

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 <i>Search API</i>	6
2.2.2 <i>Streaming API</i>	9
2.3 OAuth	12
2.3.1 <i>Application-only authentication</i>	13
2.3.2 <i>3-legged authorization</i>	14
2.3.3 <i>PIN-based authorization</i>	14
2.4 KIRI API	15
2.4.1 <i>Routing Web Service</i>	15
2.4.2 <i>Search Place Web Service</i>	17
2.4.3 <i>Nearest Transports Web Service</i>	17
2.5 Twitter4J	18
2.5.1 <i>TwitterFactory</i>	19
2.5.2 <i>TwitterStream</i>	19
2.5.3 <i>TwitterStreamFactory</i>	20
2.5.4 <i>UserStreamListener</i>	21
2.5.5 <i>TweetsResources</i>	22
2.5.6 <i>OAuthSupport</i>	22
2.5.7 <i>RequestToken</i>	23
2.5.8 <i>AccessToken</i>	24
2.5.9 <i>Status</i>	24
2.5.10 <i>TweetsResources</i>	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	<i>Use Case Diagram</i>	33
3.2.3	<i>Class Diagram</i>	34
4	PERANCANGAN PERANGKAT LUNAK	35
4.1	Perancangan Perangkat Lunak	35
4.1.1	Perancangan Kelas	35
4.1.2	Sequence Diagram	38
5	IMPLEMENTASI DAN PENGUJIAN APLIKASI	41
5.1	Lingkungan Pembangunan	41
5.2	Pengujian	41
5.2.1	Pengujian Fungsional	42
5.2.2	Pengujian Eksperimental	42
	DAFTAR REFERENSI	47

DAFTAR GAMBAR

2.1	Ilustrasi sign in	14
2.2	Contoh PIN-based authorization	15
3.1	Use case Twitter Bot	33
3.2	<i>Class Diagram</i> Twitter Bot	34
4.1	Class Diagram Pembuatan Twitter Bot untuk Mencari Jalur Transportasi Publik . .	35
4.2	Sequence Diagram Twitter Bot untuk Mencari Jalur Transportasi Publik	39
5.1	Tweet dari BIP menuju IP	43
5.2	Hasil Pencarian Rute Transportasi Publik dari BIP menuju IP	43
5.3	Tweet dari BIP menuju PVJ	44
5.4	Hasil Pencarian Rute Transportasi Publik dari BIP menuju PVJ	44
5.5	Hasil Pencarian Jalur Transportasi Publik dari BIP menuju IP Melalui Website KIRI	45
5.6	Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI	46

DAFTAR TABEL

2.1	Contoh berbagai macam pencarian <i>tweet</i>	7
2.2	Contoh <i>mapping</i> dari <i>search query</i> ke <i>query</i> pengkodean URL	8
2.3	Parameter POST <i>statuses/filter</i>	10
2.4	Parameter GET <i>statuses/sample</i>	10
2.5	Parameter GET <i>statuses/firehose</i>	10
2.6	Parameter GET <i>user</i>	11
2.7	Parameter <i>Routing Web Service</i>	16
2.8	Tabel parameter <i>Search Place Web Service</i>	17
2.9	Tabel parameter <i>Nearest Transports Web Service</i>	18
3.1	Skenario <i>Tweet</i> mencari informasi transportasi	33
5.1	Tabel Hasil pengujian fungsionalitas pada Aplikasi Twitter Bot untuk mencari jalur transportasi publik	42

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 remaja¹. 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 akun². *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 terjadi 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 kendaraan⁴. 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

¹Kominfo 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

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

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

⁴Oris 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

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

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

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

26 1.2 Rumusan Masalah

27 Mengacu kepada deskripsi yang diberikan, maka rumusan masalah pada penelitian ini ada-
28 lah:

- 29 • Bagaimana membuat *Twitter bot* untuk mencari jalur transportasi publik?
- 30 • Bagaimana membuat *Twitter bot* untuk dapat merespon secara *real time*?
- 31 • Bagaimana memformat petunjuk rute perjalanan dalam keterbatasan tweet 140 ka-
32 rakter?

