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
2014

DAFTAR ISI

Daftar Isi iii

Daftar Gambar v

Daftar Tabel vi

1 Pendahuluan 1
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 2
- 1.3 Tujuan . . . 2
- 1.4 Batasan Masalah . . . 3
- 1.5 Metode Penelitian . . . 3

2 Dasar Teori 5
- 2.1 Twitter . . . 5
- 2.2 Twitter API . . . 6
  - 2.2.1 *Search* API . . . 6
  - 2.2.2 Streaming API . . . 9
- 2.3 OAuth . . . 12
  - 2.3.1 Application-only authentication . . . 13
  - 2.3.2 *3-legged authorization* . . . 14
  - 2.3.3 *PIN-based authorization* . . . 14
- 2.4 KIRI API . . . 15
  - 2.4.1 *Routing Web Service* . . . 15
  - 2.4.2 *Search Place Web Service* . . . 17
  - 2.4.3 *Nearest Transports Web Service* . . . 17
- 2.5 Twitter4J . . . 18
  - 2.5.1 TwitterFactory . . . 19
  - 2.5.2 TwitterStream . . . 19
  - 2.5.3 TwitterStreamFactory . . . 20
  - 2.5.4 UserStreamListener . . . 21
  - 2.5.5 TweetsResources . . . 22
  - 2.5.6 OAuthSupport . . . 22
  - 2.5.7 RequestToken . . . 23
  - 2.5.8 AccessToken . . . 24
  - 2.5.9 Status . . . 24
  - 2.5.10 TweetsResources . . . 26

3 Analisis 27
- 3.1 Analisis Data . . . 27
  - 3.1.1 Analisis Twitter API . . . 27
  - 3.1.2 Analisis OAuth . . . 28
  - 3.1.3 Analisis KIRI API . . . 28

3.1.4 Analisis Twitter4J . . . 31
3.2 Analisis Perangkat Lunak . . . 32
3.2.1 Spesifikasi Kebutuhan Fungsional . . . 32
3.2.2 *Use Case Diagram* . . . 33
3.2.3 *Class Diagram* . . . 33

4 Perancangan Perangkat Lunak 35
4.1 Perancangan Perangkat Lunak . . . 35
4.1.1 Perancangan Kelas . . . 35
4.1.2 Sequence Diagram . . . 38
4.1.3 Perancangan Antar Muka . . . 38

5 Implementasi Dan Pengujian Aplikasi 43
5.1 Lingkungan Pembangunan . . . 43
5.2 Hasil Tampilan Antarmuka . . . 43
5.3 Pengujian . . . 45
5.3.1 Pengujian Fungsional . . . 45
5.3.2 Pengujian Eksperimental . . . 46

Daftar Referensi 49

DAFTAR GAMBAR

2.1 Ilustrasi sign in 14
2.2 Contoh PIN-based authorization 15

3.1 Use case Twitter Bot 33
3.2 *Class Diagram* Twitter Bot 34

4.1 Class Diagram Pembuatan Twitter Bot untuk Mencari Jalur Transportasi Publik . . 35
4.2 Sequence Diagram Pembuatan Twitter Bot untuk Mencari Jalur Transportasi Publik 39
4.3 Homepage Twitter versi mobile 41
4.4 Tampilan untuk melakukan tweet 42

5.1 Homepage Twitter versi mobile 44
5.2 Tampilan untuk melakukan tweet 45
5.3 Hasil streaming tweet 45
5.4 Hasil balasan tweet kepada user 45
5.5 Starting program 47
5.6 Compose new Tweet 47
5.7 Tweet Kepada User @kiriupdate 47
5.8 Tweet Diterima oleh Aplikasi 47
5.9 Tweet Rute Jalur Transportasi Publik 48

DAFTAR TABEL

2.1 Contoh berbagai macam pencarian *tweet* 7
2.2 Contoh *mapping* dari *search query* ke *query* pengkodean URL 8
2.3 Parameter POST statuses/*filter* 10
2.4 Parameter GET *statuses/sample* 10
2.5 Parameter GET *statuses/firehose* 10
2.6 Parameter GET *user* 11
2.7 Parameter *Routing Web Service* 16
2.8 Tabel parameter *Search Place Web Service* 17
2.9 Tabel parameter *Nearest Transports Web Service* 18

3.1 Skenario *Tweet* mencari informasi transportasi 33

5.1 Tabel Hasil pengujian fungsionalitas pada Aplikasi Twitter Bot untuk mencari jalur transportasi publik 46

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 pengguna internet di Indonesia capai 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 social media atau jejaring social online.

Twitter adalah salah satu layanan jejaring sosial online yang memungkinkan pengguna memposting pesan berbasis teks hingga 140 karakter. Pengguna Twitter menyebutnya sebagai *tweet*. *Tweet* ini akan meneruskan pesan singkat yang ditujukan ke semua *follower* suatu akun[2].*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 *reply* atau lebih dikenal dengan nama *mention*[3]. 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 ramai di Bandung, maka akun @kviniink membuat *mention tweet* yang berisikan "@infobdg Halo saya ingin bertanya apa saja mall yang sedang ramai di Bandung yah?".

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 sudah lebih dari 12000 unit kendaraan[4]. 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. Seiring dengan perkembangan teknologi, menaiki transportasi publik menjadi semakin mudah. Dengan adanya KIRI di Indonesia terutama di daerah Bandung, masyarakat dapat naik

[1]Kominfo bint005 , Pengguna Internet di Indonesia Capai 82 Juta, http://www.aviationtoday.com/rw/military/attack/High-Sea-Piracy-Crisis-in-Aden_32500.html, pada tanggal 15 April 2015 pukul 12.58

[2]Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 7

[3]Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 9

[4]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

transportasi publik tanpa harus mengetahui terlebih dahulu kendaraan yang harus dinaikinya. Dengan adanya KIRI, pengguna hanya perlu tahu tempat asal dan tempat tujuan untuk dapat menaiki transportasi publik di Bandung ini.

KIRI API adalah aplikasi pihak ketiga yang memungkinkan *programmer* mendapatkan data tentang info jalur transportasi publik. Twitter API adalah aplikasi pihak ketiga yang memungkinkan *programmer* melakukan manipulasi dan pengolahan data di Twitter. Dengan memanfaatkan KIRI API dan Twitter API peneliti akan membuat program yang dapat membalas *tweet* untuk mencari jalur transportasi publik. Program yang dibuat akan bersifat *real time* sehingga jika seseorang melakukan mention kepada bot pencari jalur maka bot akan menangkapnya dan membalas mention tersebut berupa jalur yang harus ditempuh. Contoh dari jalannya program adalah ketika akun bernama @kviniink melakukan *mention* kepada @kiriupdate untuk bertanya jalur transportasi publik "@kiriupdate #find bip to ip". Maka Twitter bot @kiriupdate akan mendengarkan mention dari akun @kviniink lalu mention tersebut akan diolah oleh server dan akan di-reply dengan tiga buah tweet "@kviniink istana plaza to bandung indah plaza", "@kviniink Walk about 135 meter from your starting point to Jalan Aceh." , "@kviniink Take angkot Ciroyom - Antapani at Jalan Aceh, and alight at Jalan Pajajaran about 3.6 kilometer later.", "@kviniink Walk about 93 meter from Jalan Pajajaran to your destination.". Karena keterbatasan 140 karakter maka tweet akan dipecah sesuai dengan instruksi yang dikirimkan dari KIRI API.

Oleh karena itu, dalam penelitian ini akan dibangun sebuah perangkat lunak yang dapat memudahkan pengguna dalam mencari jalur transportasi publik. Sebuah applikasi yang menggabungkan jejaring sosial online Twitter dengan KIRI API. Jadi pengguna bisa melakukan tweet kepada Twitter bot yang dibuat dengan format tertentu yang berisikan tempat asal dan tempat yang akan dituju. Lalu pengguna akan menerima balasan tweet berupa rute jalan yang harus ditempuhnya.

1.2 Rumusan Masalah

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

- Bagaimana membuat *Twitter bot* untuk mencari jalur transportasi publik?
- Bagaimana membuat *Twitter bot* untuk dapat merespon secara real time?
- Bagaimana memformat petunjuk rute perjalanan dalam keterbatasan tweet 140 karakter?

1.3 Tujuan

Tujuan dari penelitian ini adalah:

- Membuat aplikasi *Twitter bot* untuk mencari jalur transportasi publik.
- Membuat aplikasi Twitter yang bekerja secara *real time*.
- 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:

- Input hanya mencakup Kota Bandung saja.
- Input yang diinputkan harus benar, memiliki asal dan tujuan yang jelas di Kota Bandung.
- Hasil yang dikeluarkan berupa tweet jalur transportasi publik.
- Media transportasi publik yang digunakan adalah angkutan umum.
- 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. Membuat TwitterBot sederhana
4. Melakukan analisis terhadap teori-teori yang sudah dipelajari, guna membangun perangkat lunak yang dimaksud.
5. Melakukan pengujian terhadap system yang sudah 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

Twitter adalah layanan yang memungkinkan pengguna untuk mengirim pesan menggunakan 140 karakter atau kurang. Pesan tersebut dapat diadaptasikan melalui teks, aplikasi *mobile*, atau web. (referensi dari buku Sams teach yourself the twitter api) Berikut ini adalah daftar istilah umum pada Twitter:

Twitter adalah salah satu layanan jejaring sosial online yang memungkinkan pengguna melakukan *posting* pesan berbasis teks hingga 140 karakter[2].

- *Tweet*

  Posting pada Twitter disebut sebagai *tweet*. *Tweet* ini akan meneruskan pesan singkat yang ditujukan ke semua *follower* suatu akun[1]. Contohnya adalah seorang akun @kviniink ingin menuliskan bahwa hari ini cuaca cerah, maka @kviniink akan men-*tweet* 'Hari ini cerah yah..'. *Tweet* juga bisa menyertakan *link* ke video, foto, atau media lain di internet selain teks biasa. URL *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, contoh saat pengguna memasukkan link `http://www.chacha.com/gallery/7253/15-movies-that-make-guys-cry`, maka 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*[2]. Sebagai contoh, diketahui akun bernama @kviniink mem-*follow* @infobdg untuk mengetahui perkembangan apa saja

[1] Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 7
[2] Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 9

yang tejadi di Kota Bandung. Lalu akun @kviniink ingin bertanya tentang info mall yang ramai di Kota Bandung, maka akun @kviniink membuat *mention tweet* yang berisikan "@infobdg Halo saya ingin bertanya apa saja mall yang sedang ramai di Bandung yah?".

