
  - Status destroyStatus(long statusId) throws TwitterException
    Menghapus status yang ditentukan oleh parameter ID yang telah ditentukan.
  - Status updateStatus(java.lang.String status) throws TwitterException
    Melakukan update status oleh user yang telah diotentikasi
  - Status updateStatus(StatusUpdate latestStatus) throws TwitterException
    Melakukan update status oleh user yang telah diotentikasi.
  - Status retweetStatus(long statusId) throws TwitterException
    Melakukan *retweet*.
  - OEmbed getOEmbed(OEmbedRequest req) throws TwitterException Mengembalikan informasi yang dapat merepresentasikan *third party Tweet*
  - ResponseList<Status> lookup(long[] ids) throws TwitterException
    Mengembalikan *fully-hydrated tweet objects* sampai dengan 100 *tweet* setiap *request*nya.
  - UploadedMedia uploadMedia(java.io.File mediaFile) throws TwitterException
    Melakukan *upload* media gambar yang telah di dilampirkan via updateStatus(twitter4j.StatusUpdate)

2.6.8 OAuthSupport

Merupakan kelas untuk membantu proses otentikasi.

- *Constant*
  - public interface OAuthSupport
- *Methods*

- void setOAuthConsumer(java.lang.String consumerKey, java.lang.String consumerSecret)

  Melakukan pengaturan terhadap *consumer key* dan *consumer secret* .

- RequestToken getOAuthRequestToken() throws TwitterException

  Mengambil *request token.*

- RequestToken getOAuthRequestToken(java.lang.String callbackURL) throws TwitterException

  Mengambil *request token.*

- RequestToken getOAuthRequestToken(java.lang.String callbackURL, java.lang.String xAuthAccessType) throws TwitterException

  Mengambil *request token.*

- AccessToken getOAuthAccessToken() throws TwitterException

  Mengembalikan *access token* yang terkait dengan instansi ini. Jika tidak ada instansi pada *access token* maka akan mengambil *access token* yang baru.

- AccessToken getOAuthAccessToken(java.lang.String oauthVerifier) throws TwitterException

  Mengambil *request token.*

- AccessToken getOAuthAccessToken(RequestToken requestToken) throws TwitterException

  Mengambil *access token* yang terkait dengan *request token* dan *userId* yang telah diberikan

- AccessToken getOAuthAccessToken(RequestToken requestToken, java.lang.String oauthVerifier) throws TwitterException

  Mengambil *access token* yang terkait dengan *request token* dan *userId* yang telah diberikan

- AccessToken getOAuthAccessToken(java.lang.String screenName, java.lang.String password) throws TwitterException

  Mengambil *access token* yang terkait dengan *screen name*dan *password* yang telah diberikan

- void setOAuthAccessToken(AccessToken accessToken)

  Melakukan pengaturan pada *access token*

2.6.9 Status

Merupakan kelas *interface* untuk status.

- *Constant*

  - public interface Status extends java.lang.Comparable<Status>, TwitterResponse, EntitySupport, java.io.Serializable

- *Methods*

- java.util.Date getCreatedAts()

  Mengembalikan *created_at*

- public long getUserId()

  Mengembalikan *user id*

- java.lang.String getText()

  Mengembalikan teks dari status

- java.lang.String getSource()

  Mengembalikan *source*

- boolean isTruncated()

  Menguji apakah sebuah status terpotong atau tidak

- long getInReplyToStatusId()

  Mengembalikan *in_reply_tostatus_id*

- long getInReplyToUserId()

  Mengembalikan *in_reply_user_id*

- java.lang.String getInReplyToScreenName()

  Mengembalikan *in_reply_to_screen_name*

- GeoLocation getGeoLocation()

  Mengembalikan lokasi dari suatu *tweet* jika tersedia.

- Place getPlace()

  Mengembalikan tempat yang terdapat pada sebuah status.

- boolean isFavorited()

  Menguji apakah status tersebut *favorite* atau tidak

- boolean isRetweeted()

  Menguji apakah status tersebut *retweet* atau tidak

- int getFavoriteCount()

  Menunjukkan berapa kali *Tweet* telah menjadi *favorite*

- User getUser()

  Mengembalikan *user* yang terdapat pada sebuah status.

- boolean isRetweet()

- Status getRetweetedStatus()

- long[] getContributors()

  Mengembalikan array yang berisi kontributor atau mengembalikan *null* jika tidak ada kontributor yang terkait dengan status ini

- int getRetweetCount()

  Menunjukkan berapa kali *tweet* telah di *retweet*, jika belum terdapat maka akan mengembalikan nilai -1

- boolean isRetweetedByMe()

  Mengembalikan nilai *true* jika *user* yang telah diotentikasi melakukan *retweet* terhadap suatu *tweet*, atau mengembalikan nilai *false* jika tidak.

- long getCurrentUserRetweetId()

  Mengembalikan *retweet id* sebuah *tweet* dari *user* yang telah diotentikasi, jika belum terdapat maka akan mengembalikan nilai -1L

- boolean isPossiblySensitive()

  Mengembalikan nilai *true* jika pada status terdapat *sensitive links*

- java.lang.String getLang()

  Mengembalikan *lang* dari sebuah status teks jika tersedia

- Scopes getScopes()

  Mengembalikan target dari *scopes* yang diaplikasikan kepada sebuah status.

2.7 Twitter4J *Properties*

Untuk menggunakan Twitter4J diperlukan *properties* untuk proses konfigurasi. Konfigurasi dapat dilakukan dengan cara membuat *file* twitter4j.properties atau kelas *ConfigurationBuilder* atau *System Property*. Berikut adalah contoh penggunaan dari ketiganya :

1. via twitter4j.properties

   Menyimpan standar *properties file* yang diberi nama "twitter4j.properties". *File* ini diletakkan pada *folder* yang sama dengan pembuatan perangkat lunak.

   Listing 2.5: isi dari twitter4j.properties

```
1  {
2      debug=false
3      oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIUl
4      oauth.consumerSecret=YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7t313
5      oauth.accessToken=313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
6      oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptcuR4i
7  }
```

2. via *ConfigurationBuilder*

   Menggunakan *ConfigurationBuilder class* untuk melakukan konfigurasi Twitter4J.

   Listing 2.6: isi dari twitter4j ConfigurationBuilder

```
1  {
2      ConfigurationBuilder cb = new ConfigurationBuilder();
3      cb.setDebugEnabled(true)
4          .setOAuthConsumerKey("3iT8duMItTTrdaU1qTHxwDIUl")
5          .setOAuthConsumerSecret("YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7t313")
6          .setOAuthAccessToken("313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ")
7          .setOAuthAccessTokenSecret("LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptcuR4i");
8      TwitterFactory tf = new TwitterFactory(cb.build());
9      Twitter twitter = tf.getInstance();
10 }
```

3. via *System Properties*

   Menggunakan *System Properties* untuk melakukan konfigurasi Twitter4J.

Listing 2.7: isi dari twitter4j System Properties

```
1 $ export twitter4j.debug=true
2 $ export twitter4j.oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIUl
3 $ export twitter4j.oauth.consumerSecret=YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7t313
4 $ export twitter4j.oauth.accessToken=313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
5 $ export twitter4j.oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptcuR4i
6 $ java -cp twitter4j-core-4.0.2.jar:yourApp.jar yourpackage.Main
```

BAB 3

ANALISIS

Pada bab ini akan dibahas mengenai analisis Twitter API, OAuth, KIRI API, Twitter4J, Spesifikasi kebutuhan fungsional, Diagram *Use Case*, dan *Diagram Class*.

3.1 Analisis Data

Pada sub bab ini, akan dilakukan analisa tentang Twitter API, OAuth, KIRI API, dan Twitter4j. Setelah membaca dan menganalisis maka penulis akan menentukan hal-hal yang akan digunakan dalam membangun *Twitter bot* untuk mencari jalur transportasi publik.

3.1.1 Analisis Twitter API

Setelah melakukan analisis, perangkat lunak yang akan dibangun menggunakan *Streaming* API karena:

- Streaming API adalah *real-time* API, sedangkan *search* API hanya dapat menangkap *tweet* setiap beberapa waktu sekali. Pada perangkat lunak yang akan dibuat skenarionya adalah pengguna akan menanyakan rute transportasi publik dalam bentuk *tweet* yang dikirimkan kepada akun *Twitter bot*, dalam skenario seperti ini dibutuhkan jawaban yang *real-time*.

- *Endpoint streaming* menggunakan *public stream*. *Public Stream* mengambil semua data publik, sehingga semua *tweet* bisa ditangkap oleh perangkat lunak. Dalam pembuatan *Twitter Bot* untuk mencari jalur transportasi publik, pengguna akan melakukan *mention tweet* kepada akun *Twitter bot* untuk dapat memperoleh balasan *tweet* yang berisi hasil pencarian jalur transportasi publik. *Public Stream* mempunyai fitur bernama *track*. Fitur *track* berguna untuk menyaring *tweet* berdasarkan *keyword* tertentu. *Keyword* yang akan di-*track* adalah nama akun dari *Twitter bot*, jadi perangkat lunak hanya menerima *tweet* yang di-*mention* kepada akun *Twitter bot* saja. *User stream* mengandung semua data yang berhubungan dengan satu akun tertentu seperti *update status*, *mention*, dan *direct message*. Dalam kasus ini bisa saja menggunakan *user stream* tetapi penggunaannya kurang efisien. Pengambilan *tweet update status* dan *direct message* tidak dibutuhkan dalam pembuatan perangkat lunak *Twitter bot*. *Site stream* merupakan versi *multi-user stream*. Dalam kasus *Twitter bot* untuk mencari jalur transportasi publik ini akun yang digunakan untuk menjadi *Twitter bot* hanya satu akun saja. Jadi penggunaan *site stream* dalam kasus ini kurang efisien.

3.1.2 Analisis OAuth

Setelah melakukan analisis, OAuth yang digunakan dalam pembuatan *Twitter bot* untuk mencari jalur transportasi publik adalah *3-legged authorization*. Penggunaan *3-legged authorization* ini digunakan untuk melakukan otentikasi akun *Twitter bot*. Proses otentikasi tidak perlu dilakukan kepada pengguna, karena *Twitter bot* yang dibuat menggunakan otentikasi langsung dari pengembang perangkat lunak. *Application-only authentication* tidak bisa digunakan karena *application-only authentication* tidak dapat melakukan *posting tweet* dan tidak dapat melakukan koneksi dengan *streaming endpoint*. Sedangkan dalam kasus *Twitter bot* untuk mencari jalur transportasi publik dibutuhkan otentikasi yang dapat memposting *tweet* dan melakukan koneksi dengan *streaming endpoint*. Penggunaan otentikasi *PIN-based authorization* tidak cocok, karena otentikasi sudah dilakukan langsung dari pengembang perangkat lunak. Maka dari itu tidak diperlukan PIN untuk proses otentikasi.

3.1.3 Analisis KIRI API

KIRI API menyediakan tiga layanan yang dapat digunakan. *Twitter bot* yang akan dibangun membutuhkan dua layanan yang diberikan KIRI API. Layanan tersebut adalah *Routing Web Service* dan *Search Place Web Service*. *Routing Web Service* adalah layanan yang digunakan untuk mendapatkan langkah perjalanan dari lokasi awal menuju lokasi tujuan. Sedangkan *Search Place Web Service* berguna untuk menemukan rute perjalanan berdasarkan *latitute* dan *longitude* koordinat. Layanan *Search Place Web Service* ini membantu mengubah *latitude* dan *longitude* ke-dalam bentuk string.

Untuk setiap permintaan terhadap KIRI API dibutuhkan *API key*. *API key* ini sendiri berguna sebagai *password* untuk mengakses KIRI API. *API key* ini sendiri dapat didapatkan di https://dev.kiri.travel/bukitjarian/. Dalam pembuatan *Twitter bot* untuk mencari jalur transportasi publik ini, KIRI memberikan *API key* khusus yaitu 889C2C8FBB82C7E6.

Berikut adalah contoh pemanfaatan KIRI API :

- *Search Place Web Service*

  Format *Search Place Web Service* yang dikirim melalui URL adalah kiri.travel\/handle.php?version=2&mode=searchplace&region=cgk\/bdo\/sub&querystring="string"&apikey=889C2C8FBB82C7E6.

  Parameter yang dikirimkan adalah :

  1. version : 2

     Version 2 merupakan versi KIRI API yang terbaru. Oleh karena itu, penulis akan menuliskan parameter version dengan nilai 2.

  2. mode : "searchplace"

     Mode "searchplace" merupakan mode dari *Search Place Web Service* yang digunakan untuk mencari lokasi.

  3. region : bdo

     Region berfungsi sebagai parameter untuk memberitahukan kota yang akan menjadi bagian dalam pencarian lokasi. Parameter yang terdapat di region ada tiga yaitu "cgk" untuk Kota Jakarta, "bdo" untuk Kota Bandung, dan "sub" untuk Kota Surabaya.

4. querystring

   Merupakan kata kunci untuk lokasi.

5. apikey : 889C2C8FBB82C7E6

   Merupakan *password* yang digunakan untuk mengakses KIRI API.

Penulis mencoba mencari lokasi pvj dari kata kata kunci "pvj" yang berada di Kota Bandung. Layanan dikirimkan ke URL kiri.travel/handle.php. Berikut adalah format layanan yang dituliskan: http://kiri.travel/handle.php?version=2&mode=searchplace&region=bdo&querystring=pvj&apikey=889C2C8FBB82C7E6

Berikut adalah hasil kembalian dari KIRI API:

Listing 3.1: hasil kembalian dari *Search Place Web Service*

```
{
        "status":"ok",
        "searchresult":[
                {
                    "placename":"J.Co Donuts & Coffee",
                    "location":"-6.88929,107.59574"
                },
                {
                    "placename":"Pepper Lunch Bandung (PVJ)",
                    "location":"-6.88923,107.59615"
                },
                {
                    "placename":"Domino's Pizza Pvj",
                    "location":"-6.90348,107.61709"
                },
                {
                    "placename":"Outlet Alleira Batik PVJ Bandung",
                    "location":"-6.88875,107.59634"
                },
                {
                    "placename":"Burger King Bandung PVJ Mall",
                    "location":"-6.88894,107.59342"
                },
                {
                    "placename":"Killiney Kopitiam PVJ",
                    "location":"-6.88947,107.59654"
                },
                {
                    "placename":"Adidas Pvj",
                    "location":"-6.88909,107.59614"
                },
                {
                    "placename":"Crocs - PVJ",
                    "location":"-6.88894,107.59342"
                },
                {
                    "placename":"Cross Pvj",
                    "location":"-6.88906,107.59619"
                },
                {
                    "placename":"Jonas Photo - PVJ",
                    "location":"-6.88913,107.59643"
                }
            ],
            "attributions":null
}
```

- *Routing Web Service*

  Format *Routing Web Service* yang dikirim melalui URL adalah kiri.travel/handle.php?version=2&mode=findroute&locale=en/id&start=lat,lng&finish=lat,lng&presentation=mobile/desktop&apikey=889C2C8FBB82C7E6.