33 1.3 Tujuan

34 Tujuan dari penelitian ini adalah:

- 35 • Membuat aplikasi *Twitter bot* untuk mencari jalur transportasi publik.
- 36 • Membuat aplikasi Twitter yang bekerja secara *real time*.
- 37 • Membuat algoritma untuk memecah instruksi dari KIRI API dan mengubahnya ke
38 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 akun¹. 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 menghabiskan 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

¹Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 7

²Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 9

yang terjadi 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 relevansi, bukan terhadap kelengkapan data. Ini berarti bahwa beberapa *Tweet* dan pengguna akan hilang dari hasil pencarian.

1 **Bagaimana cara membuat sebuah *query*** Cara terbaik dalam membuat sebuah *qu-*
 2 *ery*, melakukan percobaan yang valid dan mengembalikan tweet yang sesuai adalah dengan
 3 mencobanya di twitter.com/search. URL yang ditampilkan pada browser akan berisi sin-
 4 taks *query* yang sesuai agar dapat digunakan kembali pada API *endpoint*. Berikut adalah
 5 contohnya:

- 6 1. Melakukan pencarian untuk *tweet* yang direferensikan kepada akun @twitterapi. Per-
 7 tama kita harus melakukan pencarian pada twitter.com/search.
- 8 2. Lakukan pengecekan dan salin URL yang ditampilkan. Sebagai contoh didapatkan
 9 URL seperti ini, <https://twitter.com/search?q=%40twitterapi>.
- 10 3. Ganti <https://twitter.com/search> dengan [https://api.twitter.com/1.1/search/](https://api.twitter.com/1.1/search/tweets.json)
 11 [tweets.json](https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi) dan akan didapatkan [https://api.twitter.com/1.1/search/tweets.](https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi)
 12 [json?q=%40twitterapi](https://api.twitter.com/1.1/search/tweets.json?q=%40twitterapi)
- 13 4. Eksekusi URL tersebut untuk melakukan pencarian di dalam API.

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

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

Tabel 2.1: Contoh berbagai macam pencarian *tweet*

18 Dipastikan bahwa pengkodean URL terhadap *query* dilakukan terlebih dahulu sebelum
 19 melakukan *request*. Tabel 2.2 memberikan contoh *mapping* dari *search query* ke *query* peng-
 20 kodean URL.

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

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

1 **Additional parameters** Terdapat parameter tambahan yang dipergunakan untuk hasil
2 pencarian yang lebih baik. Berikut adalah penjelasan dari parameter tambahan tersebut :

- 3 • **Result Type.** Seperti hasil yang terdapat pada twitter.com/search, parameter
4 *result_type* memungkinkan hasil pencarian akan berdasarkan *tweet* yang paling baru
5 atau *tweet* yang paling populer atau bahkan gabungan dari keduanya.
- 6 • **Geolocatization.** Pencarian tempat tidak tersedia pada API, tetapi ada beberapa
7 cara yang tepat untuk membatasi *query* dengan cara menggunakan parameter geo-
8 code lalu menentukan "*latitude, longitude, radius*". Contohnya adalah "37.781157,-
9 122.398720,1mi". Ketika pencarian lokasi pencarian API pertama akan mencoba me-
10 nemukan *tweet* yang memiliki *latitude* yang sudah dimasukan kedalam *query* geoco-
11 de, jika tidak berhasil maka API akan mencoba menemukan *tweet* yang dibuat oleh
12 pengguna yang lokasi profilnya terdapat pada *latitude* tersebut. Artinya adalah ha-
13 sil pencarian mungkin menerima *tweet* yang tidak mencakup informasi *latidute* atau
14 *longitude*.
- 15 • **Language.** Bahasa dapat dijadikan parameter untuk mencari tweet yang sesuai de-
16 ngan bahasa tersebut.
- 17 • **Iterating in a result set.** Parameter seperti *count, until, since_id, max_id* me-
18 mungkinkan untuk mengkontrol bagaimana iterasi melalui hasil pencarian.