- *Retweet*

  Retweet ini merupakan salah satu yang paling penting dari Twitter. *Retweet* ini berguna ketika pengguna menemukan *tweet* menarik dan berbagi *tweet* tersebut dengan *follower* akun tersebut (*follower*). *Retweet* ini juga secara tidak langsung mengatakan bahwa "saya menghormati anda dan pesan yang anda buat". *Retweet*.

- *Hashtag*

  Sebuah fitur yang diciptakan oleh Twitter untuk membantu pencarian kata kunci dan penandaan suatu diskusi.

- *Direct Message*(DM)

  Direct message digunakan untuk mengirim pesan yang bersifat *private* antara dua orang. Orang yang mengirim *direct message* ini hanya bisa untuk orang yang mengikuti akun tersebut.

- *Timeline*

  Timeline adalah sekumpulan *tweet* dari semua orang yang anda *follow* lalu akan ditampilkan di halaman utama.

2.2 Twitter API

Twitter API adalah aplikasi pihak ketiga yang memungkinkan *programmer* melakukan manipulasi dan pengolahan data di Twitter. Twitter API tidak seperti API pada umumnya karena Twitter memaparkan hampir semuanya termasuk *setup account* dan informasi kostumisasi[1]. Ini adalah salah satu bentuk pendekatan dari Twitter yang berfokus pada jaringan dan memungkinkan developer memiliki hak untuk berpikir '*out of the box*' untuk membuat aplikasi yang mereka inginkan. Tetapi tetap akan terjadi keterbatasan yang dimiliki Twitter API, yaitu :

- Hanya bisa men-update 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.
- OAuth diijinkan 350 permintaan 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 Twitter itu sendiri. API ini difokuskan kepada relefansi, bukan terhadap kelengkapan data. Ini berarti bahwa beberapa *Tweet* dan pengguna akan hilang dari hasil pencarian.

Bagaimana cara membuat sebuah *query* Cara berbaik dalam membuat sebuah *query*, melakukan percobaan yang valid dan mengembalikan tweet yang sesuai adalah dengan mencobanya di twitter.com/search. URL yang ditampilkan pada browser akan berisi sintaks *query* yang sesuai agar dapat digunakan kempali pada API *endpoint*. Berikut adalah contohnya:

1. Melakukan pencarian untuk *tweet* yang direferensikan kepada akun @twitterapi. Pertama kita harus melakukan pencarian pada twitter.com/search.
2. Lakukan pengecekan dan salin URL yang ditampilkan. Sebagai contoh didapatkan URL seperti ini, 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 bahwa *request* sudah diotentifikasi. Perlu diingat juga bahwa hasil pencarian yang dilakukan di twitter.com dapat menghasilkan hasil yang sudah sangat lama, sedangkan Search API hanya melayani tweet dari seminggu terakhir. Contoh berbagai macam pencarian dapat dilihat pada tabel 2.1:

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 kata "*root*".
#haiku	Mengandung *hashtag* "*haiku*".
from:alexiskold	Dikirim melalui *user* "alexiskold".
to:techcrunch	Dikirimkan kepada *user* "techcrunch".
@mashable	Mereferensikan kepada *user* "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 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 dipost melalui twitterfeed.

Tabel 2.1: Contoh berbagai macam pencarian *tweet*

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.

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

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

Additional parameters Terdapat parameter tambahan yang dipergunakan untuk hasil 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 poluler 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 pertama akan mencoba menemukan *tweet* yang memiliki *latitude* yang sudah dimasukan kedalam *query* geocode, jika tidak berhasil maka API akan mencoba menemukan *tweet* yang dibuat oleh pengguna yang lokasi profilenya terdapat pada *latitude* tersebut. Artinya adalah hasil pencarian mungkin menerima *tweet* yang tidak mencakup informasi *latidute* atau *longitude*.
- ***Language***. Bahasa dapat dijadikan parameter untuk mencari tweet yang sesuai dengan bahasa tersebut.
- ***Iterating in a result set***. Parameter seperti *count, until, since_id, max_id* memungkinkan untuk mengkontrol bagaimana iterasi melalui hasil pencarian.

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

Contoh Pencarian Ketika anda mengikuti suatu acara yang sedang berlangsung, anda tertarik untuk mencarinya dengan melihat tweet yang paling baru dan menggunakan *hastag* dari acara tersebut, maka langkah-langkah yang dilakukan adalah:

- Anda ingin mencari *tweet* yang paling baru dengan menggunakan hastag #*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:

- Anda ingin mencari *tweet* yang poluler 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 pengembang dengan kebutuhan data yang intensif. Contohnya jika mencari cara untuk membangun sebuah data produk *data-mining* atau tertarik dalam analisis penelitian. *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 terjadi.

- *Public stream*

  Steaming data publik yang mengalir melalui Twitter. Dipergunakan untuk mengikuti sebuah *user* atau topik tertentu. Selain itu juga *public stream* digunakan untuk *data mining*.

- *User Stream*

  Single-user streams, mengandung hampir semua data yang berhubungan dengan satu *user* tertentu.

- *Site Stream*

  Versi dari *multi-user stream*. *Site stream* harus 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*, aplikasi akan menyampaikan umpan Tweet tanpa perlu khawatir akan keterbatasan *rate limit*.

Endpoints

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

POST statuses/*filter* POST *filter* ini mengembalikan status publik yang sesuai dengan satu atau lebih predikat yang telah di filter. *Multiple parameter* memungkinkan klien untuk menggunakan koneksi tunggal untuk ke *Streaming* API. Antara GET dan POST *request* keduanya didukung tetapi 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. Hal ini memungkinkan akses hingga 400 kata kunci, 5000 *follow users*. Perintah ini dikembalikan dalam format JSON, memerlukan otentifikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk parameter ini dapat dilihat pada tabel 2.3

follow	Tanda koma memisahkan list user ID, hal ini menunjukkan pengguna untuk kembali ke status untuk stream.
track	Kata pencarian untuk track. Fase kata kunci dipisahkan oleh tanda koma.
locations	Menentukan lokasi yang dilacak.
delimited	Menentukan apakah pesan harus dibatasi limitnya.
stall_ warnings	Menentukan apakah pesan warning harus dikirim atau tidak.

Tabel 2.3: Parameter POST statuses/*filter*

GET *statuses/sample* Mengembalikan *random* sampel dari semua status publik. *Tweet* akan dikembalikan dengan cara seperti biasa, jadi jika terdapat dua client yang terhubung dengan *endpoint* ini, maka mereka akan melihat Tweet yang sama. Perintah ini dikembalikan dalam format JSON, memerlukan otentifikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk parameter ini dapat dilihat pada tabel 2.4

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

Tabel 2.4: Parameter GET *statuses/sample*

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 maka dapat memuaskan pengguna. Perintah ini dikembalikan dalam format JSON, memerlukan otentifikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk parameter ini dapat dilihat pada tabel 2.5

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

Tabel 2.5: Parameter GET *statuses/firehose*

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. Proses pengembalian data oleh *streaming* API dilakukan dengan cara mengikuti petunjuk dalam pengolahan data *streaming*.

Koneksi Setiap *user* hanya dapat membuat satu koneksi yang terbuhung dengan *public endpoint* dan jika melakukan koneksi ke *public stream* lebih dari satu kali dengan menggunakan *user* 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 spesific untuk pengguna yang sudah diotentifikasi. *User Stream* tidak dimaksudkan untuk koneksi server ke server, Jika anda perlu membuat koneksi atas nama beberapa user dari mesin yang sama maka lebih baik menggunakan *site stream*. Perintah ini dikembalikan dalam format JSON, memerlukan otentifikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk parameter ini dapat dilihat pada tabel 2.6

Endpoints

- 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 untuk user yang sudah diotentifikasi atau dikirim juga kepada akun yang difollow oleh akun yang sudah diotentifikasi tersebut.
replies	Menentukan apakah harus mengembalikan @replies.
follow	Termasuk tweet public tambahan dari daftar yang disediakan 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 diterdiri dari array of integer dan array of string.

Tabel 2.6: Parameter GET *user*

Koneksi Meminimalkan jumlah koneksi suatu aplikasi untuk membuat *user stream*. Setiap user Twitter terbatas hanya untuk beberapa koneksi *user streams* per aplikasi OAuth, terlepas dari IP. Setelah mencapai batasnya maka koneksi tertua atau terlama akan diberhentikan secara otomatis. *User* login dari beberapa instansi dari aplikasi AOuth yang sama akan mengalami siklus koneksi yaitu akan dihubungan dan diputuskan satu sama lain.

Sebuah aplikasi harus dapat mengatasi HTTP 420 *error code* yang memberitahukan bahwa suatu akun sudah terlalu sering *login*. Oleh karena itu *user* yang seperti itu akan secara otomatis di *banned* dari *User Stream* untuk tingkat *login* yang berlebihan. Untuk memulihkan akses *streaming user* harus menutup aplikasi tambahan yang ada, mungkin berjalan di perangkat atau *device* yang berbeda.

Perhatikan bahwa setiap aplikasi memiliki alokasinya masing-masing, sehingga *login* dari aplikasi pertama tidak akan mempengaruhi aplikasi ke dua begitu juga sebaliknya. Tetapi

menjalankan terlalu banyak salinan aplikasi pertama maupun ke dua akan menimbulkan masalah. Perhatikan juga bahwa jumlah koneksi yang serentak per alamat IP masih terbatas terlepas dari aplikasi yang ada.

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) mereka tanpa berbagi password mereka. Cara ini juga memberikan cara untuk memberikan akses yang terbatas(dalam satu lingkup atau durasi). Sebagai contoh, seorang pengguna web dapat memberikan layanan percetakan(*client*) untuk mengakses foto pribadinya yang disimpan di layanan berbagi foto(server) tanpa harus memberikan *username* dan *password*nya. Ia akan mengotentikasi langsung dengan layanan berbagi foto tersebut yang mengeluarkan layanan percetakan.