  Parameter yang dikirimkan adalah :

1. version : 2

   Version 2 merupakan versi KIRI API yang terbaru. Oleh karena itu, penulis akan menuliskan parameter version dengan nilai 2.

2. mode : "findroute"

   Mode "findroute" merupakan mode dari *Routing Web Service* yang digunakan untuk mendapatkan langkah yang harus dilakukan dari lokasi awal menuju lokasi tujuan.

3. locale : id

   locale berfungsi sebagai parameter untuk bahasa yang digunakan. Karena target dari perangkat lunak ini adalah orang Indonesia, maka parameter locale menggunakan "id" untuk Bahasa Indonesia, jika ingin menggunakan Bahasa Ingris maka menggunakan parameter "en".

4. start

   Merupakan koordinat awal. Parameter ini berupa latitude dan longitude.

5. finish

   Merupakan koordinat tujuan. Parameter ini berupa latitude dan longitude.

6. presentation : "desktop"

   Parameter *presentation* ini terdapat dua jenis yaitu "mobile" untuk perangkat bergerak dan "desktop" untuk komputer. Perbedaan mobile dan desktop terletak pada *icon* yang diberikan. Jika menggunakan presentation "mobile" maka hasil kembalian akan terdapat %toicon dan %fromicom, hasil tersebut tidak dibutuhkan oleh pengguna karena pengguna *Twitter bot* tidak dapat melihat map jalur transportasi publik yang diberikan.

7. apikey : 889C2C8FBB82C7E6

   Merupakan password yang digunakan untuk mengakses KIRI API.

Penulis mencoba mencari rute perjalanan dari PVJ(Paris van Java) menuju BIP(Bandung Indah Plaza). Layanan dikirimkan ke URL `kiri.travel/handle.php`. Berikut adalah format layanan yang dituliskan: `http://kiri.travel/handle.php?version=2&mode=findroute&locale=en&start=-6.88923,107.59615&finish=-6.90864,107.61108&presentation=desktop&apikey=889C2C8FBB82C7E6`.

Berikut adalah hasil kembalian dari KIRI API:

Listing 3.2: hasil kembalian dari *Routing Web Service*

```
{
        "status":"ok",
        "routingresults":[
        {
            "steps":[
                [
                    "walk",
                    "walk",
                    ["-6.88923,107.59615","-6.88958,107.59691"],
                    "Walk about 92 meter from your starting point to Jalan Sukajadi.",
                    null
                ],
                [
                    "angkot",
                    "kalapakarangsetra",
                    ["-6.88958,107.59691","-6.89052,107.59696","-6.89146,107.59701",
                    "-6.89239,107.59706","-6.89333,107.59711","-6.89333,107.59711",
```

```
18                          "-6.89466,107.59719","-6.89598,107.59727","-6.89598,107.59727",
19                          "-6.89700,107.59731","-6.89801,107.59735","-6.89903,107.59740",
20                          "-6.90005,107.59744","-6.90005,107.59744","-6.90113,107.59747",
21                          "-6.90222,107.59751","-6.90331,107.59754","-6.90439,107.59757",
22                          "-6.90439,107.59757","-6.90540,107.59760","-6.90641,107.59763",
23                          "-6.90641,107.59763","-6.90650,107.59781","-6.90667,107.59887",
24                          "-6.90684,107.59992","-6.90684,107.59992","-6.90690,107.60086",
25                          "-6.90696,107.60179","-6.90696,107.60179","-6.90704,107.60306",
26                          "-6.90711,107.60433"],
27                          "Take angkot Kalapa - Karang Setra at Jalan Sukajadi, and alight at Jalan
                                Pajajaran about 2.6 kilometer later.",
28                          null
29                      ],
30                      [
31                          "angkot",
32                          "ciroyomantapani",
33                          ["-6.90713,107.60441","-6.90713,107.60441","-6.90679,107.60440",
34                          "-6.90563,107.60438","-6.90448,107.60435","-6.90448,107.60435",
35                          "-6.90429,107.60448","-6.90422,107.60487","-6.90403,107.60527",
36                          "-6.90397,107.60564","-6.90402,107.60608","-6.90436,107.60671",
37                          "-6.90488,107.60725","-6.90522,107.60749","-6.90588,107.60771",
38                          "-6.90625,107.60772","-6.90642,107.60783","-6.90658,107.60806",
39                          "-6.90678,107.60929","-6.90678,107.60929","-6.90685,107.60939",
40                          "-6.90787,107.60939","-6.90889,107.60939","-6.90889,107.60939",
41                          "-6.90913,107.60920","-6.90918,107.60878","-6.90924,107.60847",
42                          "-6.90934,107.60843","-6.91008,107.60880","-6.91026,107.60890",
43                          "-6.91030,107.60905","-6.91029,107.60923","-6.91020,107.60951",
44                          "-6.90976,107.61056","-6.90976,107.61056","-6.90974,107.61091"],
45                          "Take angkot Ciroyom - Antapani at Jalan Pajajaran, and alight at Jalan Aceh
                                about 1.7 kilometer later.",
46                          null
47                      ],
48                      [
49                          "walk",
50                          "walk",
51                          ["-6.90974,107.61091","-6.90864,107.61108"],
52                          "Walk about 124 meter from Jalan Aceh to your destination.",
53                          null
54                      ]
55                      ],
56                          "traveltime":"25 minutes"
57                      }
58                  ]
59      }
```

3.1.4 Analisis Twitter4J

Setelah melakukan analisis, kelas yang digunakan untuk membuat *Twitter bot* untuk mencari jalur transportasi publik terdiri dari :

- Twitter
- StatusListener
- StatusUpdate
- TwitterFactory
- TwitterStream

Menggunakan *file* twitter4j.properties. Penggunaan twitter4j.properties lebih praktis dalam pemakaiannya karena memiliki file tersendiri khusus untuk menyimpan *consumerKey*, *consumerSecret*, *accessToken*, dan *accessTokenSecret*.

3.1.5 Analisis *Tweet*

Setelah melakukan analisis, *username* dari Twitter memiliki minimal 3 karakter dan maksimal 15 karakter. Dikarenakan keterbatasan 140 karakter pada setiap *tweet*, maka diperlukan algoritma

untuk memecah-mecah instruksi dari KIRI dan membuatnya ke dalam bentuk *tweet*. *Routing web service* pada KIRI API akan mengembalikan rute transportasi publik dari lokasi awal menuju lokasi tujuan dalam bentuk JSON. Terdapat kemungkinan bahwa hasil JSON dari KIRI API setelah ditambah dengan *username* pengguna *Twitter bot* akan lebih dari 140 karakter. Berikut adalah penjelasan dari algoritma untuk memecah *tweet*.

1. Menghitung panjang maksimal dari setiap *tweet* setelah dikurangi dengan panjang *username* pengguna. Contohnya adalah akun @infobdg dan akun @kviniinktest123, panjang *username* @infobdg memiliki panjang kata sepanjang tujuh huruf, sedangkan panjang *username* @kviniinktest123 memiliki panjang kata sepanjang 15 huruf. Tentu saja kedua akun ini memiliki batas kata *tweet* yang berbeda, oleh karena itu perhitungan panjang maksimal *tweet* dilakukan dengan cara mengurangi batas maksimal *tweet* dengan panjang *username* pengguna lalu dikurangi dua untuk simbol @ didepan kata *username* dan spasi diakhir kata username. Contoh panjang maksimal *tweet* untuk akun @infobdg adalah 140 dikurangi 7 dikurangi 2, maka menghasilkan 131 karakter. Sedangkan panjang maksimal *tweet* untuk akun @kviniinktest123 adalah 140 dikurangi 15 dikurangi 2, maka menghasilkan 123 karakter.

2. Membuat hasil kembalian dari KIRI API menjadi *array* dari kata. Sebagai contoh terdapat kalimat "Jalan sedikit di Jalan Pasirkaliki.", maka kalimat tersebut akan dibuat menjadi *array* dari kata yang memiliki lima panjang *array*.

3. Melakukan *looping* untuk memasukan *array of* kata kedalam variable status. Di dalam *looping* tersebut akan dilakukan pengecekan apakah *variable* status sudah melebihi panjang maksimal *tweet* atau belum. Jika belum maka kata akan ditambahkan kedalam *variable* status, jika sudah melebihi maka akan dibuat *tweet* baru.

3.2 Analisis Perangkat Lunak

Perangkat lunak yang dibangun adalah *Twitter bot* untuk mencari jalur transportasi publik. *Twitter bot* yang dibangun dapat membalas *tweet* secara *real-time* kepada pengguna untuk memberitahukan jalur-jalur yang harus ditempuh menggunakan transportasi publik. Perangkat lunak yang digunakan untuk membangun *Twitter Bot* Untuk Mencari Jalur Transportasi Publik adalah NetBeans IDE 8.0.2 dan akun yang digunakan untuk pengujian *Twitter bot* adalah akun @kviniink. Pada sub bab ini akan dibahas kebutuhan aplikasi, diagram *use case*, skenario, dan *diagram class* dari perangkat lunak yang akan dibangun.

3.2.1 Spesifikasi Kebutuhan Fungsional

Spesifikasi kebutuhan perangkat lunak yang akan dibangun untuk membuat *Twitter bot* adalah

1. Dapat melakukan otentikasi untuk akun *Twitter bot* yang digunakan.

2. Dapat menerima dan membaca *tweet* yang di *mention* kepada akun *Twitter bot* @kviniink

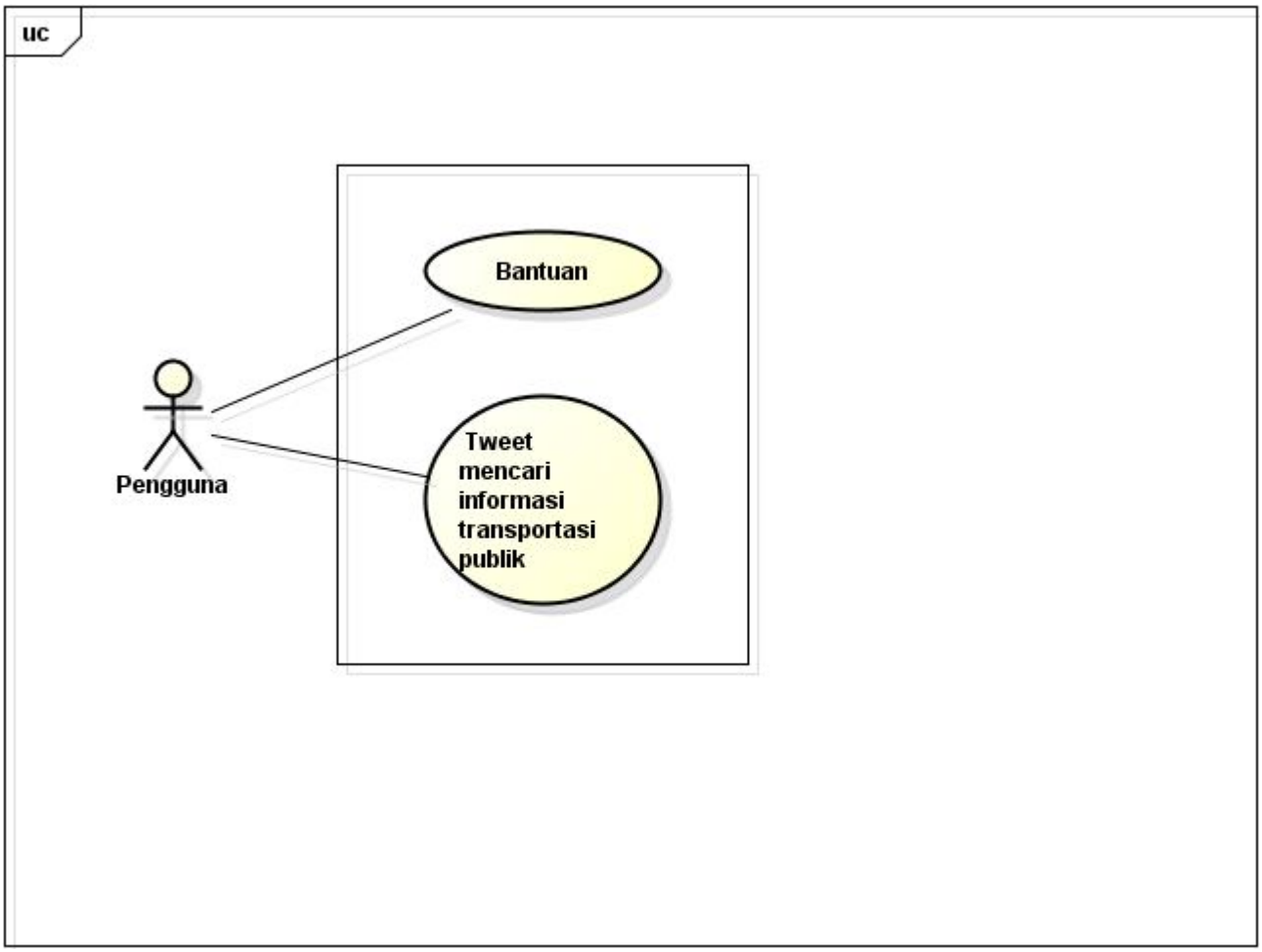
3. Dapat melakukan proses pencarian koordinat suatu lokasi

4. Dapat melakukan proses pencarian jalur transportasi publik dari lokasi awal menuju lokasi tujuan

5. Dapat membalas *tweet* pencarian jalur transportasi publik yang diterima oleh *Twitter bot* dengan melakukan *reply tweet* yang berisi hasil pencarian jalur transportasi publik dengan format yang sudah ditentukan.

3.2.2 *Use Case Diagram*

Perangkat lunak yang dibangun akan memiliki satu figur utama, yaitu *tweet* mencari informasi transportasi publik. Gambar 3.1 menunjukkan diagram *use case* dari perangkat lunak.



Gambar 3.1: Use case Twitter Bot

Skenario *Use Case* Skenario ini hanya memiliki satu aktor yaitu pengguna. *Tweet* mencari informasi transportasi publik pada skenario ini dilakukan dengan melakukan *tweet* kepada akun *Twitter bot* @kviniink berisi format yang sesuai untuk pencarian rute transportasi. Pengguna dapat melakukan *tweet* bantuan kepada akun *Twitter bot* @kviniink untuk mengetahui format pencarian rute transportasi publik.