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

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

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

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

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

1 Ketika anda ingin mencari *tweet* yang sedang populer dari spesifik *user* dan *tweet*
2 tersebut terdapat sebuah hashtag tertentu:

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

5 – Maka *search* URL akan seperti ini: [https://api.twitter.com/1.1/search/
6 tweets.json?q=from%3Akvinink%20%23nasa&result_type=popular](https://api.twitter.com/1.1/search/tweets.json?q=from%3Akvinink%20%23nasa&result_type=popular)

7 2.2.2 Streaming API

8 *Streaming* API adalah contoh *real-time* API. API ini ditujukan bagi para pengembang de-
9 ngan kebutuhan data yang intensif. Contohnya jika mencari cara untuk membangun sebuah
10 data produk *data-mining* atau tertarik dalam analisis penelitian. *Streaming* API memung-
11 kinkan melacak kata kunci yang ditentukan dalam jumlah besar dan melakukan suatu aksi
12 (seperti *tweet*) secara langsung atau *real-time*.

13 Twitter menawarkan beberapa *endpoint streaming*, disesuaikan dengan kasus yang terja-
14 di.

15 • *Public stream*

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

19 • *User Stream*

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

22 • *Site Stream*

23 Versi dari *multi-user stream*. *Site stream* harus terhubung dengan server yang terko-
24 neksi dengan Twitter atas nama banyak pengguna.

25 ***Public Streams*** *Stream* ini menawarkan sampel data publik yang mengalir melalui Twit-
26 ter. Ketika aplikasi membuat sambungan ke *streaming endpoint*, aplikasi akan menyampa-
27 ikan umpan Tweet tanpa perlu khawatir akan keterbatasan *rate limit*.

28 *Endpoints*

29 • POST statuses / *filter*

30 • GET statuses / *sample*

31 • GET statuses / *firehose*

32 **POST statuses/*filter*** POST *filter* ini mengembalikan status publik yang sesuai dengan
33 satu atau lebih predikat yang telah di filter. *Multiple parameter* memungkinkan klien untuk
34 menggunakan koneksi tunggal untuk ke *Streaming* API. Antara GET dan POST *request* ke-
35 duanya 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

<i>follow</i>	Tanda koma memisahkan list user ID, hal ini menunjukkan penggunaan untuk kembali ke status untuk stream.
<i>track</i>	Kata pencarian untuk track. Fase kata kunci dipisahkan oleh tanda koma.
<i>locations</i>	Menentukan lokasi yang dilacak.
<i>delimited</i>	Menentukan apakah pesan harus dibatasi limitnya.
<i>stall_warnings</i>	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

<i>delimited</i>	Menentukan apakah pesan harus dibatasi limitnya.
<i>stall_warning</i>	Menentukan apakah pesan warning harus dikirim atau tidak.

Tabel 2.4: Parameter GET *statuses/sample*

GET *statuses/firehose* Mengembalikan semua status publik. Beberapa aplikasi membutuhkan 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

<i>count</i>	Kumpulan pesan untuk dijadikan bahan materi
<i>delimited</i>	Menentukan apakah pesan harus dibatasi limitnya.
<i>stall_warning</i>	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 terhubung 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 spesifik 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*

<i>delimited</i>	Menentukan apakah pesan harus dibatasi limitnya.
<i>stall_warnings</i>	Menentukan apakah pesan warning harus dikirim atau tidak.
<i>with</i>	Menentukan apakah pesan informasi harus dikembalikan untuk user yang sudah diotentifikasi atau dikirim juga kepada akun yang difollow oleh akun yang sudah diotentifikasi tersebut.
<i>replies</i>	Menentukan apakah harus mengembalikan @replies.
<i>follow</i>	Termasuk tweet public tambahan dari daftar yang disediakan ID pengguna.
<i>track</i>	Termasuk tweet tambahan yang cocok dengan kata kunci tertentu.
<i>locations</i>	Termasuk tweet tambahan yang termasuk dalam batasan lokasi tertentu.
<i>stringify_friend_ids</i>	Mengirim list teman yang terdiri 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 dihentikan secara otomatis. *User* login dari beberapa instansi dari aplikasi OAuth yang sama akan mengalami siklus koneksi yaitu akan dihubungkan 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