Dalam model otentikasi *client-server* tradisional, klian menggunakan kredensial untuk mengakses *recources hosted* oleh server. Di dalam model OAuth, klien (bukan pemilik *resource*, tetapi bertindak atas namanya) meminta akses ke *resource* yang dikenalkan oleh pemilik *resource* namun diselenggarakan oleh server.

Agar klien dapat mengakses *resource*, pertama-tama ia harus mendapatkan izin dari si pemilik *resource*. Izin ini dinyatakan dalam bentuk token dan mencocokan *shared-secret*. Tujuan dari token ini adalah untuk membuat pemilik *resource* untuk berbagi kepercayaan kepada klien. Berbeda dengan kepercayaan pemilik *resource*. Token dapat dikeluarkan dalam ruang lingkup terbatas, durasi yang terbatas, dan akan dicabut secara independen.

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.

API v1.1's Authentication Model Otentifikasi model baru terdapat dalam dua bentuk, dan keduanya masih memanfaatkan OAuth 1.0A

Application-user authentication Application-user authentication adalah bentuk paling umum dari otentikasi *resource* dalam pelaksanaan OAuth 1.0A Twitter sampai saat ini. Permintaan anda menandatangani baik untuk mengidentifikasi identitas aplikasi anda yang akan menyertakan izin untuk diberikan kepada pengguna. Hal ini bertujuan untuk dapat membuat panggilan API atas nama anda yang diwakili oleh akses token.

Application-only authentication Application-only authentication adalah bentuk dari otentifikasi dimana aplikasi anda membuat *API request* atas nama aplikasi itu sendiri tanpa

adanya konteks dari pengguna. Pemanggilan API masih terbatas dalam setiap *API method* .

2.3.1 Application-only authentication

Twitter menawarkan aplikasi yang mampu mengeluarkan permintaan otentifikasi atas nama aplikasi itu sendiri. Dengan menggunakan *Application-only authentication* anda tidak mempunyai konteks dari otentifikasi pengguna dan ini berarti setiap *request* API untuk endpoint akan membutuhkan konteks *user*, seperti memposting *tweet* tidak akan bekerja. Aplikasi yang akan di dapat adalah:

- Melihat *timeline*
- Mengakses *following* dan *follower* dari suatu *akun*
- Mencari dalam *tweet*
- mengambil informasi dari *user* manapun

Tetapi *application-only authentication* tidak bisa melakukan :

- Posting *tweet*
- Melakukan koneksi dengan *Streaming endpoint*
- Mencari *user* seseorang
- Menggunakan geo endpoint
- Mengakses DM

Auth Flow Langkah-langkah dari *application-only auth* terdiri dari beberapa langkah yaitu : Sebuah aplkasi dikodekan berdasarkan *consumer key* dan *secret* ke dalam satu set khusus yang dikodekan secara kredensial. Aplikasi membuat *request* ke POST OAuth2/*token endpoint* untuk merubah kredensial tersebut untuk *token bearer*. Ketika mengakses REST API, aplikasi menggunakan *token bearer* untuk otentifikasi. Kerena tidak ada kebutuhan duntuk menandatangani *request*, pendekatan ini lebih sederhana dari model standar OAuth 1.0a

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

SSL benar-benar dibutuhkan karena ini adalah cara otentifikasi yang aman. Oleh karena itu semua *request* (baik untuk mendapatkan atau menggunakan token) harus menggunakan endpoint HTTPS, yang juga merupakan syarat untuk menggunakan API v1.1.

Tidak ada konteks pengguna. Ketika mengeluarkan permintaan menggunakan *application-only auth*, tidak ada konsep *'current-user'*. Karena itu *endpoint* seperti POST status / *update* tidak akan berfungsi dengan *application-only auth*.

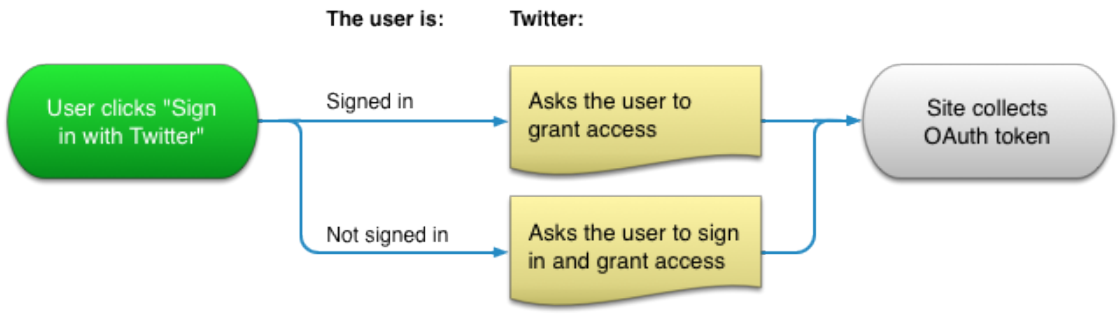
Rate limiting. Request yang dibuat atas nama pengguna tidak akan menguras ketersediaan *rate limit* dan *request* tidak akan menguras batas penggunaan **limit** dalam *user-based auth.*

2.3.2 3-*legged authorization*

Cara kerja dari *3-legged authorization* adalah dengan memberikan aplikasi yang anda buat untuk mengambil *access token* dengan cara melakukan *redirect* user dengan Twitter dan memberikan mereka sebuah otorisasi dari aplikasi yang anda buat. Cara kerja ini hampir identik dengan cara kerja yang dijelaskan pada implementasi *Sign in* dengan Twitter, hanya saja terdapat dua pengecualian yaitu:

- *GET oauth endpoint* digunakan sebagai pengganti GET oauth
- User akan selalu diminta untuk mengotorisasi akses ke aplikasi anda, bahkan jika akses sebelumnya telah diberikan

Beginilah ilustrasi interaksi *sign in* dengan menggunakan *following flowchart*



Gambar 2.1: Ilustrasi sign in

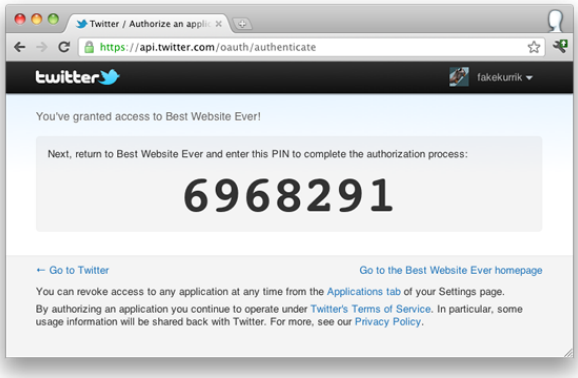
2.3.3 *PIN-based authorization*

cara kerja dari *PIN-based authorization* ini ditujukan untuk aplikasi yang tidak bisa mengakses atau menanamkan *web browser* untuk mengarahkan *user* kepada *authorization endpoint.* Contohnya adalah aplikasi 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*, perbedaannya terletak pada nilai dari *oauth_ callback* yang harus di set menjadi *oob* saat proses pemanggilan *POST oauth* atau *request_ token.*

Setelah aplikasi anda telah mendapatkan *GET oauth/authenticate* atau *GET oauth/authorize URL*, tampilkan URL kepada user agar mereka dapat menggunakan *web browser* untuk mengakses Twitter.

Ketika *callback oob* diminta dan user pengunjungi Twitter, *user* tidak akan dipindahkan secara otomatis ke aplikasi setelah menyetujui akses. Sebaliknya, mereka akan melihat kode PIN, dengan instruksi untuk kembali ke aplikasi dan memasukkan nilai dari kode PIN tersebut.



Gambar 2.2: Contoh PIN-based authorization

Aplikasi anda harus memungkinkan *user* untuk memasukkan *PIN code* ini untuk menyelesaikan *flow* tersebut. Nilai dari *PIN code* harus lolos sebagai *oauth_ verifier* untuk *POST oauth/access_ token request*. Semua *request* akan berjalan normal kedepannya.

2.4 KIRI API

KIRI API adalah aplikasi pihak ketiga yang memungkinkan *programmer* mendapatkan data tentang info jalur transportasi publik. KIRI API dapat diakses dengan beberapa cara. Semua *request* harus berisikan 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*

2.4.1 *Routing Web Service*

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

Berikut ini adalah parameter *request* yang diperlukan:

Listing 2.1: code *respond* pencarian rute

```
{
    "status": "ok" or "error"
    "routingresults": [
        {
            "steps": [
                [
                    "walk" or "none" or others,
                    "walk" or vehicle_id or "none",
                    ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
                    "human readable description, dependant on locale",
                    URL for ticket booking or null (future)
                ],
                [
                    "walk" or "none" or others,
                    "walk" or vehicle_id or "none",
                    ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
                    "human readable description, dependant on locale",
                    URL for ticket booking or null (future)
                ]
            ],
            "traveltime": any text string, null if and only if route is not found.
```

Parameter	*Valid values*	*Description*
version	2	Memberitahukan bahwa layanan yang dipakaia-dalah protokol versi 2
mode	"*findroute*"	Mengintruksikan layanan untuk mencari rute
locale	"en" or "id"	Respon bahasa yang digunakan
start	lat,lng (*both are decimal values*)	Titik awal *Latitude* dan *longitude*
finish	lat,lng (*both are decimal value*s)	Titik akhir *Latitude* dan *longitude*
presentation	"*mobile*" or "*desktop*"	Menentukan tipe presentasi untuk hasil keluaran. Contoh, jika tipe presentasi "*mobile*", maka link "tel:" akan ditambahkan di hasil.
apikey	16-digit *hexadecimals*	API *key* yang digunakan

Tabel 2.7: Parameter *Routing Web Service*

```
1          } ,
2          {
3              "steps": [ ... ],
4              "traveltime": "..."
5          } ,
6          {
7              "steps": [ ... ],
8              "traveltime": "..."
9          } ,
10         ...
11     ]
12 }
```

13 Ketika pencarian route berhasil yaitu dengan memberitahukan bahwa status "ok" seperti
14 pada baris 2, maka server juga harus memberikan hasil dari rute, yang berisikan langkah-
15 langkah yang disimpan di dalam array. Berikut ini adalah keterangan dari array tersebut:

16 • *Index* 0 (baris ke) berisikan "*walk*" atau "*none*" atau "*others*". Arti dari "*walk*"
17 adalah jalan kaki, "*none*" berarti rute jalan tidak ditemukan, dan "*others*" berarti
18 menggunakan kendaraan.