Tabel 3.1: Skenario *Tweet* mencari informasi transportasi

Nama	*Tweet* mencari informasi transportasi publik
Aktor	Pengguna
Deskripsi	Melakukan *Tweet* (*Tweet* berupa lokasi asal dan lokasi tujuan)
Kondisi Awal	Belum menuliskan *Tweet* pada kolom *update*
Kondisi Akhir	Sudah melakukan *Tweet* kepada user @kviniink
Skenario Utama	Pengguna melakukan *Tweet* kepada *user* @kviniink dengan format yang sudah ditentukan
Eksepsi	Format penulisan salah

Tabel 3.2: Skenario Bantuan

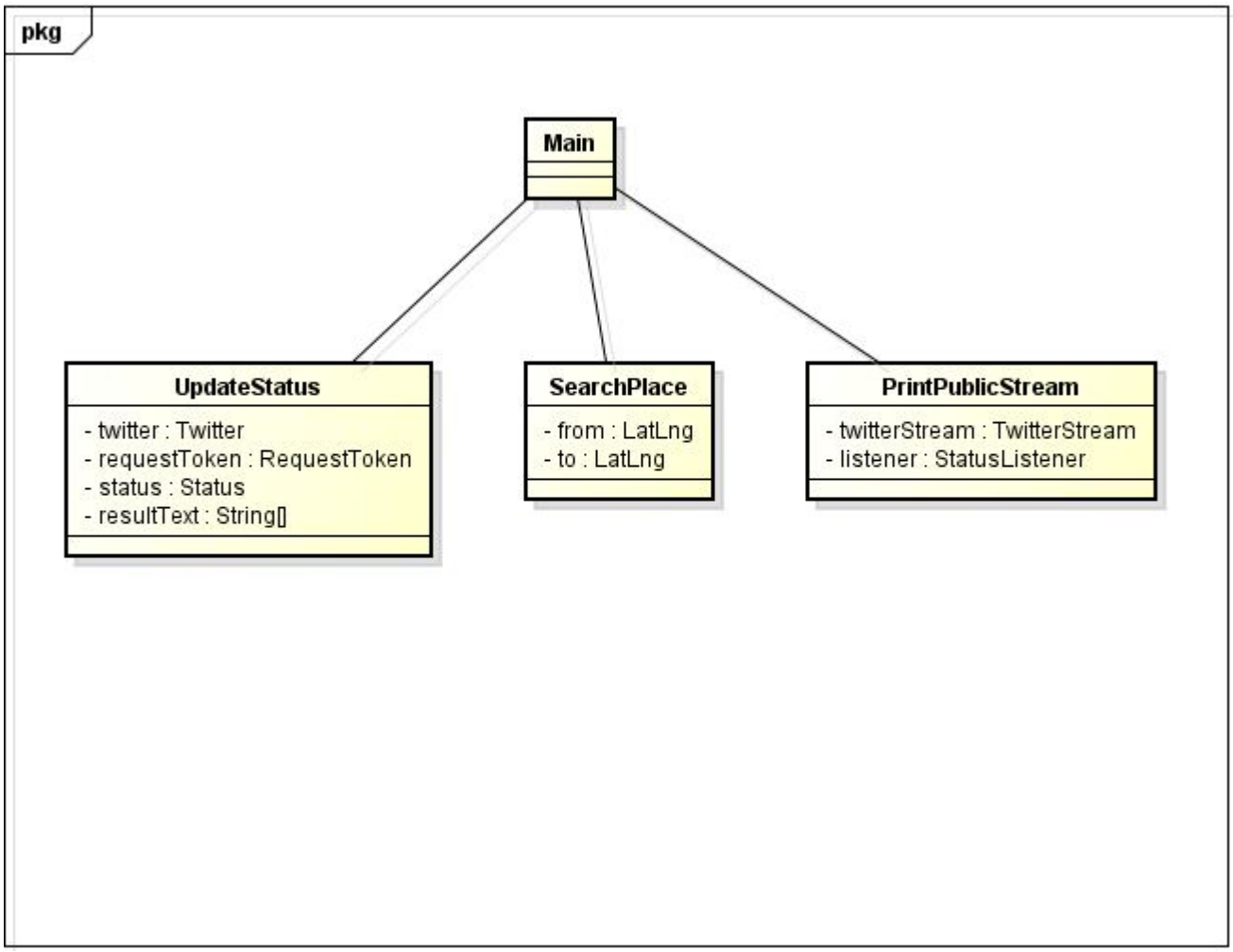
Nama	Bantuan
Aktor	Pengguna
Deskripsi	Melakukan *Tweet* Bantuan
Kondisi Awal	Belum menuliskan *Tweet* pada kolom *update*
Kondisi Akhir	Sudah melakukan *Tweet* bantuan kepada user @kviniink
Skenario Utama	Pengguna melakukan *Tweet* bantuan kepada *user* @kviniink
Eksepsi	Format penulisan salah

3.2.3 *Class Diagram*

Untuk membuat *class diagram Twitter bot* untuk mencari jalur transportasi publik, dibutuhkan kebutuhan kelas dari skenario. Pada tabel skenario 3.1, masukan akan terjadi hal-hal seperti dibawah ini:

1. *Twitter bot* akan berjalan terus hingga *Twitter bot* di non-aktifkan.
2. Pengguna melakukan *tweet* mencari informasi transportasi dengan cara melakukan *mention* kepada akun *Twitter bot* @kviniink dengan format yang sesuai dengan ketentuan.
3. *Twitter bot* menerima *mention* dari pengguna.
4. *Twitter bot* mencari jalur transportasi publik.
5. *Twitter bot* melalukan *reply* kepada pengguna yang berisi jalur transportasi publik yang harus ditempuh.

Gambar 3.2 memiliki kelas *main* yang berfungsi untuk membuat koneksi dengan Twitter ketika perangkat lunak dijalankan. Kelas *PrintPublicStream* merupakan kelas yang berfungsi untuk menangkap *tweet* yang akan di *mention* kepada akun *Twitter bot* @kviniink. Kelas *SearchPlace* merupakan kelas yang berhubungan dengan KIRI API, semua aktifitas yang berhubungan dengan KIRI API akan dilakukan pada kelas ini. Lalu kelas *Update Status* merupakan kelas yang berfungsi agar *Twitter bot* dapat melakukan *tweet* balasan kepada pengguna.



Gambar 3.2: *Class Diagram Twitter Bot*

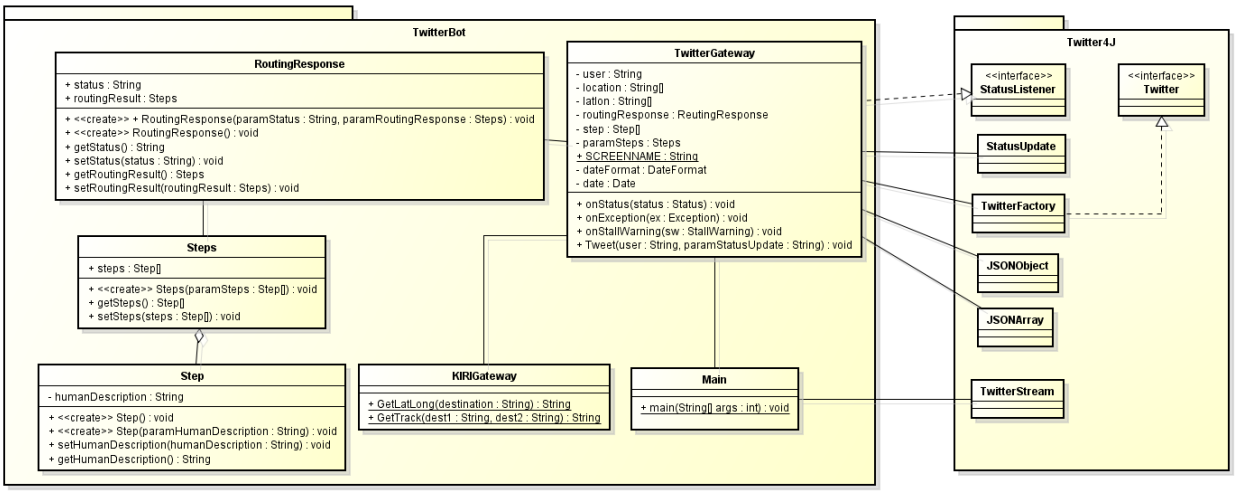
BAB 4

PERANCANGAN PERANGKAT LUNAK

Pada bab ini akan dibahas mengenai perancangan aplikasi untuk membuat *Twitter bot* untuk mencari jalur transportasi publik sesuai analisa yang sudah dibahas pada bab 3.

4.1 Perancangan Kelas

Sub bab ini akan membahas tentang rancangan kelas dan *method* yang akan dibuat pada perangkat lunak *Twitter bot* untuk mencari jalur transportasi publik. Untuk lebih jelas mengenai kelas yang ada pada aplikasi ini, penulis menyajikan gambar kelas diagram yang dapat dilihat pada Gambar 4.1



Gambar 4.1: Diagram Kelas Pembuatan *Twitter bot* untuk Mencari Jalur Transportasi Publik

- Kelas Main merupakan kelas yang berfungsi untuk membuat koneksi dengan Twitter ketika perangkat lunak dijalankan.
  - Method
    - public static void main(String[] args), merupakan method main untuk menjalankan program.
- Kelas Twitter Gateway, merupakan kelas untuk menangkap dan membalas *tweet*. Kelas Twitter Gateway ini mengimplementasikan *StatusListener*.

- Atribut
  - String user, digunakan untuk menampung nama akun pengguna *Twitter bot*.
  - String location[], berupa *array* yang digunakan untuk menampung lokasi awal dan lokasi tujuan.
  - String latlon[], berupa *array* yang digunakan untuk menampung koordinat lokasi awal dan koordinat lokasi tujuan.
  - RoutingResponse routingResponse, merupakan atribut yang digunakan untuk menampung hasil yang diberikan oleh KIRI API.
  - Step[] step, berupa *array* yang berguna untuk menampung langkah-langkah informasi perjalanan.
  - Steps steps, merupakan atribut yang berguna untuk menampung semua step.
- Method
  - public void onStatus(Status status), merupakan *method* yang berguna untuk menangkap *tweet* dan memproses *tweet* tersebut. Jika ada *tweet* yang di-*mention* kepada akun *Twitter bot* dan *tweet* yang diterima merupakan *tweet* untuk mencari jalur transportasi publik, maka *tweet* tersebut akan dimasukan ke atribut yang sudah disediakan. Atribut tersebut antara lain adalah *user*, lokasi awal dan lokasi tujuan. Setelah mendapatkan lokasi awal dan lokasi tujuan barulah proses pencarian dimulai dengan menggunakan *GetLatLong method* dan *GetTrack method* yang terdapat di kelas KIRIGateway. Hasil pencarian akan dimasukan ke dalam atribut *routingResponse*, *step*, dan *steps*. Setelah itu akan dilakukan pemanggilan *Tweet method* untuk melakukan proses *reply*.
  - public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice), merupakan *overload method* dari kelas *interface StatusListener*.
  - public void onTrackLimitationNotice(int numberOfLimitedStatuses), merupakan *overload method* dari kelas *interface StatusListener*.
  - public void onScrubGeo(long userId, long upToStatusId), merupakan *overload method* dari kelas *interface StatusListener*.
  - public void onException(Exception ex), merupakan *method* yang berguna untuk menangkap *exception*.
  - public void onStallWarning(StallWarning sw), merupakan *overload method* dari kelas *interface StatusListener*.
  - public void Tweet(String user, String paramStatusUpdate), merupakan *method* untuk melakukan *reply* yang ditujukan kepada akun pengguna *Twitter bot*. Twitter hanya dapat melakukan *tweet* dengan batas 140 karakter, oleh karena itu *method* ini akan mengatasi keterbatasan *tweet* tersebut dengan melakukan pembagian *tweet*. Method ini akan memberi tambahan waktu yang sesuai dengan *server* di setiap akhir *tweet*, hal ini bertujuan untuk menghindari adanya *duplicate tweet*.

- Kelas KIRIGateway merupakan kelas untuk memanggil KIRI API. Pemanggilan KIRI API ini digunakan untuk mendapatkan koordinat suatu lokasi dan mencari jalur transportasi publik.

  - Method
    * public static String GetLatLong(String destination), merupakan *method* yang digunakan untuk mencari koordinat dari suatu lokasi. Hasil kembalian dari *method* ini berupa *latitude* and *longitude* yang diberikan oleh KIRI API lalu diubah ke dalam bentuk *String*.
    * public static String GetTrack(String dest1, String dest2), merupakan *method* yang digunakan untuk mencari jalur transportasi publik dari lokasi awal ke lokasi tujuan. Hasil kembalian dari *method* ini adalah langkah-langkah perjalanan dari lokasi awal ke lokasi tujuan dengan menggunakan transportasi publik.

- Kelas RoutingResult merupakan kelas untuk menampung hasil kembalian dari KIRI API
  - Atribut
    * status, merupakan atribut yang digunakan untuk menyimpan status dari hasil pencarian.
    * routingResult, merupakan atribut yang digunakan untuk menyimpan langkah-langkah perjalanan.
  - Method
    * public RoutingResponse(String paramStatus, Steps paramRoutingResult), merupakan *constructor* dari kelas RoutingResult.
    * public RoutingResponse(), merupakan *constructor* dari kelas RoutingResult.
    * public String getStatus(), merupakan *getter* dari atribut status.
    * public void setStatus(String status), merupakan *setter* dari atribut status.
    * public Steps getRoutingResult(), merupakan *getter* dari atribut routingResult.
    * public void setRoutingResult(Steps routingResult), merupakan *setter* dari atribut routingResult.

- Kelas Step merupakan kelas untuk menampung jalur perjalanan dari lokasi awal ke lokasi tujuan dengan menggunakan transportasi publik yang diberikan oleh KIRI API.
  - Atribut
    * String humanDescription, merupakan atribut untuk menjelaskan cara perjalanan yang bahasanya dimengerti oleh pengguna.
  - Method
    * public Step(), merupakan *constructor* dari kelas Step.
    * public Step(String paramHumanDescription), merupakan *constructor* dari kelas Step.
    * public String getHumanDescription(), merupakan *getter* dari atribut humanDescription.
    * public void setHumanDescription(String humanDescription), merupakan *setter* dari atribut humanDescription.

- Kelas Steps merupakan kelas untuk menampung kumpulan step.
  - Atribut
    - Step[] steps, merupakan atribut yang berisi *array* step
  - Method
    - public Steps(Step[] paramSteps), merupakan konstruktor dari kelas Steps.
    - public Step[] getSteps(), merupakan *getter* dari atribut steps.
    - public void setSteps(Step[] steps), merupakan *setter* dari atribut steps.