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

4 2.3 OAuth

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

10 OAuth menyediakan suatu metode bagi pengguna untuk memberi akses pihak ketiga
 11 untuk *resources* (sumber daya) mereka tanpa berbagi password mereka. Cara ini juga
 12 memberikan cara untuk memberikan akses yang terbatas(dalam satu lingkup atau dura-
 13 si). Sebagai contoh, seorang pengguna web dapat memberikan layanan percetakan(*client*)
 14 untuk mengakses foto pribadinya yang disimpan di layanan berbagi foto(server) tanpa harus
 15 memberikan *username* dan *passwordnya*. Ia akan mengotentikasi langsung dengan layanan
 16 berbagi foto tersebut yang mengeluarkan layanan percetakan.

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

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

26 Twitter OAuth yang diberikan memiliki fitur

27 • *Secure*

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

30 • *Standard*

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

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

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

39 *Application-only authentication* *Application-only authentication* adalah bentuk dari oten-
 40 tifikasi dimana aplikasi anda membuat *API request* atas nama aplikasi itu sendiri tanpa

1 adanya konteks dari pengguna. Pemanggilan API masih terbatas dalam setiap *API method*
2 .

3 **2.3.1 Application-only authentication**

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

- 9 • Melihat *timeline*
- 10 • Mengakses *following* dan *follower* dari suatu *akun*
- 11 • Mencari dalam *tweet*
- 12 • mengambil informasi dari *user* manapun

13 Tetapi *application-only authentication* tidak bisa melakukan :

- 14 • Posting *tweet*
- 15 • Melakukan koneksi dengan *Streaming endpoint*
- 16 • Mencari *user* seseorang
- 17 • Menggunakan geo endpoint
- 18 • Mengakses DM

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

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

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

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

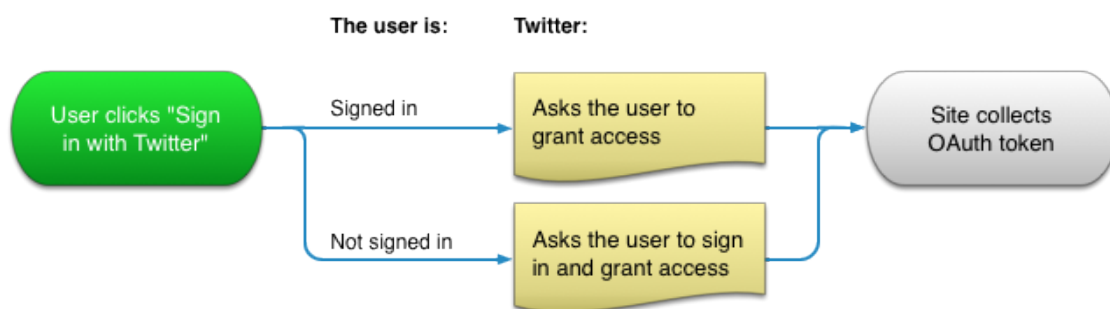
1 *Rate limiting.* Request yang dibuat atas nama pengguna tidak akan menguras ketersedi-
 2 aan *rate limit* dan *request* tidak akan menguras batas penggunaan **limit** dalam *user-based*
 3 *auth*.

4 2.3.2 3-legged authorization

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

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

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



Gambar 2.1: Ilustrasi sign in

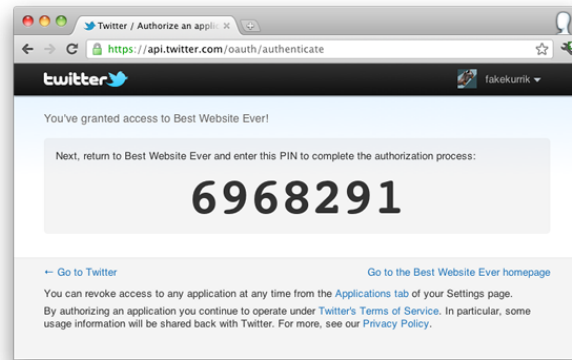
14 2.3.3 PIN-based authorization

15 cara kerja dari *PIN-based authorization* ini ditujukan untuk aplikasi yang tidak bisa mengak-
 16 ses atau menanamkan *web browser* untuk mengarahkan *user* kepada *authorization endpoint*.
 17 Contohnya adalah aplikasi yang bersifat *command-line*, *embedded systems*, *game* konsol, dan
 18 beberapa jenis aplikasi *mobile*.