19 • *Index* ke 1 merupakan detail dari *index* ke 0 yang memiliki arti:

20 – Jika berisikan "*walk*" berarti *index* ini pun harus berisikan "*walk*",

21 – Jika berisikan "*none*" maka *index* ini pun harus berisikan "*none*",

22 – Selain itu, maka field ini berisikan id kendaraan yang dapat digunakan untuk
23 menambilkan gambar dari id kendaraan tersebut.

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

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

28 – %*fromicon* = sebuah ikon penanda yang menunjukkan titik awal atau "*from*".
29 Biasanya digunakan untuk mode presentasi perangkat bergerak.

30 – %*toicon* = sebuah ikon penanda yang menunjukkan titik akhir atau "*to*". Bia-
31 sanya 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.

2.4.2 *Search Place Web Service*

Search Place Web Service berguna untuk menemukan rute perjalanan berdasarkan *latitute* dan *longitude* koordinat, yang tidak nyaman bagi pengguna akhir. Layanan *Search Place Web Service* ini membantu untuk mengubah string teks untuk *latitude* dan *longitude*. Untuk permintaan *routing*, berikut parameter *request* yang diperlukan berikut penjelasannya:

version	2	Memberitahukan bahwa layanan yang dipakai adalah protokol veris 2
mode	"*searchplace*"	mengintruksikan 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

Tabel 2.8: Tabel parameter *Search Place Web Service*

Berikut format kembalian dari Kiri API:

Listing 2.2: 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*, yang merupakan array dari langkah-langkah dan masing-masing berisi tentang deskripsi dalam format pemetaan:

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

2.4.3 *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:

Berikut format kembalian dari Kiri API:

version	2	Memberitahukan bahwa layanan yang dipakai adalah protokol veris 2
mode	"*nearbytransports*"	mengintruksikan 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

Tabel 2.9: Tabel parameter *Nearest Transports Web Service*

Listing 2.3: 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*"), jika sukses maka respon akan mengembalikan array yang berisikan transportasi terdekat yang diurutkan dari yang terdeket ke yang terjauh. 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 kendaran.
- *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.

2.5 Twitter4J

Twitter4J merupakan *Java Library* untuk Twitter API. Dengan adanya Twitter4J ini, kita 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 saya buat saya membutuhkan beberapa *library* yang telah diberikan oleh Twitter4j. Berikut adalah *library* yang diperlukan:

2.5.1 TwitterFactory

- *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 diotentifikasi.
  - public Twitter getInstance(Authorization auth)
  - public static Twitter getSingleton()
    Mengembalikan *singleton* standar Twitter *instance*.

2.5.2 TwitterStream

- *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.

2.5.3 TwitterStreamFactory

- *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 diotentifikasi.
  - 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*.

2.5.4 UserStreamListener

- *Constant*
  - public interface UserStreamListener extends StatusListener

- *Methods*
  - void onDeletionNotice(long directMessageId, long userId)
  - void onFriendList(long[] friendIds)
  - void onFavorite(User source, User target, Status favoritedStatus)
  - void onUnfavorite(User source, User target, Status unfavoritedStatus)
  - void onFollow(User source, User followedUser)
  - void onUnfollow(User source, User unfollowedUser)
  - void onDirectMessage(DirectMessage directMessage)
  - void onUserListMemberAddition(User addedMember, User listOwner, UserList list)
  - void onUserListMemberDeletion(User deletedMember, User listOwner, UserList list)
  - void onUserListSubscription(User subscriber, User listOwner, UserList list)
  - void onUserListUnsubscription(User subscriber, User listOwner, UserList list)
  - void onUserListCreation(User listOwner, UserList list)
  - void onUserListUpdate(User listOwner, UserList list)
  - void onUserListDeletion(User listOwner, UserList list)
  - void onUserProfileUpdate(User updatedUser)
  - void onBlock(User source, User blockedUser)
  - void onUnblock(User source, User unblockedUser)

Tidak ada penjelasan yang diberikan oleh Twitter4J

2.5.5 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 diotentifikasi
  - Status updateStatus(StatusUpdate latestStatus) throws TwitterException
    Melakukan update status oleh user yang telah diotentifikasi.
  - 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.StatusUpdat

2.5.6 OAuthSupport

- *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*

2.5.7 RequestToken

- *Constant*
  - public final class RequestToken extends OAuthToken implements java.io.Serializable
- *Constructor*
  - RequestToken(HttpResponse res, OAuthSupport oauth)
  - RequestToken(java.lang.String token, java.lang.String tokenSecret)

- RequestToken(java.lang.String token, java.lang.String tokenSecret, OAuthSupport oauth)

- *Methods*
  - public java.lang.String getAuthorizationURL()
  - public java.lang.String getAuthenticationURL()

2.5.8 AccessToken

- *Constant*
  - public class AccessToken extends OAuthToken implements java.io.Serializable
- *Constructor*
  - AccessToken(HttpResponse res)
  - AccessToken(java.lang.String token, java.lang.String tokenSecret)
  - AccessToken(java.lang.String token, java.lang.String tokenSecret, long userId)
- *Methods*
  - public java.lang.String getScreenName()
    Mengembalikan *screen name*
  - public long getUserId()
    Mengembalikan *user id*
  - public boolean equals(java.lang.Object o)
  - public int hashCode()
  - public java.lang.String toString()

2.5.9 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 diotentifikasi melakukan *retweet* terhadap suatu *tweet*, atau mengembalikan nilai *false* jika tidak.
- long getCurrentUserRetweetId()
  Mengembalikan *retweet id* sebuah *tweet* dari *user* yang telah diotentifikasi, 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.

2.5.10 TweetsResources

- *Constant*
  - public interface TweetsResources
- *Methods*
  - ResponseList<Status> getRetweets(long statusId) throws TwitterException
    Mengembalikan hingga dengan seratus *retweet* pertama
  - IDs getRetweeterIds(long statusId, long cursor) throws TwitterException
    Mengembalikan hingga dengan 100 *user ID* yang melakukan *retweet* terhadap *tweet* ditentukan dari *id parameter*
  - IDs getRetweeterIds(long statusId, int count, long cursor) throws TwitterException
    Mengembalikan hingga dengan *"count" user ID* yang melakukan *retweet* terhadap *tweet* ditentukan dari *id parameter*
  - Status showStatus(long id) throws TwitterException
    Mengembalikan *status* yang ditentukan dari parameter id.
  - Status destroyStatus(long statusId) throws TwitterException
    Menghapus *status* yang ditentukan dari parameter id.
  - Status updateStatus(java.lang.String status) throws TwitterException
    Melakukan *update status* terhadap *user* yang telah diotentifikasi.
  - Status updateStatus(StatusUpdate latestStatus) throws TwitterException
    Melakukan *update status* terhadap *user* yang telah diotentifikasi.
  - Status retweetStatus(long statusId) throws TwitterException
    Melakukan *retweet* terhadap sebuah *tweet*.
  - OEmbed getOEmbed(OEmbedRequest req) throws TwitterException
    Mengembalikan informasi yang mengizinkan terciptanya *embedded representation* dari tweet yang berada di *third party sites*
  - ResponseList<Status> lookup(long[] ids) throws TwitterException
    Mengembalikan objek *tweet* hingga dengan 100 *tweet* per **request**.
  - UploadedMedia uploadMedia(java.io.File mediaFile) throws TwitterException
    Melakukan *upload* gambar.

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 peneliti 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 akan menggunakan *Streaming* API, karena:

- Streaming API adalah *real-time* API, sedangkan Search API hanya dapat menangkap *tweet* setiap beberapa waktu sekali. Pada aplikasi yang akan dibuat skenarionya adalah pengguna akan menanyakan rute transportasi publik dalam bentuk *tweet* yang dikirimkan kepada user @kiriupdate, dalam skenario seperti ini dibutuhkanlah jawaban yang *real-time*.

- Menggunakan *Public Stream* dalam *endpoint streaming*. *Public Stream* mengambil semua data publik, sehingga semua *tweet* bisa ditangkap menggunakan *Public Stream*. Dalam pembuatan *Twitter Bot* untuk mencari jalur transportasi publik pungguna akan melakukan *mention tweet* kepada akun @kiriupdate untuk dapat memperoleh balasan *tweet* yang berisikan hasil pencarian jalur transportasi publik. Public Stream mempunyai fitur bernama *track*, fitur ini berguna untuk menyaring *tweet* berdasarkan *keyword* yang sudah di *track*. *Keyword* yang akan di *track* adalah @kiriupdate jadi program hanya menerima *tweet* yang di *mention* kepada akun @kiriupdate 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 kurang efisien karena *tweet update status* dan *direct message* tidak dibutuhkan. *Site stream* merupakan *multi-user stream*, dalam kasus Twitter Bot untuk mencari jalur transportasi publik ini akun yang dipakai untuk Twitter Bot hanya satu akun saja. Jadi penggunaan *site stream* dalam kasus ini kurang efisien.

- Menggunakan *User Stream* dalam *endpoint streaming*. *User Stream* mengandung hampir semua data yang berhubungan dengan satu user tertentu. Dalam pembuatan

Twitter Bot untuk mencari jalur transportasi publik pengguna hanya dapat melakukan *mention tweet* kepada user @kiriupdate untuk dapat memperoleh balasan *tweet* yang berisikan hasil pencarian jalur transportasi publik. Sedangkan *public stream* ini mengambil semua data publik, dalam kasus ini bisa saja menggunakan *public stream* tetapi tidak efisien. *Site stream* merupakan *multi-user stream*, dalam kasus Twitter Bot untuk mencari jalur transportasi publik ini akun yang dipakai untuk 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 mengotorisasi akun @kiriupdate, tetapi tidak perlu ada otentifikasi ke user karena Twitter Bot yang dibuat menggunakan otentifikasi langsung dari developer. *Application-only authentication* tidak bisa digunakan karena *application-only authentication* tidak bisa melakukan *posting tweet* dan tidak bisa melakukan koneksi dengan *streaming endpoint*. Sedangkan dalam kasus Twitter Bot untuk mencari jalur transportasi publik dibutuhkan otentifikasi yang dapat memposting *tweet* dan melakukan koneksi dengan *streaming endpoint*. Lalu untuk otentifikasi *PIN-based authorization* tidak cocok karena otentifikasi sudah dilakukan langsung dari developer tidak lagi meminta PIN untuk proses otentifikasi.