4.2 Sequence Diagram

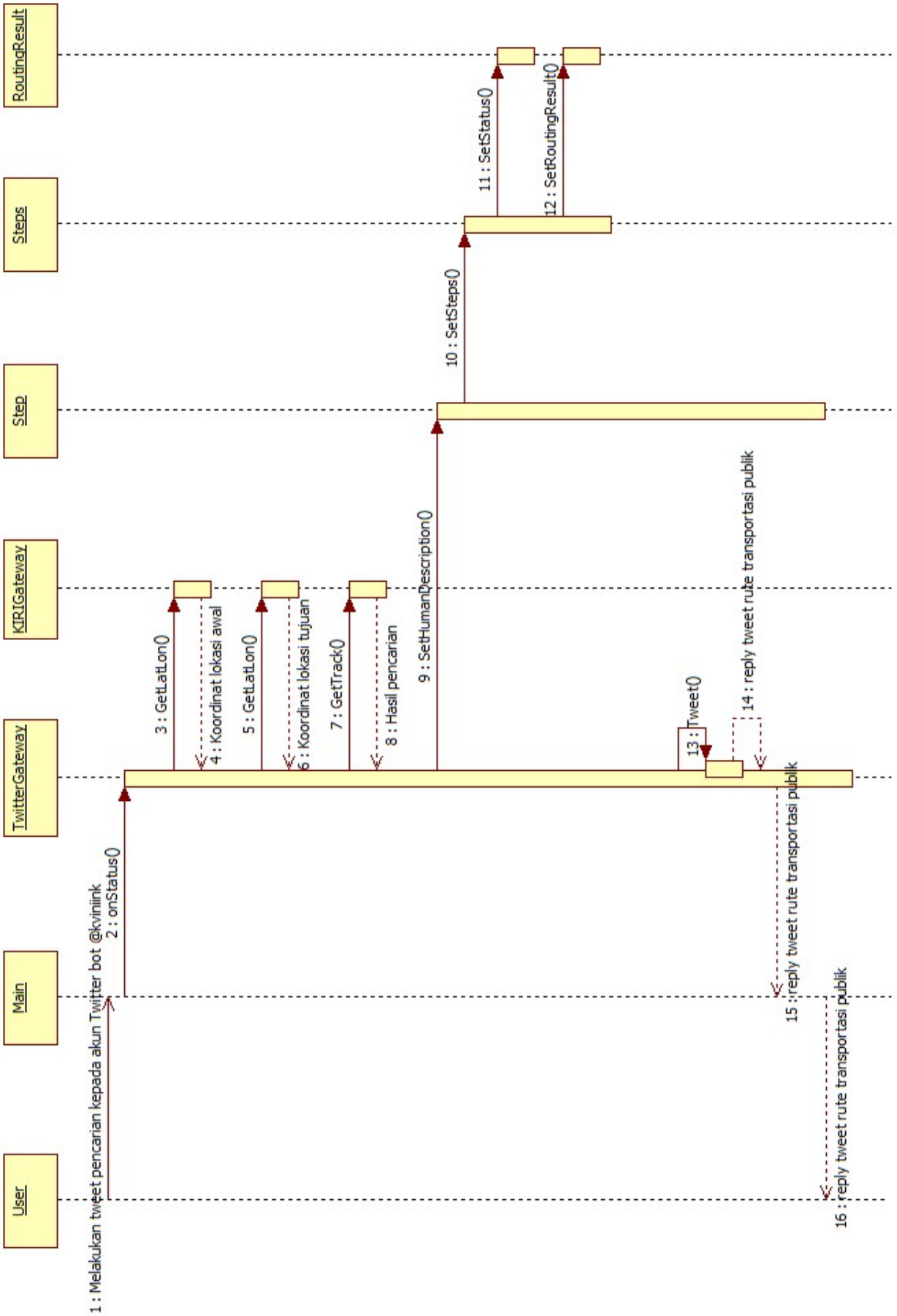
Pada sub bab ini, akan dijelaskan alur program dengan menggunakan *sequence diagram* pada Gambar 4.2

Pertama, perangkat lunak akan melakukan *streaming* pada saat kelas *main* dijalankan. Kelas *main* akan membuka gerbang untuk mengakses *Twitter API*, dengan menggunakan *Streaming* API perangkat lunak akan menangkap semua *tweet* yang melakukan *mention* kepada akun *Twitter bot* @kviniink. Perangkat lunak akan terus melakukan *streaming tweet* hingga perangkat lunak dinon-aktifkan.

Kelas TwitterGateway akan memproses *tweet* yang di-*mention* kepada akun *Twitter bot* @kviniink. *onStatus Method* akan melakukan pengecekan apakah *tweet* tersebut merupakan *tweet* untuk mencari jalur transportasi publik atau bukan. Jika benar, maka nama akun pengirim, lokasi awal, dan lokasi tujuan akan disimpan di atribut yang sudah disediakan. Setelah itu akan dicari koordinat dari masing-masing lokasi menggunakan KIRI API. Proses pencariian koordinat dilakukan oleh kelas KIRIGateway.

Kelas KIRIGateway akan memanggil *GetLatLon method* untuk mencari koordinat suatu lokasi. Setelah didapatkan koordinat lokasi awal dan lokasi tujuan, kelas TwitterGateway akan mengubah hasil dari *GetLatLon method* yang berupa *JSON* menjadi format *String*. Setelah didapatkan koordinat lokasi awal dan koordinat lokasi tujuan maka hasil dari masing-masing koordinat dikembalikan kepada kelas KIRIGateway untuk dicari jalur transportasi publik dari lokasi awal menuju lokasi tujuan menggunakan *GetTrack method*. Hasil dari *GetTrack method* akan disimpan pada atribut *step, steps*, dan *routingResult*. *Method setHumanDescription* berguna untuk menyimpan hasil kembalian KIRI pada atribut *step*. *Method setSteps* merupakan *method* untuk menyimpan *step* kedalam atribut *steps*. *Method setStatus* merupakan *method* untuk menyimpan *status* hasil kembalian KIRI kedalam atribut *status*. *Method setRoutingResult* merupakan *method* untuk menyimpan *steps* kedalam atribut *routingResult*.

Setelah selesai, langkah-langkah jalur transportasi publik siap untuk di *reply* kepada pengguna. Proses *reply* dilakukan oleh *tweet method* yang terdapat pada kelas TwitterGateway. *Tweet* tersebut berisi tentang jalur transportasi publik dari lokasi awal menuju lokasi tujuan.*Tweet* akan di-*reply* satu per satu sesuai dengan banyaknya *step* yang ada. Perangkat lunak akan terus melakukan proses tersebut hingga perangkat lunak dinon-aktifkan.



Gambar 4.2: Sequence Diagram *Twitter bot* untuk Mencari Jalur Transportasi Publik

4.3 Perancangan Antarmuka

Perangkat lunak yang dibangun memiliki antarmuka berbasis teks. Untuk menjalankan perangkat lunak *Twitter bot* cukup dengan menjalankan aplikasi. *Twitter bot* akan terus berjalan hingga perangkat lunak di *non*-aktifkan. Antarmuka yang akan digunakan oleh pengguna menggunakan antar muka yang disediakan oleh Twitter. Antarmuka yang disediakan oleh Twitter adalah antarmuka yang terdapat pada:

1. *website* Twitter,
2. Android,
3. iOS,
4. Windows Phone.

BAB 5

IMPLEMENTASI DAN PENGUJIAN APLIKASI

Pada bab 5 akan dibahas implementasi dan pengujian aplikasi pembuatan *Twitter bot* untuk mencari jalur transportasi publik.

5.1 Lingkungan Pembangunan

Lingkungan perangkat lunak dan perangkat keras yang digunakan untuk membangun dan menguji aplikasi pembuatan *Twitter bot* untuk mencari jalur transportasi publik ini adalah:

- Komputer
  - Processor: Intel Core i7-2630QM CPU 2.00 GHz
  - RAM: 4096MB
  - Hardisk: 211GB
  - VGA : NVDIA GeForce GT 540M
- Sistem operasi: Windows 7 Professional
- Platform: NetBeans: IDE 8.0.2
- Akun *Twitter bot*
  - Nama akun: kviniink
  - ConsumerKey : 3iT8duMItTTrdaU1qTHxwDIUl
  - ConsumerSecret : YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7t313
  - AccessToken : 313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
  - AccessTokenSecret : LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptcuR4i
- Akun Twitter penguji : kviniinktest123

5.2 Hasil Penggunaan Antarmuka

Pembuatan perangkat lunak *Twitter bot* untuk mencari jalur transportasi publik ini memiliki tampilan antarmuka berbasis teks yang berguna untuk melihat hasil *tweet* yang diterima *Twitter bot*, dan hasil *tweet* yang di-*reply* *Twitter bot* kepada pengguna. Gambar 5.1 adalah tampilan antarmuka *Twitter bot* untuk mencari jalur transportasi publik.

```
run:
[Tue May 19 19:01:11 ICT 2015]Establishing connection.
[Tue May 19 19:01:30 ICT 2015]Connection established.
[Tue May 19 19:01:30 ICT 2015]Receiving status stream.
@kviniinktest123 - @kviniink bip to ip
Lokasi 1 : bip
Lokasi 2 : ip
@kviniinktest123 Jalan dari lokasi mulai Anda ke Jalan Merdeka sejauh kurang lebih 58 meter.
@kviniinktest123 Naik angkot Dago - St. Hall di Jalan Merdeka, dan turun di Jalan Riau kurang lebih setelah 3,9 kilometer.
@kviniinktest123 Jalan dari Jalan Riau ke tujuan akhir Anda sejauh kurang lebih 92 meter.
@kviniinktest123 Untuk lebih lengkap silahkan lihat di http://kiri.travel?start=%20bip&finish=ip&region=bdo
@kviniinktest123 - @kviniink badung to unpar
Lokasi 1 : badung
Lokasi 2 : unpar
@kviniinktest123  badung tidak ditemukan
```

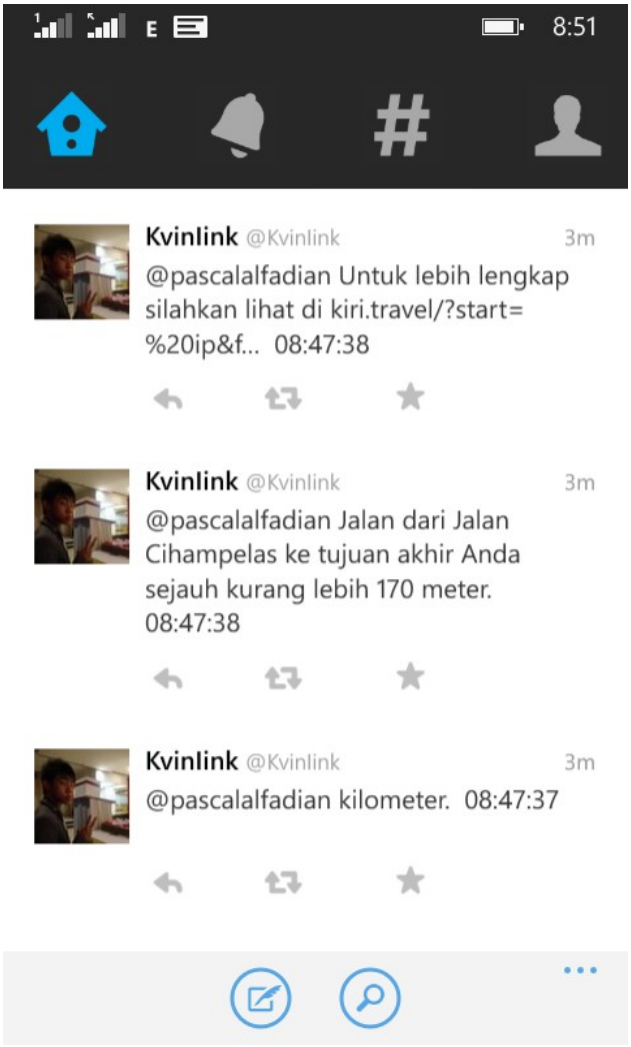
Gambar 5.1: Antarmuka Perangkat Lunak Twitter Bot Untuk Mencari Jalur Transportasi Publik

Antarmuka Twitter Pengguna dapat mencoba perangkat lunak *Twitter bot* menggunakan Twitter, baik menggunakan *website* Twitter ataupun aplikasi Twitter. Oleh karena itu, tampilan antarmuka setiap pengguna akan berbeda-beda sesuai dengan *device* yang digunakan pengguna. Berikut adalah contoh beberapa tampilan antarmuka yang diberikan oleh Twitter:

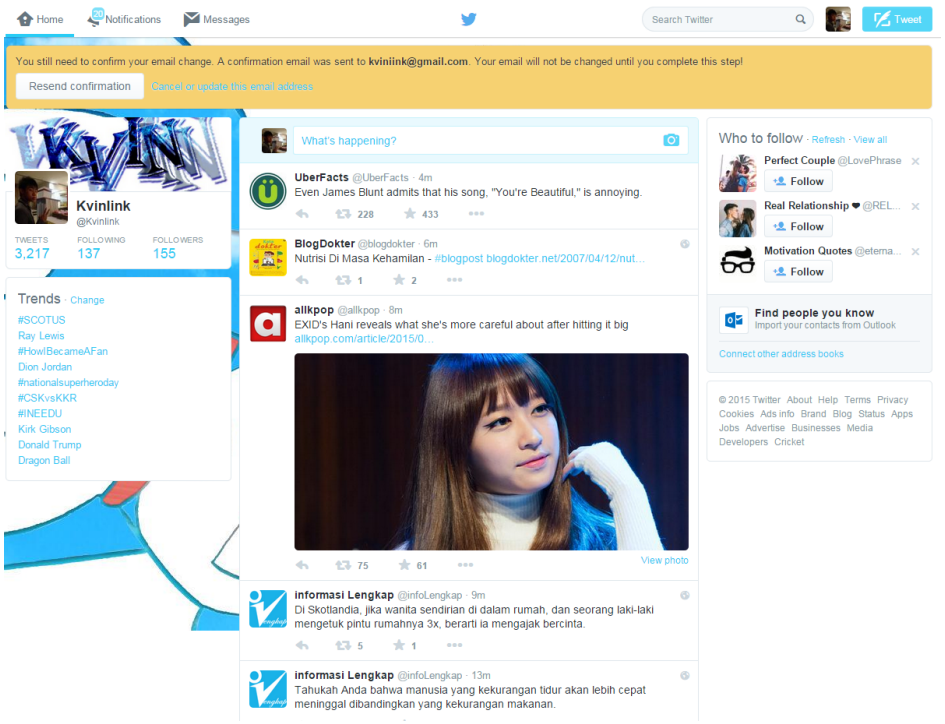
- Antarmuka Twitter yang diakses melalui aplikasi Twitter di Android dapat dilihat pada Gambar 5.2
- Antarmuka Twitter yang diakses melalui aplikasi Twitter di iOS dapat dilihat pada Gambar 5.3
- Antarmuka Twitter yang diakses melalui aplikasi Twitter di Windows Phone dapat dilihat pada Gambar 5.4
- Antarmuka Twitter yang diakses melalui aplikasi Twitter di Website Twitter dapat dilihat pada Gambar 5.5

Gambar 5.2: Antarmuka Twitter yang diakses melalui aplikasi Twitter di Android

Gambar 5.3: Antarmuka Twitter yang diakses melalui aplikasi Twitter di iOS



Gambar 5.4: Antarmuka Twitter yang diakses melalui aplikasi Twitter di Windows Phone



Gambar 5.5: Antarmuka Twitter yang diakses melalui aplikasi Twitter di Website Twitter

5.3 Pengujian

Pada sub-bab ini akan dibahas mengenai hasil pengujian yang telah dilakukan terhadap perangkat lunak yang dibangun oleh penulis. Pengujian terdiri dari dua bagian, yaitu pengujian fungsional dan pengujian eksperimental. Pengujian fungsional bertujuan untuk memastikan semua fungsi aplikasi berjalan sesuai harapan. Sementara pengujian eksperimental bertujuan untuk mengetahui keberhasilan proses kerja dari aplikasi yang dibangun.