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

22 Setelah aplikasi anda telah mendapatkan *GET oauth/authenticate* atau *GET oauth/a-*
 23 *uthorize URL*, tampilkan URL kepada user agar mereka dapat menggunakan *web browser*
 24 untuk mengakses Twitter.

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



Gambar 2.2: Contoh PIN-based authorization

1 Aplikasi anda harus memungkinkan *user* untuk memasukkan *PIN code* ini untuk menye-
 2 lesaikan *flow* tersebut. Nilai dari *PIN code* harus lolos sebagai *oauth_verifier* untuk *POST*
 3 *oauth/access_token request*. Semua *request* akan berjalan normal kedepannya.

4 2.4 KIRI API

5 KIRI API adalah aplikasi pihak ketiga yang memungkinkan *programmer* mendapatkan data
 6 tentang info jalur transportasi publik. KIRI API dapat diakses dengan beberapa cara.
 7 Semua *request* harus berisikan API key, yang dapat diambil melalui KIRI API *Management*
 8 *Dashboard*. Berikut adalah spesifikasi dari KIRI API

- 9 • *Routing Web Service*
- 10 • *Search Place Web Service*
- 11 • *Nearest Transports Web Service*

12 2.4.1 Routing Web Service

13 *Routing Web Service* adalah salah satu KIRI API yang digunakan untuk mendapatkan lang-
 14 kah perjalanan dari lokasi asal menuju lokasi tujuan.

15 Berikut ini adalah parameter *request* yang diperlukan:

Listing 2.1: code *respond* pencarian rute

```

16 {
17   "status": "ok" or "error"
18   "routingresults": [
19     {
20       "steps": [
21         [
22           "walk" or "none" or others ,
23           "walk" or vehicle_id or "none",
24           ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
25           "human readable description, dependant on locale",
26           URL for ticket booking or null (future)
27         ],
28         [
29           "walk" or "none" or others ,
30           "walk" or vehicle_id or "none",
31           ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
32           "human readable description, dependant on locale",
33           URL for ticket booking or null (future)
34         ]
35       ],
36       "traveltime": any text string, null if and only if route is not found.

```

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

```

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

- *Index* 0 (baris ke) berisikan "walk" atau "none" atau "others". Arti dari "walk" adalah 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 mengambil 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 ikon penanda yang menunjukkan titik awal atau "from". Biasanya digunakan untuk mode presentasi perangkat bergerak.
 - %toicon = sebuah ikon 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.

2.4.2 Search Place Web Service

Search Place Web Service berguna untuk menemukan rute perjalanan berdasarkan *latitude* 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:

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

Tabel 2.8: Tabel parameter *Search Place Web Service*

Berikut format kembalian dari Kiri API:

Listing 2.2: code *respond* pencarian lokasi

```

9  {
10   "status": "ok" or "error"
11   "searchresult": [
12     {
13       "placename": "place name"
14       "location": "lat,lon"
15     },
16     {
17       "placename": "place name"
18       "location": "lat,lon"
19     },
20     ...
21   ]
22   "attributions": [
23     "attribution_1", "attribution_2", ...
24   ]
25 }
```

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 penjelasannya:

Berikut format kembalian dari Kiri API:

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

Tabel 2.9: Tabel parameter *Nearest Transports Web Service*Listing 2.3: code *respond* menemukan lokasi terdekat