3.1.3 Analisis KIRI API

KIRI API menyediakan tiga layanan yang dapat digunakan, untuk aplikasi Twitter Bot akan 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 asal ke lokasi tujuan. Sedangkan *Search Place Web Service* berguna untuk menemukan rute perjalanan berdasarkan *latitute* dan *longitude* koordinat, layanan *Search Place Web Service* ini juga membantu untuk mengubah string teks untuk *latitude* dan *longitude*.

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.travelbukitjarian. 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

     Memberitahukan versi KIRI API, mengikuti versi yang paling baru 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 *Search Place 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

     Memberitahukan versi KIRI API, mengikuti versi yang paling baru 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 ke lokasi tujuan.

  3. locale : id

     locale berfungsi sebagai parameter untuk bahasa yang digunakan. Karena target dari perangkat lunak ini adalah orang Indonesia maka menggunakan parameter "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 : "mobile"

     Parameter *presentation* ini terdapat dua jenis yaitu "mobile" untuk perangkat bergerak dan "desktop" untuk komputer. Karena perangkat lunak ini dirancang untuk Twitter Bot yang kebanyakan penggunanya menggunakan perangkat bergerak maka parameter dari *presentation* yang cocok adalah "mobile".

  7. apikey : 889C2C8FBB82C7E6

     Merupakan password yang digunakan untuk mengakses KIRI API.

  Penulis mencoba mencari langkah perjalanan dari pvj menuju bip. 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=mobile&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 \%fromicon to Jalan
                            Sukajadi \%toicon.",
                        null
                    ],
                    [
                        "angkot",
                        "kalapakarangsetra",
                        ["-6.88958,107.59691","-6.89052,107.59696","-6.89146,107.59701","-6.89239,107.59706","

                        "Take angkot Kalapa - Karang Setra at Jalan Sukajadi \%fromicon, and
                            alight at Jalan Pajajaran \%toicon about 2.6 kilometer later
                            .",
                        null
                    ],
                    [
                        "angkot",
                        "ciroyomantapani",
                        ["-6.90713,107.60441","-6.90713,107.60441","-6.90679,107.60440","-6.90563,107.60438","

                        "Take angkot Ciroyom - Antapani at Jalan Pajajaran \%fromicon, and
                            alight at Jalan Aceh \%toicon about 1.7 kilometer later.",
                        null
                    ],
                    [
                        "walk",
                        "walk",
                        ["-6.90974,107.61091","-6.90864,107.61108"],
                        "Walk about 124 meter from Jalan Aceh \%fromicon to your destination
                            \%toicon.",
                        null
                    ]
                    ],
                        "traveltime":"25 minutes"
                    }
                ]
    }
```

3.1.4 Analisis Twitter4J

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

- *TwitterStream*
- *UserStreamListener*
- *TwitterFactory*
- *RequestToken*
- *Status*

Untuk menggunakan Twitter4J diperlukan *properties* untuk proses konfigurasi. Konfigurasi dapat dilakukan dengan cara membuat *file* twitter4j.properties , kelas *ConfigurationBuilder*, dan *System Property*. Ketiganya dapat digunakan untuk melakukan konfigurasi Twitter4J, tetapi penulis menggunakan *file* twitter4j.properties karena lebih praktis dalam pemakaiannya. 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 3.3: isi dari twitter4j.properties

```
    {
        debug=true
        oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIUl
        oauth.consumerSecret=YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****
```

```
        oauth.accessToken=313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
        oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptc****
    }
```

2. via *ConfigurationBuilder*

   Menggunakan *ConfigurationBuilder class* untuk melakukan konfigurasi Twitter4J.

Listing 3.4: isi dari twitter4j.properties

```
    {
        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true)
            .setOAuthConsumerKey("3iT8duMItTTrdaU1qTHxwDIU1")
            .setOAuthConsumerSecret("YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****")
            .setOAuthAccessToken("313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ")
            .setOAuthAccessTokenSecret("LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptc****");
        TwitterFactory tf = new TwitterFactory(cb.build());
        Twitter twitter = tf.getInstance();
    }
```

3. via *System Properties*

   Menggunakan *System Properties* untuk melakukan konfigurasi Twitter4J.

Listing 3.5: isi dari twitter4j.properties