5.3.1 Pengujian Fungsional

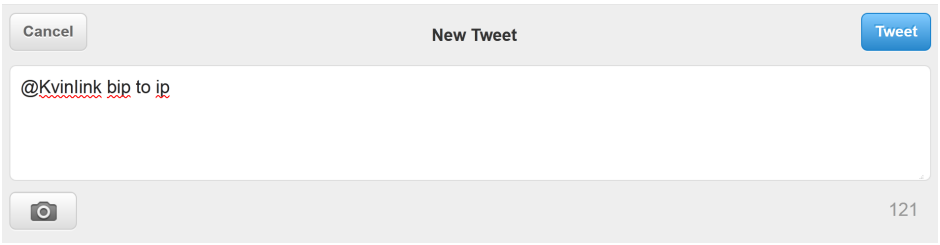
Pengujian fungsional dilakukan pada fungsionalitas yang tersedia pada aplikasi yang dibangun. Pengujian ini dilakukan untuk mengetahui kesesuaian reaksi nyata dengan reaksi yang diharapkan dari aplikasi yang dibangun. Hasil pengujian ditunjukan pada tabel 5.1.

5.3.2 Pengujian Eksperimental

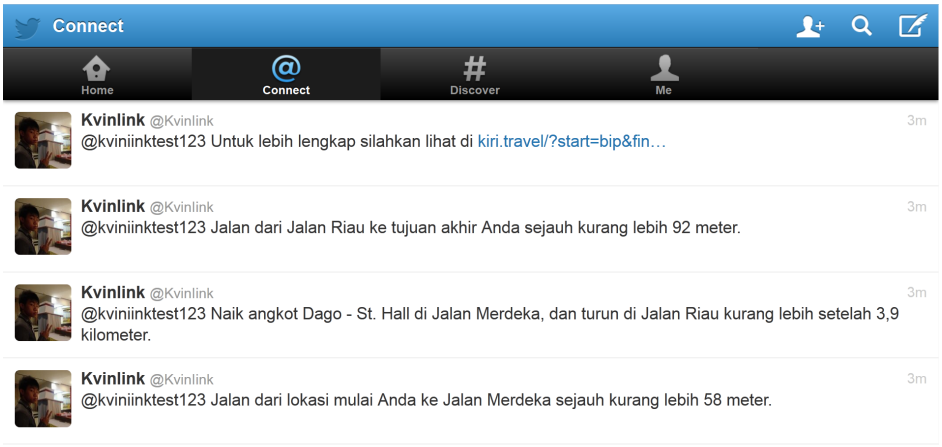
Pada sub bab ini akan dilakukan pengujian terhadap *Twitter bot* untuk mencari jalur transportasi publik. Peneliti meminta kepada beberapa orang untuk melakukan pencarian jalur transportasi publik kepada *Twitter bot* untuk mencari jalur transportasi publik. Selain itu juga peneliti mencoba melakukan *tweet* pencarian melalui akun @kviniinktest123.

1. Pengujian 1

   Pada pengujian satu, peneliti mencoba untuk mencari jalur transportasi publik untuk lokasi yang umum dikunjungi yaitu *mall*. Pencarian dilakukan dengan lokasi awal yaitu BIP



Gambar 5.6: Tweet dari BIP menuju IP



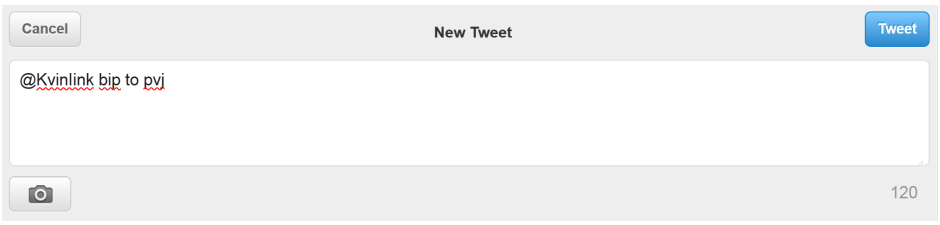
Gambar 5.7: Hasil Pencarian Rute Transportasi Publik dari BIP menuju IP

(Bandung Indah Plaza) menuju lokasi tujuan yaitu IP (Istana Plaza). Akun penguji @kviniinktest123 melakukan *mention* kepada akun *Twitter bot* @kviniink123 yang dapat dilihat pada Gambar 5.6.

Setelah proses *tweet* dilakukan, *Twitter bot* akan menangkap *tweet* tersebut dan memprosesnya. Setelah proses pencarian selesai dilakukan, akun *Twitter bot* @kviniink melakukan *reply* kepada akun @kviniinktest123 yang dapat dilihat pada Gambar 5.7.

Pencarian kedua dilakukan dengan lokasi awal yaitu BIP (Bandung Indah Plaza) dan lokasi tujuan yaitu PVJ (Paris van Java). Dapat dilihat pada Gambar 5.8, akun @kviniinktest123 melakukan *tweet* pencarian jalur transportasi publik yang di-*mention* kepada akun *Twitter bot* @kviniink dengan lokasi awal yaitu BIP dan lokasi tujuan yaitu PVJ.

Setelah itu *tweet* tersebut diproses oleh aplikasi untuk dicari jalur transportasi publiknya, lalu akun *Twitter bot* @kviniink melakukan *reply* kepada akun @kviniinktest123. *Reply tweet* tersebut merupakan jalur transportasi publik yang harus ditempuh, *reply tweet* tersebut dapat



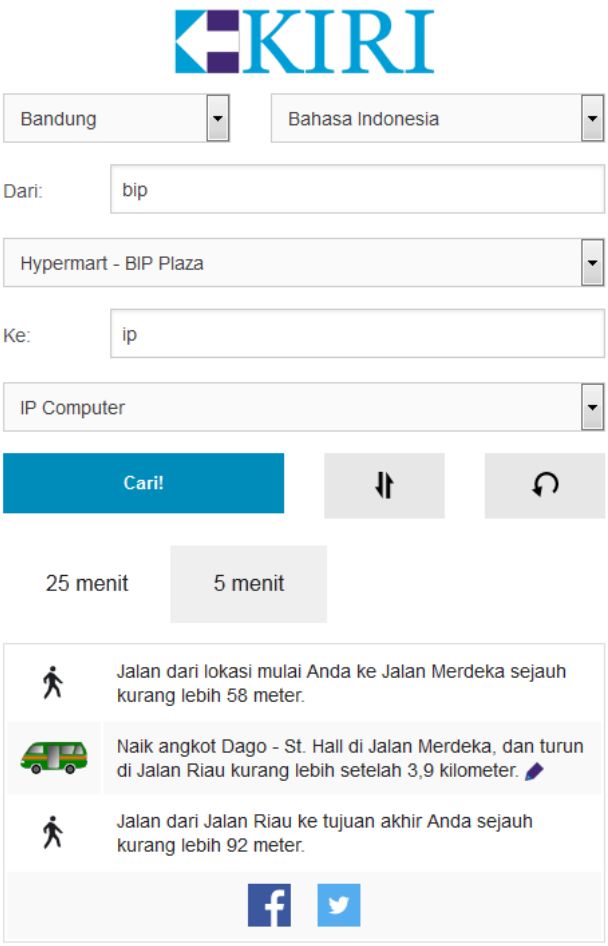
Gambar 5.8: *Tweet* dari BIP menuju PVJ

Gambar 5.9: Hasil Pencarian Rute Transportasi Publik dari BIP menuju PVJ

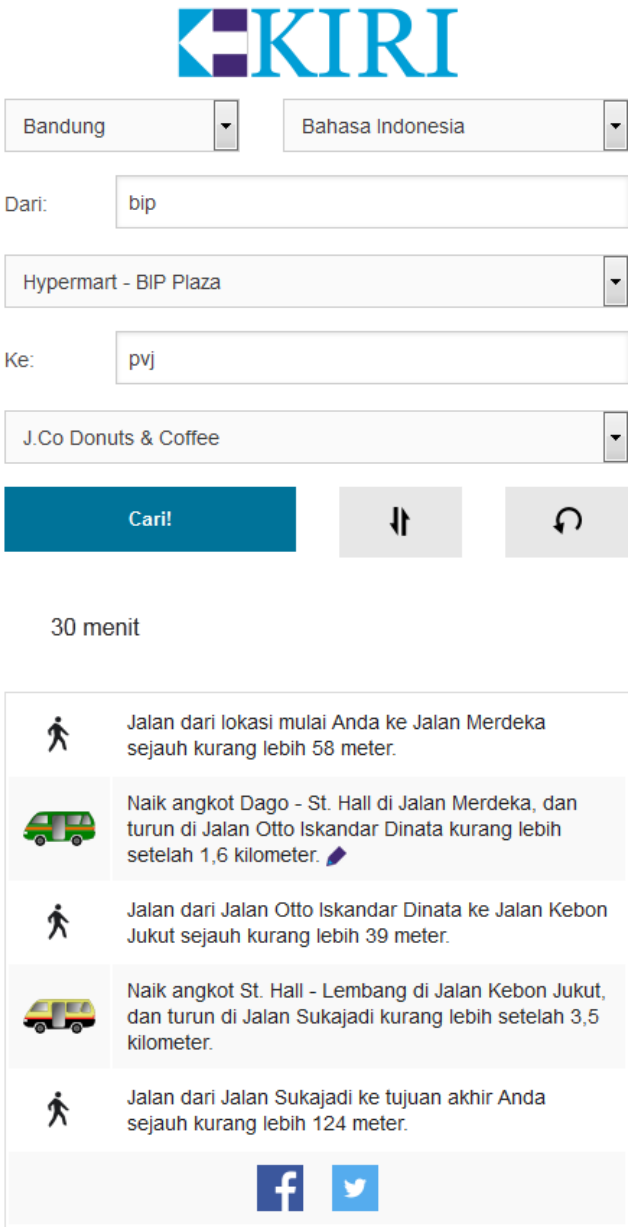
dilihat pada Gambar 5.9.

Pada pencarian kedua dapat dilihat pada *tweet* pertama terjadi ketidak sesuaian hasil dari KIRI API dengan hasil *tweet*. Peneliti lalu melakukan pencarian melalui *website* KIRI yaitu `http://kiri.travel`. Pencarian pertama pada *website* KIRI dilakukan dengan lokasi awal yaitu BIP dan lokasi tujuan yaitu IP. Hasil pencarian pada *website* KIRI dapat dilihat pada Gambar 5.16.

Lalu pencarian kedua pada *website* KIRI dilakukan dengan lokasi awal yaitu BIP dan lokasi tujuan yaitu PVJ. Hasil pencarian KIRI dari BIP menuju PVJ dapat dilihat pada Gambar 5.17. Setelah dilihat dari hasil keduanya, *Twitter bot* melakukan *duplicate tweet* pada *tweet* pertama dalam pencarian ke dua yang dilakukan oleh akun @kviniink123. *Duplicate tweet* adalah *tweet* yang dinyatakan dinyatakan identik oleh Twitter dalam jangka waktu tertentu. *Duplicate tweet* tidak diperbolehkan oleh Twitter. Oleh karena itu, untuk menghindari adanya *duplicate tweet*, penulis menambahkan waktu untuk jam, menit, dan detik di setiap *tweet* yang dilakukan oleh *Twitter bot* agar membuat setiap *tweet* tersebut bersifat unik.