```

1  {
2      "status": "ok" or "error"
3      "nearbytransports": [
4          [
5              "walk" or "none" or others,
6              "walk" or vehicle_id or "none",
7              text string,
8              decimal value
9          ],
10         [
11             "walk" or "none" or others,
12             "walk" or vehicle_id or "none",
13             text string,
14             decimal value
15         ],
16         ...
17     ]
18 }

```

Pencarian akan memberitahukan status berhasil ("ok") atau tidak ("error"), jika sukses maka respon akan mengembalikan array yang berisikan transportasi terdekat yang diurutkan dari yang terdekat 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 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.

2.5 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 listener.

- 1 – void clearListeners()
2 Menghilangkan *status listener*.
- 3 – void replaceListener(StreamListener toBeRemoved, StreamListener toBeAdded)
4 Menimpa listener yang sudah ada sebelumnya.
- 5 – void firehose(int count)
6 Mendengarkan semua status publik.
- 7 – void links(int count)
8 Mendengarkan semua status publik yang mengandung link.
- 9 – void retweet()
10 Mendengarkan semua retweet.
- 11 – void sample()
12 Mendengarkan status publik secara acak.
- 13 – void user()
14 *User Streams* menyediakan update dari semua data secara *real-time*.
- 15 – void user(java.lang.String[] track)
16 *User Streams* menyediakan update dari semua data secara *real-time*. Parameter
17 track merupakan kata kunci untuk kata yang akan ditampilkan.
- 18 – StreamController site(boolean withFollowings, long[] follow)
19 Menerima update secara *real-time* untuk sejumlah pengguna tanpa perlu kere-
20 potan dalam mengelola REST API *rate limits*.
- 21 – void filter(FilterQuery query)
22 Menerima status publik yang telah di *filter* dari satu atau lebih kata kunci.
- 23 – void cleanUp()
24 Menon-aktifkan penggunaan *thread stream*.
- 25 – void shutdown()
26 Menon aktifkan *dispatcher thread* bersama dengan semua instansi TwitterStream.

27 2.5.3 TwitterStreamFactory

28 • *Constant*

- 29 – public final class TwitterStreamFactory extends java.lang.Object implements ja-
30 va.io.Serializable
31 Sebuah *factory class* untuk Twitter. Instansi dari kelas ini memiliki thread yang
32 aman dan digunakan secara berkala lalu dapat digunakan kembali.

33 • *Constructor*

- 34 – TwitterStreamFactory() Membuat TwitterStreamFactory dengan konfigurasi dari
35 sumber.
- 36 – TwitterStreamFactory(Configuration conf) Membuat TwitterStreamFactory de-
37 ngan konfigurasi yang diberikan.

1 – `TwitterStreamFactory(java.lang.String configTreePath)` Membuat `TwitterStreamFactory` yang berasal dari *config tree* yang spesifik.

3 • *Methods*

4 – `public TwitterStream getInstance()`
 5 Mengembalikan contoh yang terkait dengan konfigurasi.

6 – `public TwitterStream getInstance(AccessToken accessToken)`
 7 Mengembalikan OAuth yang sudah diotentifikasi.

8 – `public TwitterStream getInstance(Authorization auth)`
 9 Mengembalikan *instance*.

10 – `private TwitterStream getInstance(Configuration conf, Authorization auth)`
 11 Mengembalikan *instance* dengan konfigurasi dan otorisasi yang sesuai.

12 – `public static Twitter getSingleton()`
 13 Mengembalikan *singleton* standar Twitter *instance*.

14 2.5.4 UserStreamListener

15 • *Constant*

16 – `public interface UserStreamListener extends StatusListener`

17 • *Methods*

18 – `void onDeleteNotice(long directMessageId, long userId)`

19 – `void onFriendList(long[] friendIds)`

20 – `void onFavorite(User source, User target, Status favoritedStatus)`

21 – `void onUnfavorite(User source, User target, Status unfavoritedStatus)`

22 – `void onFollow(User source, User followedUser)`

23 – `void onUnfollow(User source, User unfollowedUser)`

24 – `void onDirectMessage(DirectMessage directMessage)`

25 – `void onUserListMemberAddition(User addedMember, User listOwner, UserList list)`