```
        $ export twitter4j.debug=true
        $ export twitter4j.oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIU1
        $ export twitter4j.oauth.consumerSecret=
            YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****
        $ export twitter4j.oauth.accessToken=313287708-
            NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
        $ export twitter4j.oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptc
            ****
        $ java -cp twitter4j-core-4.0.2.jar:yourApp.jar yourpackage.Main
```

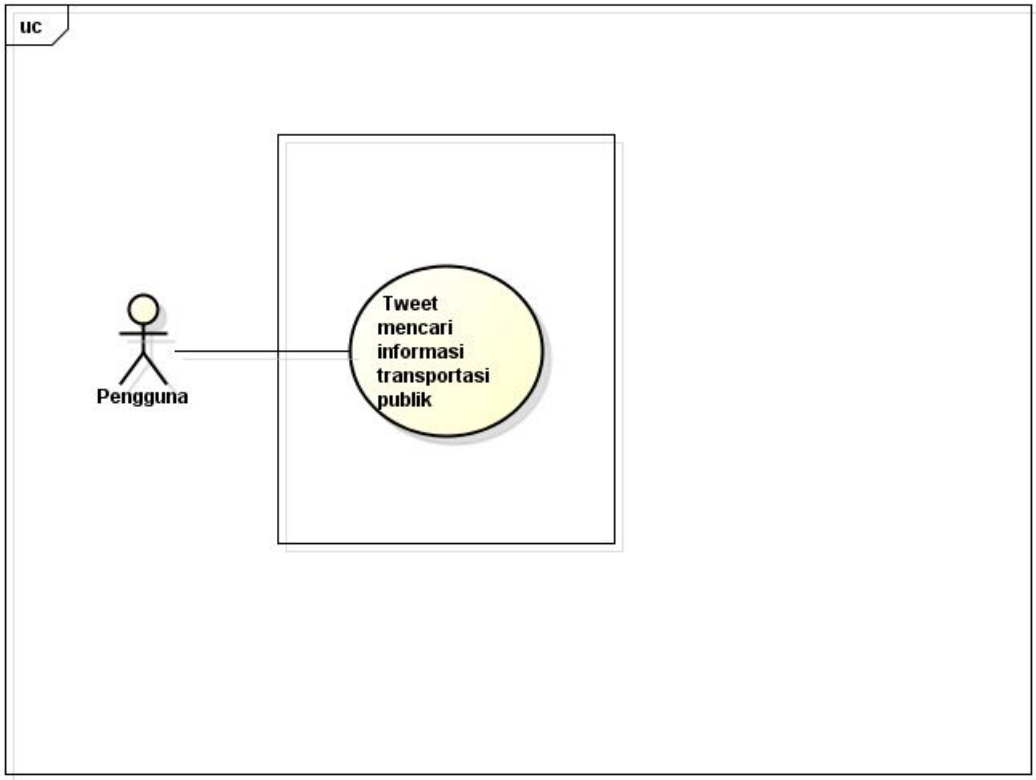
3.2 Analisis Perangkat Lunak

Perangkat lunak yang akan dibangun adalah Twitter Bot untuk mencari jalur transportasi publik. Perangkat lunak yang dibuat merupakan sebuah Twitter Bot yang berguna untuk membalas *tweet* secara *real-time* kepada *user* untuk memberitahukan jalur-jalur yang harus ditempuh menggunakan transportasi publik. Aplikasi yang digunakan untuk membangun Twitter Bot Untuk Mencari Jalur Transportasi Publik adalah NetBeans IDE 8.0. 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 menerima dan membaca *tweet* yang di *mention* kepada user @kiriupdate
2. Dapat melakukan proses pencarian jalur transportasi publik
3. Dapat membalas *tweet* dengan memberikan hasil pencarian jalur transportasi publik dengan format yang sudah ditentukan



Gambar 3.1: Use case Twitter Bot

3.2.2 *Use Case Diagram*

Use case Diagram pada perangkat lunak yang akan dibangun ini mengandung satu aktor, yaitu pengguna. *Use case diagram* dapat dilihat pada gambar.

Skenario *Use Case* Skenario ini hanya memiliki satu aktor yaitu pengguna. *Tweet* mencari informasi transportasi publik pada skenario ini dilakukan dengan melakukan *tweet* kepada user @kiriupdate berisikan format yang sesuai untuk pencarian rute 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 @kiriupdate
Skenario Utama	Pengguna melakukan *Tweet* kepada *user* @kiriupdate dengan format yang sudah ditentukan
Eksepsi	Format penulisan salah

Tabel 3.1: Skenario *Tweet* mencari informasi transportasi

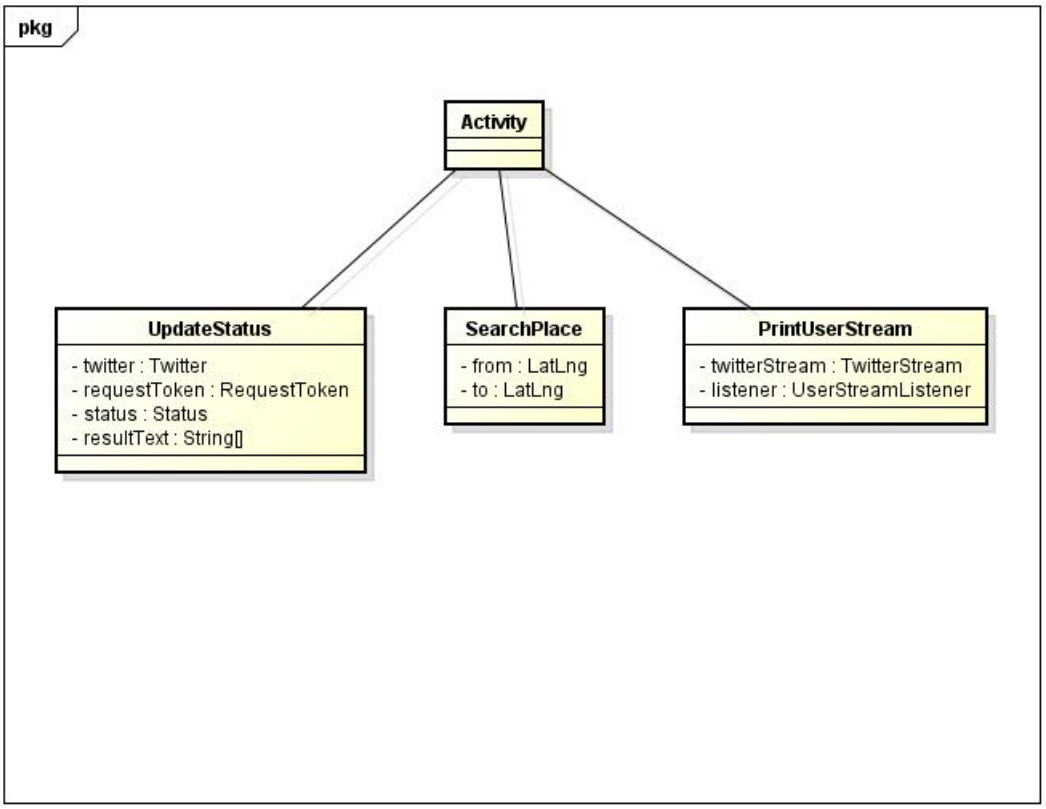
3.2.3 *Class Diagram*

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

1. Perangkat lunak akan berjalan terus untuk menjalankan Twitter Bot.

2. Pengguna melakukan *Tweet* mencari informasi transportasi dengan cara melakukan *mention* kepada *user* @kiriupdate dengan format yang sesuai dengan ketentuan.
3. Perangkat lunak menerima mention dari pengguna.
4. Perangkat lunak akan mencari jalur transportasi umum.
5. Melalukan *reply* kepada pengguna berupa jalur transportasi publik yang harus ditempuh.

Berikut adalah *class diagram* sederhana:



Gambar 3.2: *Class Diagram* Twitter Bot

BAB 4

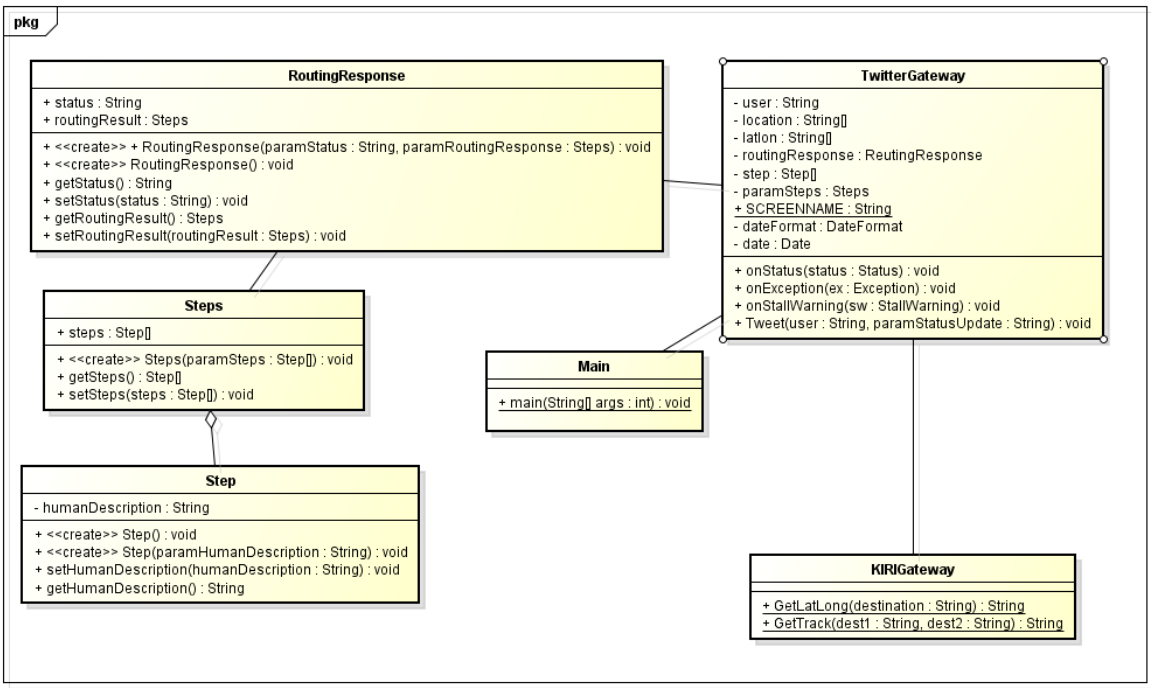
PERANCANGAN PERANGKAT LUNAK

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 Perangkat Lunak

4.1.1 Perancangan Kelas

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



Gambar 4.1: Class Diagram Pembuatan Twitter Bot untuk Mencari Jalur Transportasi Publik

- Kelas Main, merupakan kelas yang berfungsi untuk membuat koneksi dengan Twitter ketika program 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 user.
    * 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 ??/ variabel?? 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 ??/ variabel?? yang berguna untuk menampung semua step.

  – Method

    * public void onStatus(Status status), merupakan *method* yang menangkap *tweet* dan memproses *tweet* tersebut. Ketika ada *tweet* yang masuk, *tweet* tersebut akan diolah isinya. Jika *tweet* yang diterima merupakan *tweet* untuk mencari jalur transportasi publik maka *tweet* tersebut akan dimasukan ke atribut yang sudah disediakan yaitu *user*, lokasi awal dan lokasi tujuan. Setelah mendapatkan lokasi awal dan lokasi tujuan barulah proses pencarian dimulai dengan menggunakan *method GetLatLong* dan *method GetTrack* yang terdapat di kelas KIRIGateway. Hasil pencarian akan dimasukan ke dalam atribut *routingResponse*, *step*, dan *steps*. Setelah itu akan dilakukan pemanggilan *method Tweet* untuk melakukan proses balasan.