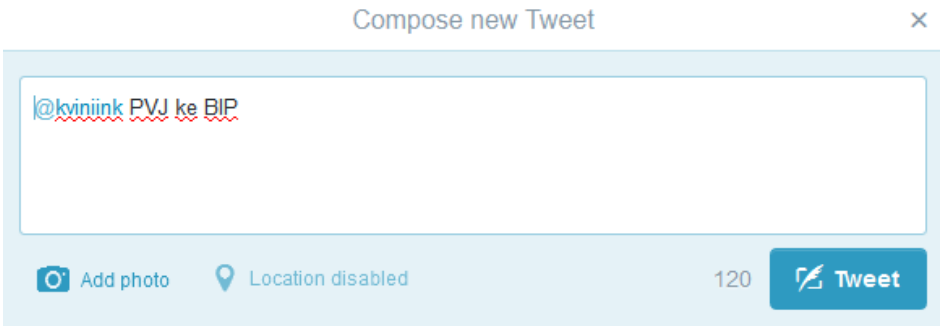
Gambar 5.10: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju IP Melalui Website KIRI



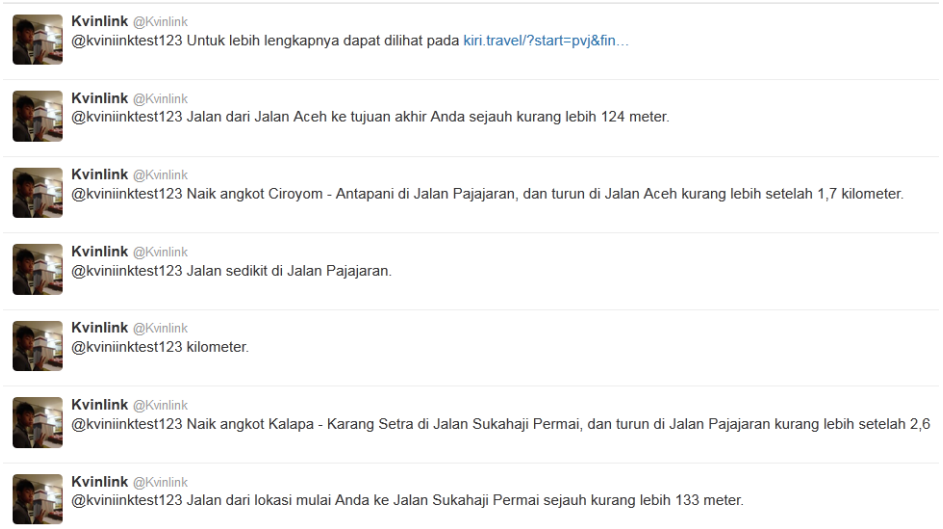
Gambar 5.11: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI

Tabel 5.1: Tabel Hasil pengujian fungsionalitas pada Aplikasi *Twitter bot* untuk mencari jalur transportasi publik

No	Pengujian	Reaksi yang Diharapkan	Reaksi Aplikasi
1	Melakukan otentikasi terhadap akun *Twitter Bot*	Otentikasi berhasil dilakukan antara Twitter dengan akun *Twitter Bot*. Otentikasi dilakukan dengan melakukan pemeriksaan terhadap *ConsumerKey*, *CustomerSecret*, *AccessToken*, dan *AccessTokenSecret*	*ConsumerKey*, *CustomerSecret*, *AccessToken*, dan *AccessTokenSecret* yang diberikan Twitter berhasil diotentikasi oleh aplikasi
2	Melakukan *streaming tweet*	Menangkap semua *tweet* yang di*mention* kepada akun @kviniink	Setiap *tweet* yang di*mention* kepada akun *Twitter bot* @kviniink dapat diterima secara *realtime*
3	Membaca *tweet* yang ditangkap	Melakukan pemeriksaan terhadap *tweet* yang ditangkap, apakah *tweet* tersebut merupakan *tweet* untuk mencari transportasi publik atau bukan	Perangkat lunak dapat membedakan *tweet* untuk mencari jalur transportasi publik dengan *tweet* yang bukan bertujuan untuk mencari jalur transportasi publik. Selain itu Perangkat lunak dapat menangkap *tweet* untuk bantuan pancarian.
4	Melakukan pencarian koordinat suatu lokasi menggunakan KIRI API	Mendapatkan hasil koordinat *latitude* dan *longitude* dari lokasi yang dicari	Perangkat lunak mendapatkan koordinat *latitude* dan *longitude* dari lokasi yang dicari
5	Melakukan pencarian jalur transportasi publik menggunakan KIRI API	Mendapatkan jalur-jalur transportasi publik yang harus ditempuh dari lokasi awal menuju lokasi tujuan	Perangkat lunak mendapatkan jalur-jalur transportasi publik yang harus ditempuh dari lokasi awal menuju lokasi tujuan
6	Melakukan *tweet* balasan	Membalas *tweet* dengan memberikan hasil pencarian jalur transportasi publik dengan format yang sudah ditentukan	Akun *Twitter bot* @kviniink melakukan *reply* kepada akun penguji @kviniinktest123, *reply* tersebut berisikan jalur transportasi publik yang harus ditempuh dari lokasi awal menuju lokasi tujuan. *Reply* sudah dapat dipecah-pecah jika *tweet* melebihi 140 karakter.



Gambar 5.12: Tweet dari PVJ menuju BIP



Gambar 5.13: Hasil Pencarian Rute Transportasi Publik dari BIP menuju IP

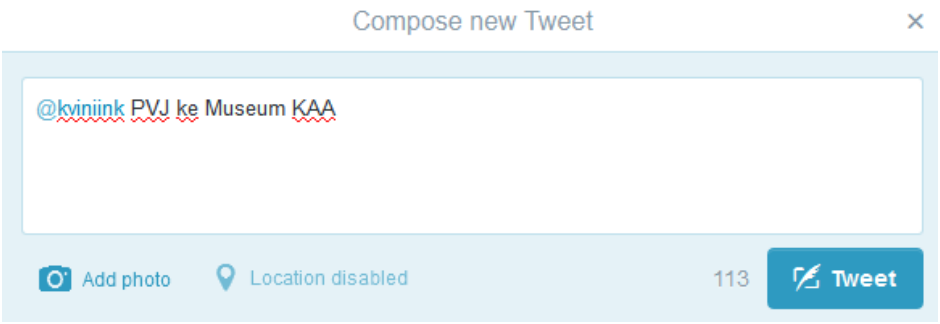
2. Pengujian 1.1

   Pada pengujian satu, peneliti mencoba untuk mencari jalur transportasi publik untuk lokasi yang umum dikunjungi yaitu *mall*. Pencarian dilakukan dengan lokasi awal yaitu PVJ (Paris Van Java) menuju lokasi tujuan yaitu BIP (Bandung Indah Plaza). Akun penguji @kviniinktest123 melakukan *mention* kepada akun *Twitter bot* @kviniink123 yang dapat dilihat pada Gambar 5.12.

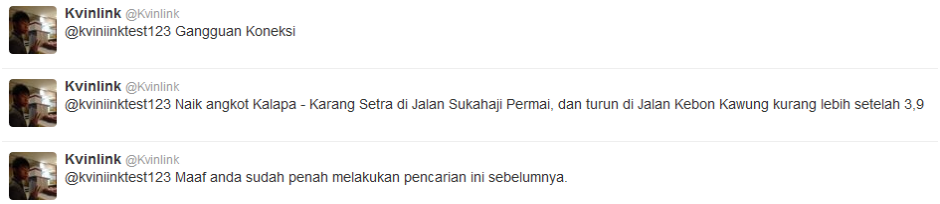
   Setelah proses *tweet* dilakukan, *Twitter bot* akan menangkap *tweet* tersebut dan memprosesnya. Setelah proses pencarian selesai dilakukan, akun *Twitter bot* @kviniink melakukan *reply* kepada akun @kviniinktest123 yang dapat dilihat pada Gambar 5.13.

   Pencarian kedua dilakukan dengan lokasi awal yaitu PVJ(Paris Van Java) dan lokasi tujuan yaitu Museum KAA(Konferensi Asia Afrika). Dapat dilihat pada Gambar 5.14, akun @kviniinktest123 melakukan *tweet* pencarian jalur transportasi publik yang di-*mention* kepada akun *Twitter bot* @kviniink dengan lokasi awal yaitu PVJ dan lokasi tujuan yaitu Museum KAA.

   Setelah itu *tweet* tersebut diproses oleh aplikasi untuk dicari jalur transportasi publiknya, lalu akun *Twitter bot* @kviniink melakukan *reply* kepada akun @kviniinktest123. *Reply tweet* tersebut merupakan jalur transportasi publik yang harus ditempuh, *reply tweet* tersebut dapat dilihat pada Gambar 5.15.



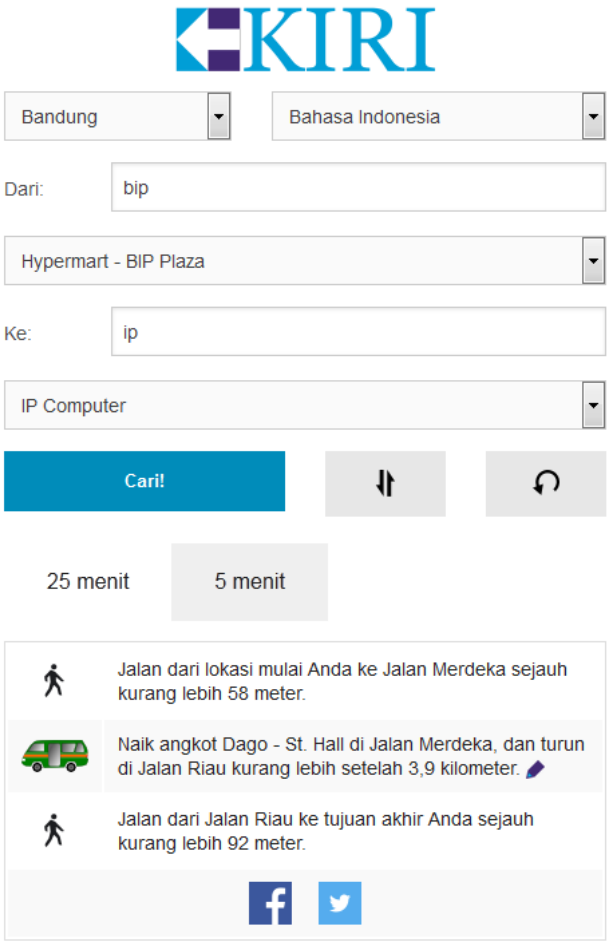
Gambar 5.14: *Tweet* dari PVJ menuju Museum KAA



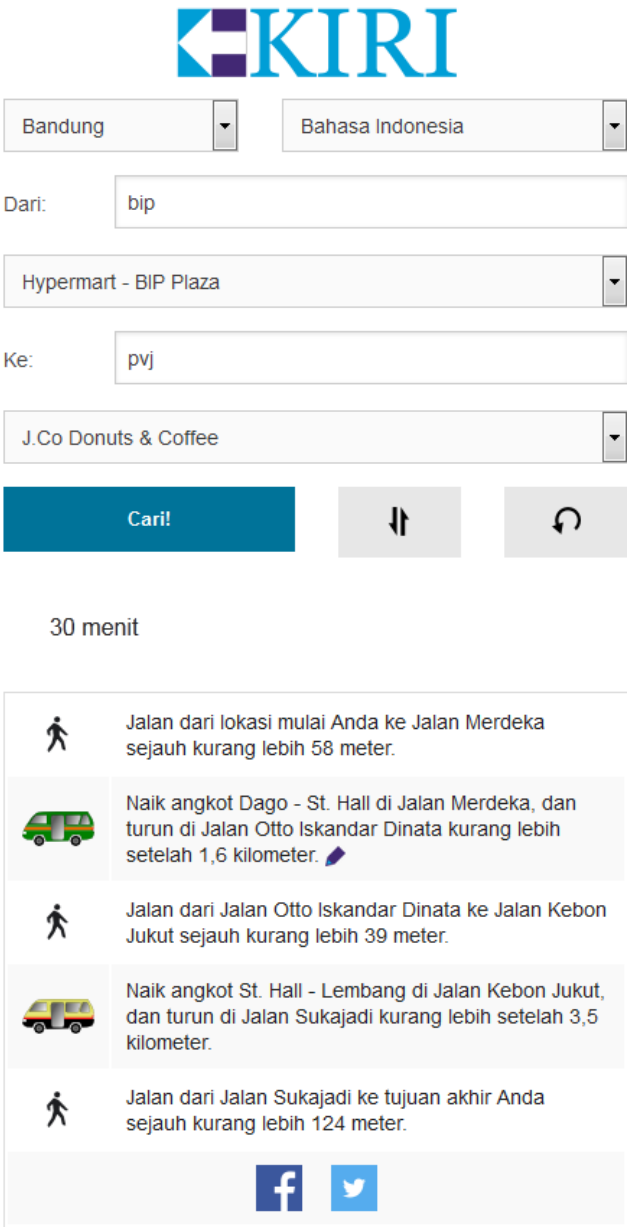
Gambar 5.15: Hasil Pencarian Rute Transportasi Publik dari PVJ menuju Museum KAA

Pada pencarian kedua dapat dilihat pada *tweet* pertama terjadi ketidak sesuaian hasil dari KIRI API dengan hasil *tweet*. Peneliti lalu melakukan pencarian melalui *website* KIRI yaitu `http://kiri.travel`. Pencarian pertama pada *website* KIRI dilakukan dengan lokasi awal yaitu PVJ dan lokasi tujuan yaitu BIP. Hasil pencarian pada *website* KIRI dapat dilihat pada Gambar 5.16.

Lalu pencarian kedua pada *website* KIRI dilakukan dengan lokasi awal yaitu PVJ dan lokasi tujuan yaitu Museum KAA. Hasil pencarian KIRI dari PVJ menuju Museum KAA dapat dilihat pada Gambar 5.17. Setelah dilihat dari hasil keduanya, *Twitter bot* melakukan *duplicate tweet* pada *tweet* pertama dalam pencarian ke dua yang dilakukan oleh akun @kviniink123. *Duplicate tweet* adalah *tweet* yang dinyatakan dinyatakan identik oleh Twitter dalam jangka waktu tertentu. *Duplicate tweet* tidak diperbolehkan oleh Twitter. Oleh karena itu, untuk menghindari adanya *duplicate tweet*, penulis menambahkan waktu untuk jam, menit, dan detik di setiap *tweet* yang dilakukan oleh *Twitter bot* agar membuat setiap *tweet* tersebut bersifat unik.



Gambar 5.16: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju IP Melalui Website KIRI

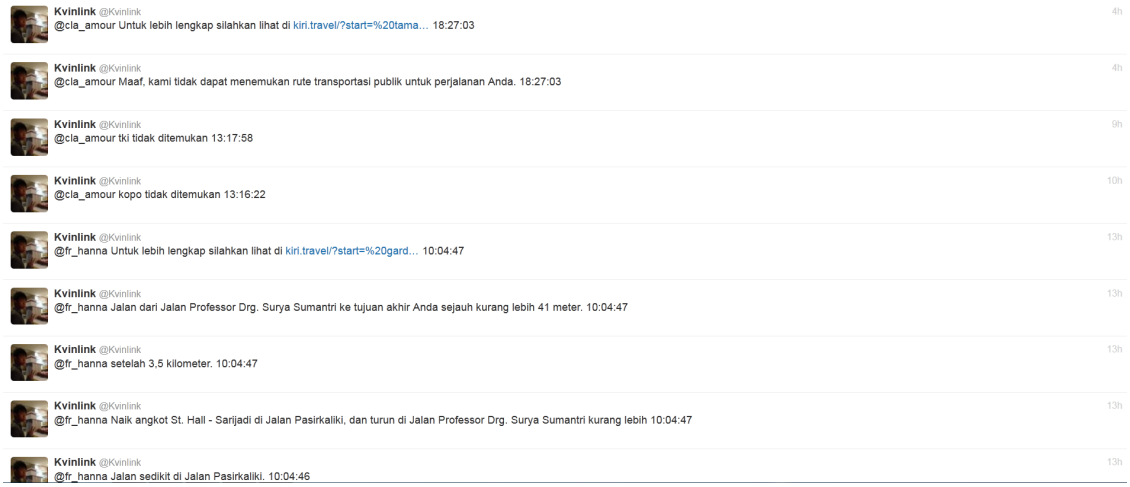


Gambar 5.17: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI

3. Pengujian 2

   Pada pengujian dua, penulis melakukan pengujian dengan cara menjalankan aplikasi selama 24 jam. Selain itu penulis juga melakukan pengujian dengan cara meminta bantuan kepada beberapa responden untuk melakukan *tweet* pencarian jalur transportasi publik. Pengujian ini dilakukan untung mengetahui apakah aplikasi *Twitter bot* berjalan dengan baik atau tidak. Dari hasil yang didapatkan *Twitter bot* dapat memberi pesan bahwa suatu lokasi pencarian tidak ditemukan. Pesan pemberitahuan yang dapat dilihat pada Gambar 5.18. Selain itu juga *Twitter bot* dapat memberi pesan juga lokasi awal dan lokasi tujuan merupakan lokasi yang sama. Pesan pemberitahuan dapat dilihat pada Gambar 5.19. Pada Gambar 5.18 dapat dilihat bahwa akun @cla_amour mencari lokasi tki dan kopo tetapi lokasi pencarian tidak ditemukan. Akun @cla_amour juga mencari jalur transportasi publik yang lokasinya ditemukan tetapi tidak ada rute transportasi publiknya. Hasil *reply Twitter bot* yang dilakukan oleh akun @cla_amour dapat dilihat pada salah satu *reply* yang terdapat pada Gambar 5.18.