26 – `void onUserListMemberDeletion(User deletedMember, User listOwner, UserList list)`

27 – `void onUserListSubscription(User subscriber, User listOwner, UserList list)`

28 – `void onUserListUnsubscription(User subscriber, User listOwner, UserList list)`

29 – `void onUserListCreation(User listOwner, UserList list)`

30 – `void onUserListUpdate(User listOwner, UserList list)`

31 – `void onUserListDeletion(User listOwner, UserList list)`

32 – `void onUserProfileUpdate(User updatedUser)`

33 – `void onBlock(User source, User blockedUser)`

34 – `void onUnblock(User source, User unblockedUser)`

37 *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 *requestnya*.

- UploadedMedia uploadMedia(java.io.File mediaFile) throws TwitterException

- Melakukan *upload* media gambar yang telah di lampirkan via updateStatus(twitter4j.StatusUpdate)

2.5.6 OAuthSupport

- *Constant*

- public interface OAuthSupport

- *Methods*

- 1 – void setOAuthConsumer(java.lang.String consumerKey, java.lang.String consu-
2 merSecret)
- 3 Melakukan pengaturan terhadap *consumer key* dan *consumer secret* .
- 4 – RequestToken getOAuthRequestToken() throws TwitterException
- 5 Mengambil *request token*.
- 6 – RequestToken getOAuthRequestToken(java.lang.String callbackURL) throws Twit-
7 terException
- 8 Mengambil *request token*.
- 9 – RequestToken getOAuthRequestToken(java.lang.String callbackURL, java.lang.String
10 xAuthAccessType) throws TwitterException
- 11 Mengambil *request token*.
- 12 – AccessToken getOAuthAccessToken() throws TwitterException
- 13 Mengembalikan *access token* yang terkait dengan instansi ini. Jika tidak ada
14 instansi pada *access token* maka akan mengambil *access token* yang baru.
- 15 – AccessToken getOAuthAccessToken(java.lang.String oauthVerifier) throws Twit-
16 terException
- 17 Mengambil *request token*.
- 18 – AccessToken getOAuthAccessToken(RequestToken requestToken) throws Twitte-
19 rException
- 20 Mengambil *access token* yang terkait dengan *request token* dan *userId* yang telah
21 diberikan
- 22 – AccessToken getOAuthAccessToken(RequestToken requestToken, java.lang.String
23 oauthVerifier) throws TwitterException
- 24 Mengambil *access token* yang terkait dengan *request token* dan *userId* yang telah
25 diberikan
- 26 – AccessToken getOAuthAccessToken(java.lang.String screenName, java.lang.String
27 password) throws TwitterException
- 28 Mengambil *access token* yang terkait dengan *screen named* dan *password* yang telah
29 diberikan
- 30 – void setOAuthAccessToken(AccessToken accessToken)
- 31 Melakukan pengaturan pada *access token*

32 2.5.7 RequestToken

33 • Constant

- 34 – public final class RequestToken extends OAuthToken implements java.io.Serializable

35 • Constructor

- 36 – RequestToken(HttpResponse res, OAuthSupport oauth)
- 37 – RequestToken(java.lang.String token, java.lang.String tokenSecret)

1 – RequestToken(java.lang.String token, java.lang.String tokenSecret, OAuthSupport oauth)
2

3 • *Methods*

4 – public java.lang.String getAuthorizationURL()

5 – public java.lang.String getAuthenticationURL()

6 **2.5.8 AccessToken**

7 • *Constant*

8 – public class AccessToken extends OAuthToken implements java.io.Serializable

9 • *Constructor*

10 – AccessToken(HttpResponse res)

11 – AccessToken(java.lang.String token, java.lang.String tokenSecret)

12 – AccessToken(java.lang.String token, java.lang.String tokenSecret, long userId)

13 • *Methods*

14 – public java.lang.String getScreenName()

15 Mengembalikan *screen name*

16 – public long getUserId()

17 Mengembalikan *user id*

18 – public boolean equals(java.lang.Object o)

19 – public int hashCode()

20 – public java.lang.String toString()

21 **2.5.9 Status**

22 • *Constant*

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

25 • *Methods*

26 – java.util.Date getCreatedAts()

27 Mengembalikan *created_at*

28 – public long getUserId()

29 Mengembalikan *user id*

30 – java.lang.String getText()

31 Mengembalikan teks dari status