    * public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice),
    * public void onTrackLimitationNotice(int numberOfLimitedStatuses),
    * public void onScrubGeo(long userId, long upToStatusId),
    * public void onException(Exception ex), merupakan method yang berguna untuk menangkap *exception*.
    * public void onStallWarning(StallWarning sw),
    * public void Tweet(String user, String paramStatusUpdate), merupakan method untuk melakukan *tweet* balasan atau *reply* yang ditujukan kepada *user* tertentu. Twitter sendiri hanya dapat melakukan tweet dengan batas 140 karakter, oleh karena itu method ini akan mengatasi keterbatasan tweet tersebut dengan cara ??memecah-mecah?? tweet. Method ini juga akan memberi waktu yang sesuai dengan server di setiap akhir tweet, hal ini bertujuan untuk menghindari adanya duplikat 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, digunakan untuk menyimpan apakah status dari hasil pencarian.
    - routingResult, 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.

4.1.2 Sequence Diagram

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

Pertama, program akan melakukan *streaming* pada saat kelas *main* dijalankan. Kelas *main* akan membuka gerbang untuk mengakses *Twitter API*, dengan menggunakan *Streaming* API aplikasi akan menangkap semua *tweet* yang memiliki kata kunci @kviniink. Aplikasi akan terus melakukan *streaming tweet* hingga aplikasi dinon-aktifkan.

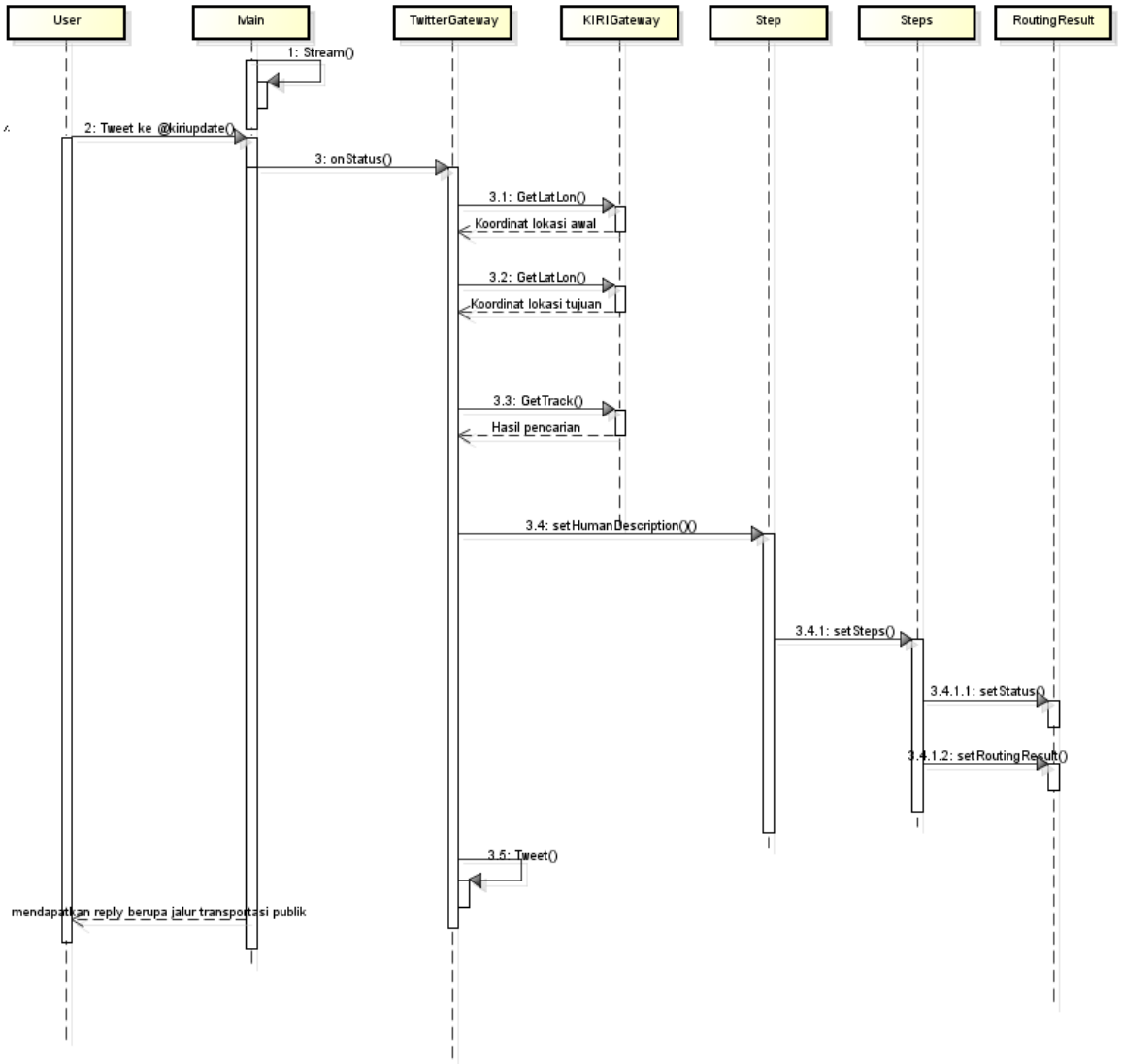
Kelas TwitterGateway akan memproses *tweet* ketika terdapat *tweet* yang dirujuk (*mention*) kepada @kviniink. *Method onStatus* akan melakukan pengecekan apakah *tweet* tersebut merupakan *tweet* untuk mencari jalur transportasi publik atau bukan. Jika benar maka nama *user* pengirim, alamat dari lokasi awal dan lokasi tujuan akan disimpan lalu akan dicari koordinat dari masing-masing lokasi menggunakan KIRI API. Proses mencari koordinat ini dilakukan oleh kelas KIRIGateway.

Kelas KIRIGateway akan memanggil *method GetLatLon* untuk mencari koordinat suatu lokasi. Setelah didapatkan koordinat dari masing-masing lokasi maka kelas TwitterGateway akan mengolahnya terlebih dahulu dikarenakan hasil dari *method GetLatLon* ini berupa JSON. Setelah itu maka hasilnya akan dikembalikan kepada kelas KIRIGateway untuk dicari jalur transportasi publik dari lokasi awal menuju lokasi tujuan menggunakan *method GetTrack*. Hasil dari *method GetTrack* akan disimpan pada atribut *step, steps*, dan *routingResult*.

Setelah selesai, langkah-langkah jalur transportasi publik siap di *reply*. Proses *reply* dilakukan oleh *method tweet* 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. Aplikasi akan terus melakukan proses tersebut hingga aplikasi dinon-aktifkan.

4.1.3 Perancangan Antar Muka

Aplikasi yang akan disusun ini memiliki antarmuka berbasis teks. Oleh karena itu, interaksi dengan pengguna dapat dilakukan melalui website atau aplikasi Twitter. Gambar 5.1 adalah

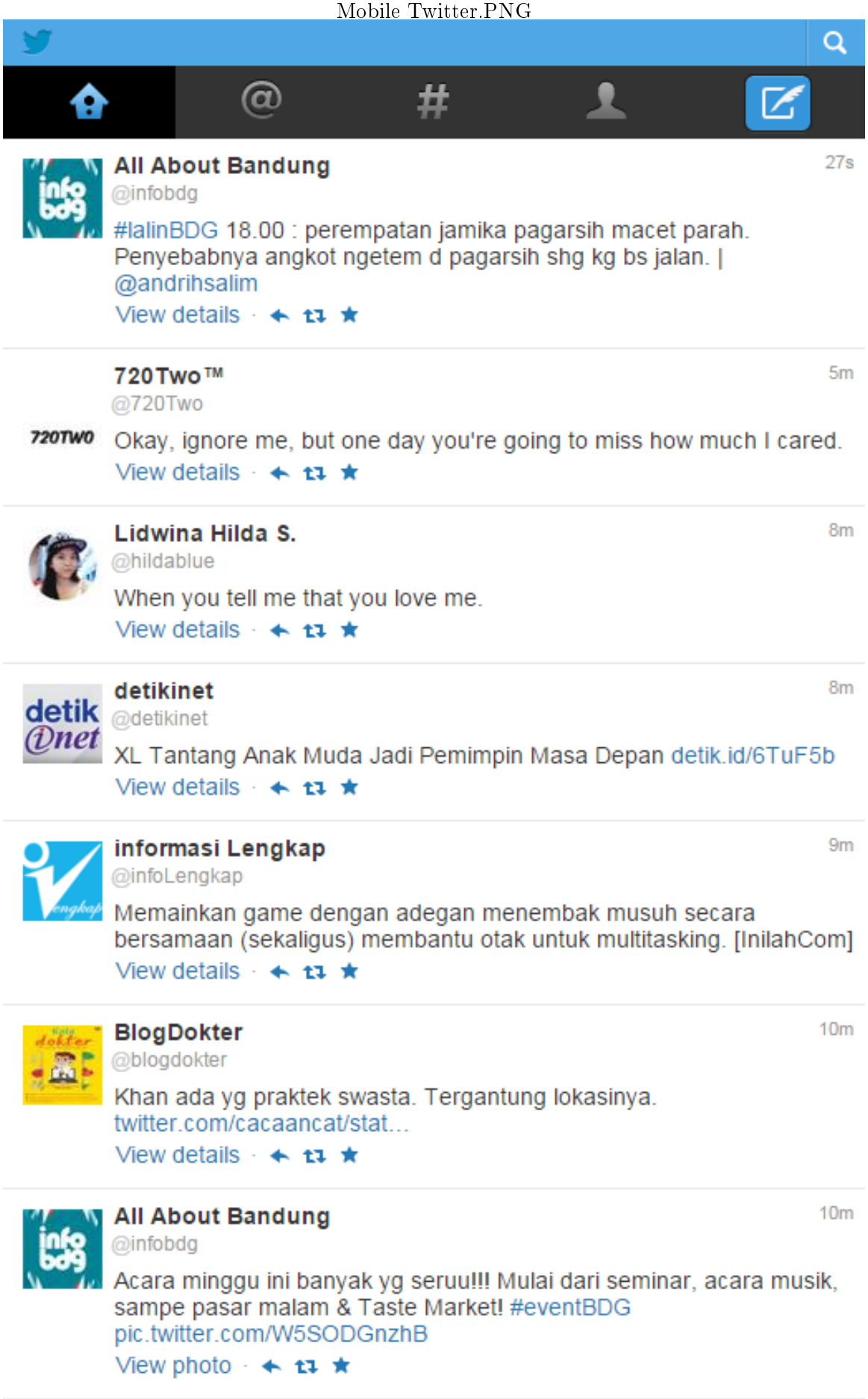


Gambar 4.2: Sequence Diagram Pembuatan Twitter Bot untuk Mencari Jalur Transportasi Publik

tampilan antar muka dari Twitter yang diakses melalui mobile. Dari situlah, pengguna dapat melakukan tweet untuk mencari jalur transportasi publik. Tweet dilakukan dengan cara menuliskan tweet kepada user @kviniink.

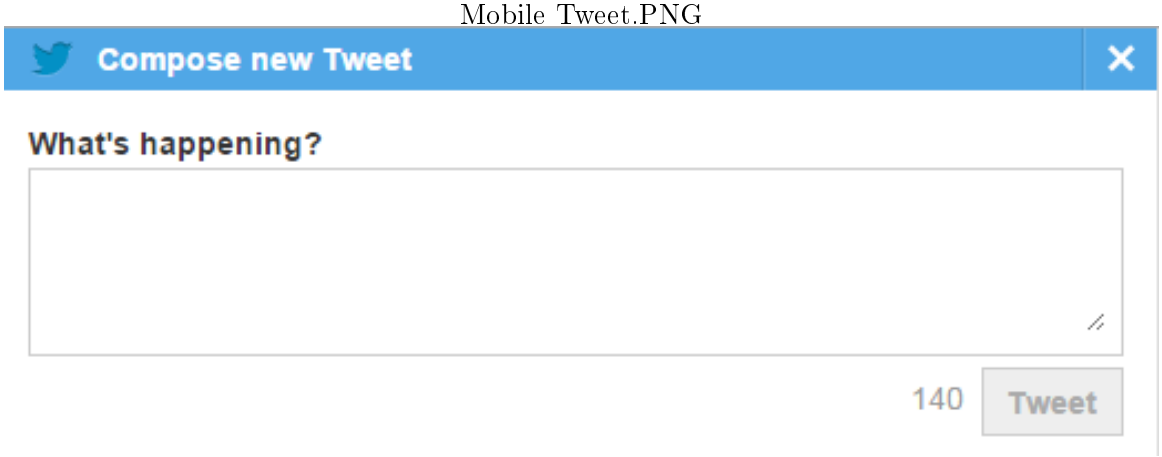
Gambar 5.2 menunjukan menu untuk melakukan tweet. Pertama, format yang dimasukan harus benar. Format dari penulisan tweet adalah "'lokasi awal"' to "'lokasi tujuan"'. Ketika user melakukan tweet maka Twitter Bot untuk mencari jalur transportasi publik akan menangkap tweet tersebut dan akan memproses tweet tersebut, rancangan dari hasil tangkapan tweet dapat dilihat pada gambar ??. Sedangkan rancangan dari hasil pencarian dapat dilihat pada gambar ??.