   Selain itu *Twitter bot* tidak akan mendapatkan *error* ketika akun *Twitter Bot* @kviniink mendapat banyak *mention* dalam satu *tweet* seperti pada Gambar 5.20, jika format penulisan benar maka *Twitter bot* akan tetap mencari jalur transportasi publiknya yang dapat dilihat pada Gambar 5.21. Gambar 5.18, Gambar 5.22, dan Gambar 5.23 merupakan beberapa hasil *reply* dari *Twitter bot*.

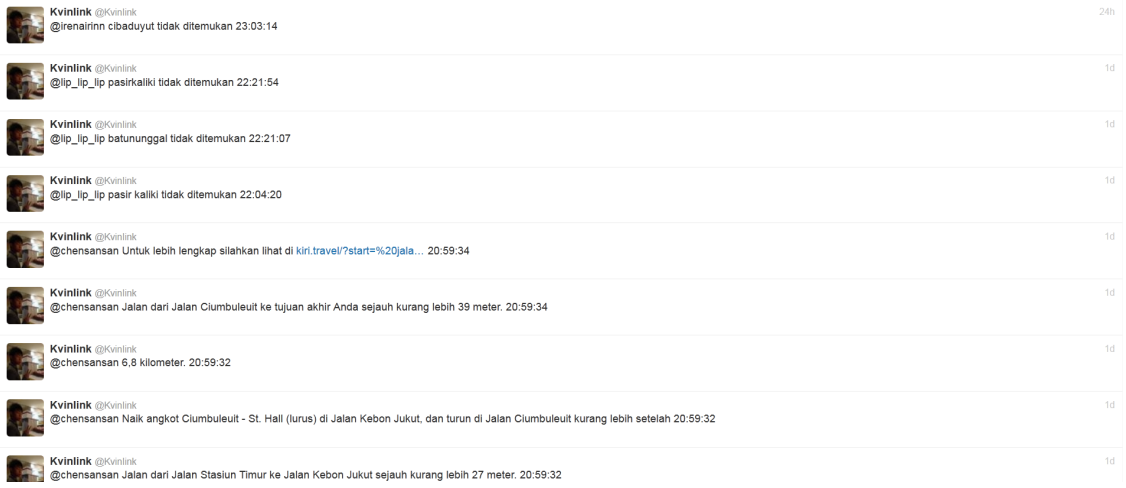


Gambar 5.18: Hasil Reply Twitter Bot

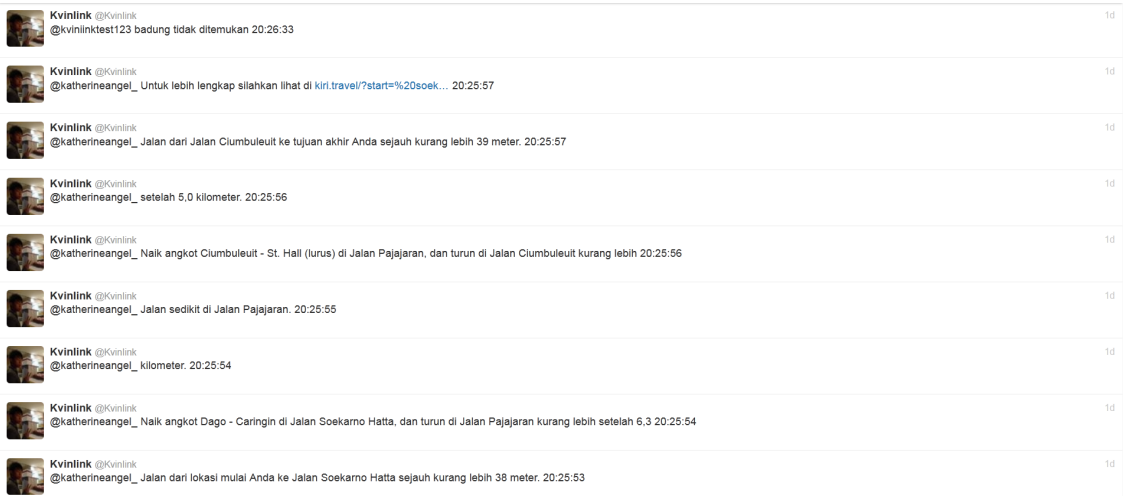
Gambar 5.19: Hasil *Tweet* Jika Lokasi Awal dan Lokasi Tujuan Sama

Gambar 5.20: Akun *Twitter Bot* Mendapat Banyak Mention Dalam Satu *Tweet*

Gambar 5.21: Hasil Reply *Twitter Bot*



Gambar 5.22: Hasil Reply *Twitter Bot*



Gambar 5.23: Hasil Reply *Twitter Bot*

4. Pengujian 3

   Pada pengujian tiga dilakukan pengujian untuk mengetahui apakah hasil *tweet* yang diberikan *Twitter bot* terdapat kesalahan atau tidak jika *Twitter bot* mendapatkan dua *tweet* atau lebih pada waktu yang bersamaan. Pengujian dilakukan dengan cara melakukan dua *tweet* pencarian secara bersamaan. Penulis meminta bantuan kepada beberapa responden untuk melakukan *tweet* pencarian secara bersamaan. Hasil pengujian untuk *tweet* pencarian yang dilakukan secara bersamaan dapat dilihat pada tabel 5.5.

Tabel 5.2: Tabel 1 hasil pengujian *tweet* yang dilakukan secara bersamaan

No	Akun penguji	Waktu *tweet*	*Tweet* yang dikirimkan pengguna	*Tweet* yang diterima pengguna
1	@ClaraKwaria	10:49	@KvinIink UNPAR to PVJ	• @ClaraKwaria Walk about 44 meter from your starting point to Jalan Ciumbuleuit. 22:49:19 • @ClaraKwaria Take angkot Ciumbuleuit - St. Hall (belok) at Jalan Ciumbuleuit, and alight at Jalan Sederhana about 2.4 kilometer 22:49:20 • @ClaraKwaria later. 22:49:20 • @ClaraKwaria Walk about 460 meter from Jalan Sederhana to your destination. 22:49:20 • @ClaraKwaria For futher information you can visit http:kiri.travel?start=unpar &finish=pvj&region=bdo 22:49:20
2	@clara00010111	10:49	@KvinIink UNPAR to PVJ	• @clara00010111 Walk about 44 meter from your starting point to Jalan Ciumbuleuit. 22:49:23 • @clara00010111 Take angkot Ciumbuleuit - St. Hall (belok) at Jalan Ciumbuleuit, and alight at Jalan Sederhana about 2.4 kilometer 22:49:23 • @clara00010111 later. 22:49:23 • @clara00010111 Walk about 460 meter from Jalan Sederhana to your destination. 22:49:24 • @clara00010111 For futher information you can visit http:kiri.travel?start=unpar &finish=pvj&region=bdo 22:49:24

Tabel 5.3: Tabel 2 hasil pengujian *tweet* yang dilakukan secara bersamaan

No	Akun penguji	Waktu *tweet*	*Tweet* yang dikirimkan pengguna	*Tweet* yang diterima pengguna
3	@ClaraKwaria	10:52	@KvinIink UNPAR ke PVJ	• @clara00010111 Jalan dari lokasi mulai Anda ke Jalan Ciumbuleuit sejauh kurang lebih 44 meter. 22:52:37 • @clara00010111 Naik angkot Ciumbuleuit - St. Hall (belok) di Jalan Ciumbuleuit, dan turun di Jalan Sederhana kurang lebih setelah 22:52:38 • @clara00010111 2,4 kilometer. 22:52:38 • @clara00010111 Jalan dari Jalan Sederhana ke tujuan akhir Anda sejauh kurang lebih 460 meter. 22:52:39 • @clara00010111 Untuk lebih lengkapnya dapat dilihat pada http:kiri.travel?start=unpar &finish=pvj&region=bdo 22:52:39
4	@clara00010111	10:52	@KvinIink UNPAR ke PVJ	• @clara00010111 Jalan dari lokasi mulai Anda ke Jalan Ciumbuleuit sejauh kurang lebih 44 meter. 22:52:34 • @clara00010111 Naik angkot Ciumbuleuit - St. Hall (belok) di Jalan Ciumbuleuit, dan turun di Jalan Sederhana kurang lebih setelah 22:52:34 • @clara00010111 2,4 kilometer. 22:52:34 • @clara00010111 Jalan dari Jalan Sederhana ke tujuan akhir Anda sejauh kurang lebih 460 meter. 22:52:35 • @clara00010111 Untuk lebih lengkapnya dapat dilihat pada http:kiri.travel?start=unpar &finish=pvj&region=bdo 22:52:35

Tabel 5.4: Tabel 3 hasil pengujian *tweet* yang dilakukan secara bersamaan

No	Akun penguji	Waktu *tweet*	*Tweet* yang dikirimkan pengguna	*Tweet* yang diterima pengguna
5	@ClaraKwaria	10:59	@KvinIink UNPAR ke BIP	• @ClaraKwaria Jalan dari lokasi mulai Anda ke Jalan Ciumbuleuit sejauh kurang lebih 44 meter. 23:00:04 • @ClaraKwaria Naik angkot Ciumbuleuit - St. Hall (lurus) di Jalan Ciumbuleuit, dan turun di Jalan Cihampelas kurang lebih setelah 23:00:04 • @ClaraKwaria 3,3 kilometer. 23:00:04 • @ClaraKwaria Jalan dari Jalan Cihampelas ke Jalan Wastukancana sejauh kurang lebih 17 meter. 23:00:05 • @ClaraKwaria Naik angkot Ciroyom - Antapani di Jalan Wastukancana, dan turun di Jalan Aceh kurang lebih setelah 1,4 kilometer. 23:00:06 • @ClaraKwaria Jalan dari Jalan Aceh ke tujuan akhir Anda sejauh kurang lebih 124 meter. 23:00:06 • @ClaraKwaria Untuk lebih lengkapnya dapat dilihat pada http:kiri.travel?start=unpar &finish=bip&region=bdo 23:00:02 23:00:06

Tabel 5.5: Tabel 4 hasil pengujian *tweet* yang dilakukan secara bersamaan

No	Akun penguji	Waktu *tweet*	*Tweet* yang dikirimkan pengguna	*Tweet* yang diterima pengguna
6	@clara00010111	10:59	@KvinIink UNPAR to BIP	• @clara00010111 Walk about 44 meter from your starting point to Jalan Ciumbuleuit. 22:59:59 • @clara00010111 Take angkot Ciumbuleuit - St. Hall (lurus) at Jalan Ciumbuleuit, and alight at Jalan Cihampelas about 3.3 kilometer 23:00:00 • @clara00010111 later. 23:00:00 • @clara00010111 Walk about 17 meter from Jalan Cihampelas to Jalan Wastukancana. 23:00:01 • @clara00010111 Take angkot Ciroyom - Antapani at Jalan Wastukancana, and alight at Jalan Aceh about 1.4 kilometer later. 23:00:01 • @clara00010111 Walk about 124 meter from Jalan Aceh to your destination. 23:00:02 • @clara00010111 For futher information you can visit http:kiri.travel?start=unpar &finish=bip&region=bdo 23:00:02
7	@ClaraKwaria	10:02	@KvinIink bantuan	• @ClaraKwaria Format penggunaan *Twitter bot* untuk mencari jalur transportasi publik adalah... 23:00:06 • @ClaraKwaria 'Lokasi awal' ke 'lokasi tujuan', contoh : BIP ke PVJ. 23:00:06
8	@clara00010111	10:02	@KvinIink help	• @clara00010111 For using this *Twitter bot* for searching public transport route, you can mention... 23:00:06 • @clara00010111 'First location' to 'second location', example : BIP to PVJ 23:00:06

BAB 6

KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dan saran dari penelitian yang dilakukan.

6.1 Kesimpulan

Berikut ini adalah kesimpulan yang diambil oleh penulis berdasarkan penelitian yang telah dilakukan:

1. *Twitter bot* sudah dapat menangkap *tweet* dari pengguna, lalu dapat melakukan *reply* kepada pengguna.
2. *Twitter bot* dapat menerima *tweet* yang di-*mention* kepada akun *Twitter bot* secara *real time* dan sudah dapat melakukan *reply* dengan benar sesuai hasil yang diberikan oleh KIRI API.
3. Instruksi dari KIRI API sudah dapat dipecah-pecah dan dapan disampaikan dengan baik kepada pengguna dalam bentuk *tweet*.

6.2 Saran

Berdasarkan hasil kesimpulan yang telah dipaparkan, penulis memberi saran sebagai berikut:

1. Ketika pengguna ingin mencari lokasi jalan, penulisan nama jalan harus lengkap. Sebagai contoh adalah jalan mekar wangi, jalan kopo. Jika penulisan hanya mekar wangi atau kopo saja, maka terjadi kemungkinan pencarian lokasi tidak akan ditemukan.
2. Membuat akun *Twitter bot* menjadi premium agar tidak mengalami adanya keterbatasan *tweet* per harinya.
3. Pada biodata informasi akun *Twitter bot*, sebaiknya diberitahu cara penggunaan *Twitter bot* untuk mencari jalur transportasi publik agar pengguna bisa langsung mencoba.
4. Pengguna tidak harus melakukan *mention* kepada akun *Twitter Bot*, pengguna dapat memanfaatkan fungsi *hashtag* yang telah diberikan Twitter.
5. Mengatasi format *tweet* agar tidak terlihat seperti *spam*.

DAFTAR REFERENSI

[1] T. O'Reilly, *The Twitter Book*. O'Reilly Media, Inc, 2009.

[2] "Twitter documentation." https://dev.twitter.com/overview/documentation, 2014. Accessed: 2014-8-20.

[3] "Json documentation." http://json.org/, 2014. Accessed: 2015-4-05.

[4] "Kiri api v2 documentation." https://bitbucket.org/projectkiri/kiri_api/wiki/KIRI%20API%20v2%20Documentation, 2014. Accessed: 2014-8-21.

[5] "Twitter4j documentation." http://twitter4j.org/javadoc/index.html, 2014. Accessed: 2014-8-29.

LAMPIRAN A

KODE PROGRAM KELAS MAIN

Listing A.1: Main.java

```
{
        import twitter4j.FilterQuery;
        import twitter4j.TwitterStream;
        import twitter4j.TwitterStreamFactory;

        public class Main {
                public static void main(String[] args){
                        TwitterStream twitterStream = new TwitterStreamFactory().getInstance();
                        TwitterGateway twittergateway = new TwitterGateway();

                        FilterQuery fq = new FilterQuery();
                        String keywords[] = {twittergateway.screenName};

                        fq.track(keywords);

                        twitterStream.addListener(twittergateway);
                        twitterStream.filter(fq);
                }
        }

}
```

LAMPIRAN B

KODE PROGRAM KELAS TWITTERGATEWAY

Listing B.1: TwitterGateway.java

```
{
        /*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


import java.text.DateFormat;
import java.text.SimpleDateFormat;
import twitter4j.*;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.logging.Level;

public final class TwitterGateway implements StatusListener{
    public static final String screenName = "@Kvinlink";
    private String user;
    private String location[];
    private String latlon[] = new String[2];
    private RoutingResponse routingResponse;
    private Step[] step;
    private Steps steps;
    DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
    Date date = new Date();

    @Override
    public void onStatus(Status status) {
        user = status.getUser().getScreenName();

        String mentionStatus = status.getText();
        System.out.println("@" + user + " - " + mentionStatus);
        String paramScreenName = screenName.toLowerCase();
        mentionStatus = mentionStatus.toLowerCase().replace(paramScreenName, "").trim();
        String locale = new String();
        if(mentionStatus.contains(" to "))
        {
            location = mentionStatus.split(" to ");
            locale = "en";
        }
        else if(mentionStatus.contains(" ke "))
        {
            location = mentionStatus.split(" ke ");
            locale = "id";
        }
        else
        {
            location = null;
        }
        boolean statusLocation1 = false;
        boolean statusLocation2 = false;

        if(mentionStatus.equals("help") || mentionStatus.equals(("bantuan")))
        {
            try {
                if(mentionStatus.equals("help"))
                {
                    Tweet(user,"For using this Twitter bot for searching public transport route, you can
                        mention...");
```

```
61                      Tweet(user,"'First location' to 'second location', example : BIP to PVJ");
62                  }
63                  else
64                  {
65                      Tweet(user,"Format penggunaan Twitter bot untuk mencari jalur transportasi publik
                            adalah...");
66                      Tweet(user,"'Lokasi awal' ke 'lokasi tujuan', contoh : BIP ke PVJ.");
67                  }
68              } catch (TwitterException ex) {
69                  System.out.println("Tweet help error");
70              }
71          }
72          else if(location.length == 2 && !location[0].contains("@") && !location[1].contains("@"))
73          {
74              try {
75                  if(location[0].equals(location[1]))
76                  {
77                      if(locale.equals("id"))
78                      {
79                          Tweet(user, "Pencarian tidak dapat dilakukan karena lokasi awal dan lokasi tujuan
                                sama");
80                      }else if(locale.equals("en"))
81                      {
82                          Tweet(user, "Route can't be found. Starting location and destination are similar")
                                ;
83                      }
84                  }
85                  else
86                  {
87                      System.out.println("Lokasi 1 : "+location[0].trim());
88                      System.out.println("Lokasi 2 : "+location[1].trim());
89                      String destination1 = KIRIGateway.GetLatLong(location[0]);
90                      String destination2 = KIRIGateway.GetLatLong(location[1]);
91
92                      JSONObject objDest1 = new JSONObject(destination1);
93                      JSONObject objDest2 = new JSONObject(destination2);
94
95                      JSONObject res1 = objDest1.getJSONArray("searchresult").getJSONObject(0);
96                      String hasilDest1 = res1.getString("placename");
97                      latlon[0] = res1.getString("location");
98                      if(hasilDest1 != null)
99                      {
100                         statusLocation1 = true;
101                     }
102
103                     JSONObject res2 = objDest2.getJSONArray("searchresult").getJSONObject(0);
104                     String hasilDest2 = res2.getString("placename");
105                     latlon[1] = res2.getString("location");
106                     if(hasilDest2 != null)
107                     {
108                         statusLocation2 = true;
109                     }
110
111                     //Mendapatkan hasil pencarian lalu dimasukan ke JSONArray paramSteps untuk dipisah-
                            pisah lalu dimasukan ke RoutingResponse
112                     String hasilPencarian = KIRIGateway.GetTrack(latlon[0], latlon[1], locale);
113                     JSONObject objTrack = new JSONObject(hasilPencarian);
114                     JSONObject routingresults = objTrack.getJSONArray("routingresults").getJSONObject(0);
115                     JSONArray paramSteps = routingresults.getJSONArray("steps");
116                     //buat variable step, steps, dan routing response
117                     step = new Step[paramSteps.length()];
118                     for (int i = 0; i < step.length; i++) {
119                         step[i] = new Step(paramSteps.getJSONArray(i).getString(3) + "");
120                     }
121                     steps = new Steps(step);
122
123                     routingResponse = new RoutingResponse(objTrack.getString("status"), steps);
124                     if(routingResponse.getStatus().equals("ok")){
125                         for (int i = 0; i < routingResponse.getRoutingResult().getSteps().length ; i++) {
126                             date = new Date();
127                             Tweet(user, routingResponse.getRoutingResult().getSteps()[i].
                                    getHumanDescription());
128                         }
129                         if(locale.equals("id"))
130                         {
131                             Tweet(user, "Untuk lebih lengkapnya dapat dilihat pada http://kiri.travel?
                                    start=" + location[0].replace(" ", "%20") + "&finish="+ location[1].
                                    replace(" ", "%20") + "&region=bdo");
132                         }else if(locale.equals("en"))
133                         {
134                             Tweet(user, "For futher information you can visit http://kiri.travel?start=" +
                                    location[0].replace(" ", "%20") + "&finish="+ location[1].replace(" ",
```

```
                                "%20") + "&region=bdo");
135                         }
136
137                         for (int i = 0; i < routingResponse.getRoutingResult().getSteps().length ; i++) {
138                             System.out.println("@"+user + " " + routingResponse.getRoutingResult().
                                    getSteps()[i].getHumanDescription());
139                         }
140                         System.out.println("@"+user + " Untuk lebih lengkap silahkan lihat di http://kiri.
                                travel?start=" + location[0].replace(" ", "%20") + "&finish="+ location[1].
                                replace(" ", "%20") + "&region=bdo");
141                     }else{
142                         System.out.println("status error");
143                     }
144                 }
145             } catch (Exception ex) {
146                 try {
147                     if(!statusLocation1)
148                     {
149                         date = new Date();
150                         Tweet(user,location[0] + " tidak ditemukan");
151                         System.out.println("@"+user+" "+location[0] + " tidak ditemukan");
152                     }
153                     else if(!statusLocation2)
154                     {
155                         date = new Date();
156                         Tweet(user,location[1] + " tidak ditemukan");
157                         System.out.println("@"+user+" "+location[1] + " tidak ditemukan");
158                     }
159                     else
160                     {
161                         date = new Date();
162                         Tweet(user,"Gangguan Koneksi");
163                         System.out.println("Gangguan Koneksi");
164                     }
165                     //java.util.logging.Logger.getLogger(TwitterGateway.class.getName()).log(Level.SEVERE,
                            null, ex);
166                 } catch (TwitterException ex1) {
167                     System.out.println("Error2");
168                     java.util.logging.Logger.getLogger(TwitterGateway.class.getName()).log(Level.SEVERE,
                            null, ex1);
169                 }
170             }
171         }
172
173     }
174
175     @Override
176     public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice) {
177         System.out.println("Got a status deletion notice id:" + statusDeletionNotice.getStatusId());
178     }
179
180     @Override
181     public void onTrackLimitationNotice(int numberOfLimitedStatuses) {
182         System.out.println("Got track limitation notice:" + numberOfLimitedStatuses);
183     }
184
185     @Override
186     public void onScrubGeo(long userId, long upToStatusId) {
187         System.out.println("Got scrub_geo event userId:" + userId + " upToStatusId:" + upToStatusId);
188     }
189
190     @Override
191     public void onException(Exception ex) {
192         System.out.println("onException");
193         ex.printStackTrace();
194     }
195
196     @Override
197     public void onStallWarning(StallWarning sw) {
198         System.out.println("onStallWarning");
199         throw new UnsupportedOperationException("Not supported yet.");
200     }
201
202     public void Tweet(String user, String paramStatusUpdate) throws TwitterException
203     {
204         Twitter twitter = new TwitterFactory().getInstance();
205         int userLength = user.length();
206         int maxTweet = 140 - (userLength + 2 + 9);
207
208         String[] tampung = paramStatusUpdate.split(" ", 0); //misahin semua kata jadi array of kata
209
210         StatusUpdate[] statusUpdate = new StatusUpdate[(paramStatusUpdate.length() / maxTweet) + 1];
```

```
211         String[] tampungStatusUpdate = new String[statusUpdate.length];
212 
213         int increment = 0;
214         int incTampung = 0;
215         for (int i = 0; i < tampungStatusUpdate.length; i++) {
216             tampungStatusUpdate[i] = "@" + user + " ";
217         }
218 
219         while(increment < statusUpdate.length){
220 
221             while(incTampung < tampung.length){
222                 tampungStatusUpdate[increment] += tampung[incTampung];
223                 if(tampungStatusUpdate[increment].length() >= 131){ ;
224 
225                     tampungStatusUpdate[increment] = tampungStatusUpdate[increment].substring(0,
                            tampungStatusUpdate[increment].length() - tampung[incTampung].length());
226                     tampungStatusUpdate[increment] += " "+ dateFormat.format(date);
227                     System.out.println(tampungStatusUpdate[increment]);
228                     increment++;
229                     break;
230                 }else{
231                     tampungStatusUpdate[increment] += " ";
232                 }
233                 incTampung++;
234                 if(incTampung >= tampung.length){
235                     tampungStatusUpdate[increment] += " "+ dateFormat.format(date);
236                     System.out.println(tampungStatusUpdate[increment]);
237                     increment++;
238                 }
239             }
240         }
241 
242         for (int i = 0; i < statusUpdate.length; i++) {
243             statusUpdate[i] = new StatusUpdate(tampungStatusUpdate[i]);
244         }
245         try {
246             for (int i = 0; i < statusUpdate.length; i++) {
247                 twitter.updateStatus(statusUpdate[i]);
248             }
249         } catch (TwitterException ex) {
250             twitter.updateStatus("@" + user + " Maaf anda sudah penah melakukan pencarian ini sebelumnya.
                    " + dateFormat.format(date) );
251             System.out.println("Error : " + ex);
252         }
253     }
254 }
255 }
```

LAMPIRAN C

KODE PROGRAM KELAS KIRIGATEWAY

Listing C.1: KIRIGateway.java

```
{
        import java.io.BufferedReader;
        import java.io.InputStreamReader;
        import java.net.HttpURLConnection;
        import java.net.URL;
        import java.net.URLEncoder;
        public class KIRIGateway {
                 public static String GetLatLong(String destination) throws Exception {
                                String url = "http://kiri.travel/handle.php?version=2&mode=searchplace&
                                    region=bdo&querystring=" + URLEncoder.encode(destination.trim(), "UTF
                                    -8")+"&apikey=889C2C8FBB82C7E6";

                                URL obj = new URL(url);
                                HttpURLConnection con = (HttpURLConnection) obj.openConnection();

                                con.setRequestMethod("GET");

                                int responseCode = con.getResponseCode();

                                BufferedReader in = new BufferedReader(
                                                new InputStreamReader(con.getInputStream()));
                                String inputLine;
                                StringBuffer response = new StringBuffer();

                                while ((inputLine = in.readLine()) != null) {
                                                response.append(inputLine);
                                }
                                in.close();

                                return response.toString();
            }

                        public static String GetTrack(String dest1, String dest2) throws Exception {
                                String url = "http://kiri.travel/handle.php?version=2&mode=findroute&
                                    locale=id&start="+dest1+"&finish="+dest2+"&presentation=desktop&apikey
                                    =889C2C8FBB82C7E6";
                                URL obj = new URL(url);
                                HttpURLConnection con = (HttpURLConnection) obj.openConnection();

                                con.setRequestMethod("GET");

                                int responseCode = con.getResponseCode();

                                BufferedReader in = new BufferedReader(new InputStreamReader(con.
                                    getInputStream()));
                                String inputLine;
                                StringBuffer response = new StringBuffer();

                                while ((inputLine = in.readLine()) != null) {
                                                response.append(inputLine);
                                }
                                in.close();

                                return response.toString();
            }
        }

}
```

LAMPIRAN D

KODE PROGRAM KELAS ROUTINGRESPONSE

Listing D.1: RoutingResponse.java

```
{
        public class RoutingResponse {
                public String status;
                public Steps routingResult;

                public RoutingResponse(String paramStatus, Steps paramRoutingResult){
                        this.status = paramStatus;
                        this.routingResult = paramRoutingResult;
                }

                public RoutingResponse() {
                        this.status = "";
                        this.routingResult = null;
                }
                public String getStatus() {
                        return status;
                }

                public void setStatus(String status) {
                        this.status = status;
                }

                public Steps getRoutingResult() {
                        return routingResult;
                }

                public void setRoutingResult(Steps routingResult) {
                        this.routingResult = routingResult;
                }
        }
}
```

LAMPIRAN E

KODE PROGRAM KELAS STEP

Listing E.1: Step.java

```
{
        class Step {
                public String humanDescription;

                public Step(String paramHumanDescription){
                        this.humanDescription = paramHumanDescription;
                }

                public Step() {
                        this.humanDescription = "";
                }
                public String getHumanDescription() {
                        return humanDescription;
                }

                public void setHumanDescription(String humanDescription) {
                        this.humanDescription = humanDescription;
                }
        }


}
```

LAMPIRAN F

KODE PROGRAM KELAS STEPS

Listing F.1: Steps.java

```
{
        class Steps {
                public Step[] steps;

                public Steps(Step[] paramSteps){
                        this.steps = paramSteps;
                }
                public Step[] getSteps() {
                        return steps;
                }

                public void setSteps(Step[] steps) {
                        this.steps = steps;
                }
        }
}
```