Mobile Twitter.PNG



Gambar 4.3: Homepage Twitter versi mobile

Mobile Tweet.PNG



Gambar 4.4: Tampilan untuk melakukan tweet

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

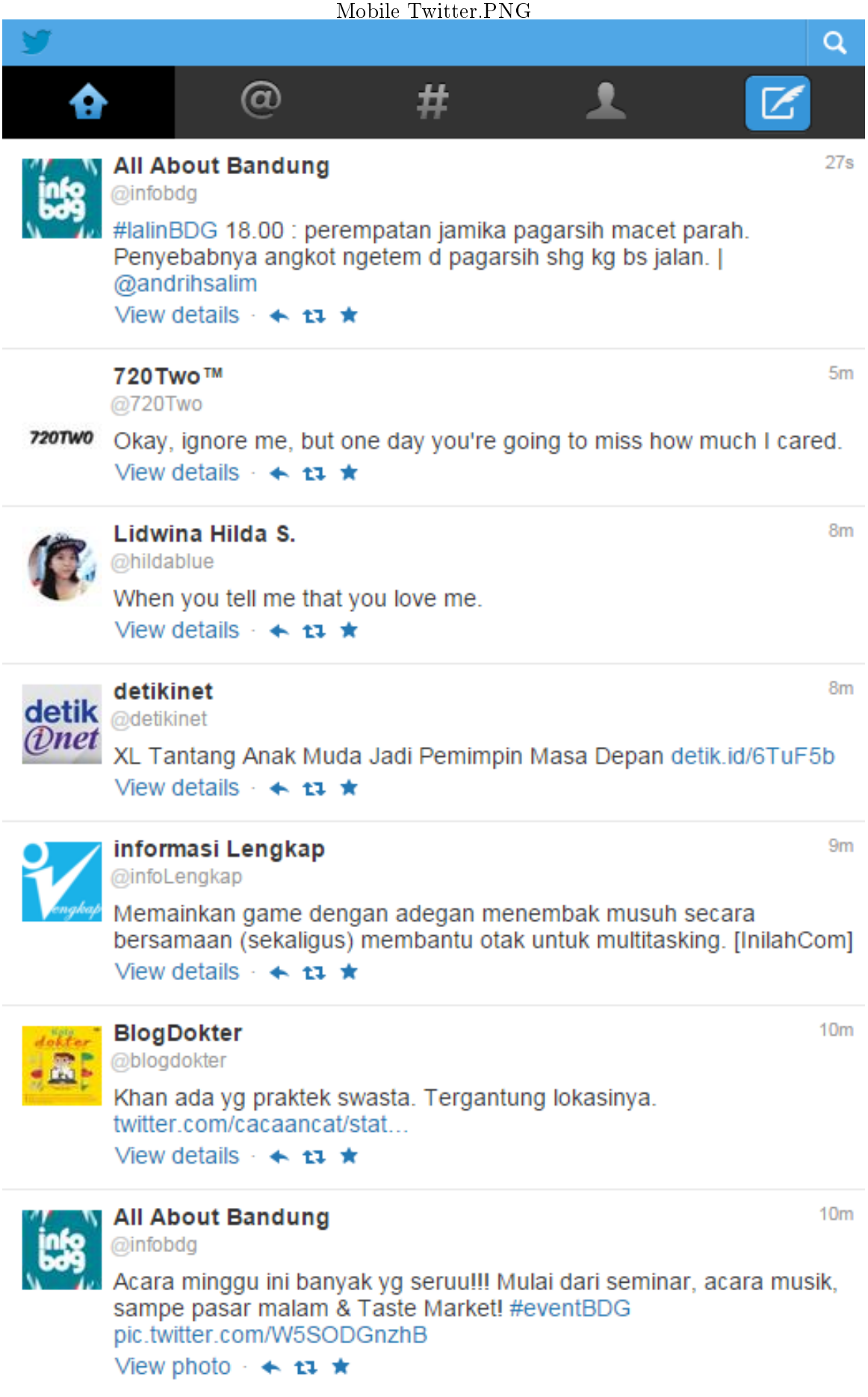
5.2 Hasil Tampilan Antarmuka

Pada aplikasi pembuatan *Twitter Bot* untuk mencari jalur transportasi publik ini memiliki tampilan antarmuka berbasis teks yang berguna untuk melihat hasil penangkapan tweet, dan hasil tweet yang diberikan kepada user. Sedangkan user dapat mencoba aplikasi ini secara langsung menggunakan Twitter, baik menggunakan website Twitter ataupun aplikasi Twitter.

Tampilan *home page* Twitter dapat dilihat pada gambar 5.1. Disini peneliti menggunakan website Twitter versi mobile agar lebih mudah dilihat karena tampilan website Twitter versi mobile lebih sederhana dibandingkan website Twitter versi desktop. Setelah itu user akan menekan tombol tweet pada pojok kanan atas dan akan memberikan tampilan seperti pada gambar 5.2. Dari situ user dapat melakukan tweet kepada Twitter Bot untuk mencari jalur transportasi publik.

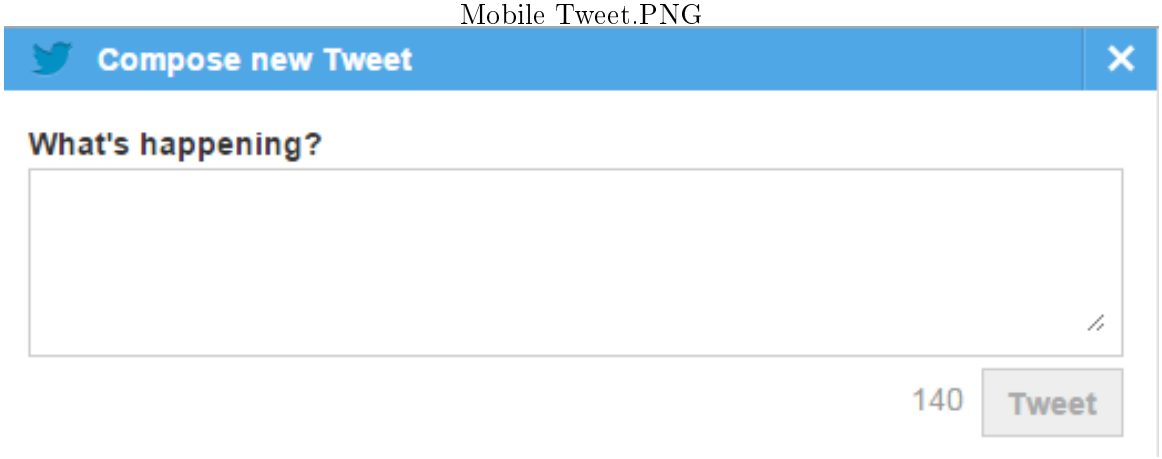
Setelah ada *mention* yang ditujukan kepada Twitter Bot, aplikasi akan menangkap *tweet* tersebut dan ditampilkan dalam bentuk pesan *tweet* yang diterima oleh aplikasi. Hasil *tweet* yang diterima aplikasi dapat dilihat pada gambar 5.3. Setelah itu *tweet* akan diperiksa

Mobile Twitter.PNG

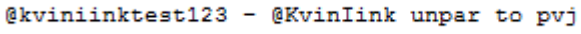


Gambar 5.1: Homepage Twitter versi mobile

Mobile Tweet.PNG



Gambar 5.2: Tampilan untuk melakukan tweet



Gambar 5.3: Hasil streaming tweet

oleh aplikasi apakah *tweet* tersebut bertujuan untuk mencari jalur transportasi publik atau tidak. Jika benar, maka aplikasi akan melakukan proses pencarian jalur transportasi publik dan melakukan *reply* atau balasan kepada pengguna. *Reply tweet* tersebut berisikan jalur transportasi publik yang harus ditempuh kepada *user*. Selain melakukan *reply*, aplikasi juga menampilkan *tweet* tersebut yang dapat dilihat pada gambar 5.4.

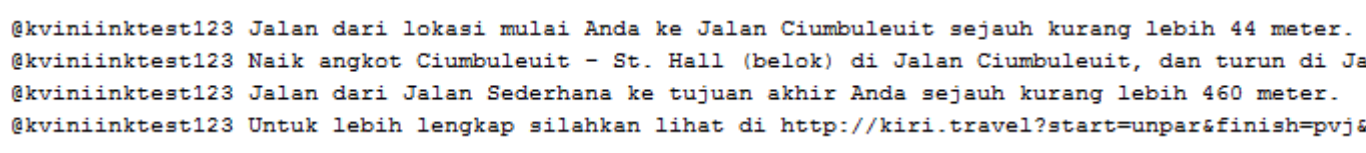
5.3 Pengujian

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

Untuk pengujian Twitter Bot, penulis menggunakan *user* @kviniink. Lalu untuk penguji, penulis menggunakan user @kviniinktest123.

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.



Gambar 5.4: Hasil balasan tweet kepada user

No	Pengujian	Reaksi yang Diharapkan	Reaksi Aplikasi
1	Melakukan otentikasi menggunakan *3-legged authorization*	Membuka gerbang agar Twitter Bot dapat melakukan *streaming tweet* dan membalas *tweet*	sesuai
2	Melakukan *streaming tweet*	Menangkap semua *tweet* yang dimention kepada *user* @kviniink	sesuai
3	Membaca *tweet* yang ditangkap	Melakukan pengecekan tweet apakah tweet tersebut untuk mencari transportasi publik atau bukan	sesuai
4	Melakukan pencarian koordinat lokasi menggunakan KIRI API	Mendapatkan hasil koordinat *latitude* dan *longitude* dari lokasi yang dicari	sesuai
5	Melakukan pencarian jalur transportasi publik menggunakan KIRI API	Mendapatkan jalur-jalur yang harus ditempuh dari lokasi awal menuju lokasi tujuan	sesuai
5	Melakukan tweet balasan	Membalas tweet dengan memberikan hasil pencarian jalur transportasi publik dengan format yang sudah ditentukan	sesuai

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

5.3.2 Pengujian Eksperimental

Pada subbab ini akan dilakukan pengujian *Twitter Bot* untuk mencari jalur transportasi publik selama 12 jam.

DAFTAR REFERENSI

[1] Twitter *Twitter Documentation* 2014 : https://dev.twitter.com/overview/documentation.

[2] Tim O'Reilly *The Twitter Book* 2009: O'Reilly Media, Inc

[3] Kiri Team *KIRI API v2 Documentation* 2014 : https://bitbucket.org/projectkiri/kiri_api/wiki/KIRI%20API%20v2%20Documentation

[4] Twitter4J *Twitter4J Documentation* 2007 : http://twitter4j.org/javadoc/index.html

[5] OAuth *Hueniverse Documentation* 2010 : http://hueniverse.com/oauth/guide/intro/