```
1      - java.lang.String getSource()
2          Mengembalikan source
3      - boolean isTruncated()
4          Menguji apakah sebuah status terpotong atau tidak
5      - long getInReplyToStatusId()
6          Mengembalikan in_reply_to_status_id
7      - long getInReplyToUserId()
8          Mengembalikan in_reply_user_id
9      - java.lang.String getInReplyToScreenName()
10         Mengembalikan in_reply_to_screen_name
11     - GeoLocation getGeoLocation()
12         Mengembalikan lokasi dari suatu tweet jika tersedia.
13     - Place getPlace()
14         Mengembalikan tempat yang terdapat pada sebuah status.
15     - boolean isFavorited()
16         Menguji apakah status tersebut favorite atau tidak
17     - boolean isRetweeted()
18         Menguji apakah status tersebut retweet atau tidak
19     - int getFavoriteCount()
20         Menunjukkan berapa kali Tweet telah menjadi favorite
21     - User getUser()
22         Mengembalikan user yang terdapat pada sebuah status.
23     - boolean isRetweet()
24     - Status getRetweetedStatus()
25     - long[] getContributors()
26         Mengembalikan array yang berisi kontributor atau mengembalikan null jika tidak
27         ada kontributor yang terkait dengan status ini
28     - int getRetweetCount()
29         Menunjukkan berapa kali Tweet telah di retweet, jika belum terdapat maka akan
30         mengembalikan nilai -1
31     - boolean isRetweetedByMe()
32         Mengembalikan nilai true jika user yang telah diotentifikasi melakukan retweet
33         terhadap suatu tweet, atau mengembalikan nilai false jika tidak.
34     - long getCurrentUserRetweetId()
35         Mengembalikan retweet id sebuah tweet dari user yang telah diotentifikasi, jika
36         belum terdapat maka akan mengembalikan nilai -1L
37     - boolean isPossiblySensitive()
38         Mengembalikan nilai true jika pada status terdapat sensitive links
```

- 1 – java.lang.String getLang()
2 Mengembalikan *lang* dari sebuah status teks jika tersedia
- 3 – Scopes getScopes()
4 Mengembalikan target dari *scopes* yang diaplikasikan kepada sebuah status.

5 2.5.10 TweetsResources

6 • *Constant*

- 7 – public interface TweetsResources

8 • *Methods*

- 9 – ResponseList<Status> getRetweets(long statusId) throws TwitterException
10 Mengembalikan hingga dengan seratus *retweet* pertama
- 11 – IDs getRetweeterIds(long statusId, long cursor) throws TwitterException
12 Mengembalikan hingga dengan 100 *user ID* yang melakukan *retweet* terhadap
13 *tweet* ditentukan dari *id parameter*
- 14 – IDs getRetweeterIds(long statusId, int count, long cursor) throws TwitterException
15 Mengembalikan hingga dengan "*count*" *user ID* yang melakukan *retweet* terhadap
16 *tweet* ditentukan dari *id parameter*
- 17 – Status showStatus(long id) throws TwitterException
18 Mengembalikan *status* yang ditentukan dari parameter *id*.
- 19 – Status destroyStatus(long statusId) throws TwitterException
20 Menghapus *status* yang ditentukan dari parameter *id*.
- 21 – Status updateStatus(java.lang.String status) throws TwitterException
22 Melakukan *update status* terhadap *user* yang telah diotentifikasi.
- 23 – Status updateStatus(StatusUpdate latestStatus) throws TwitterException
24 Melakukan *update status* terhadap *user* yang telah diotentifikasi.
- 25 – Status retweetStatus(long statusId) throws TwitterException
26 Melakukan *retweet* terhadap sebuah *tweet*.
- 27 – OEmbed getOEmbed(OEmbedRequest req) throws TwitterException
28 Mengembalikan informasi yang mengizinkan terciptanya *embedded representation*
29 dari *tweet* yang berada di *third party sites*
- 30 – ResponseList<Status> lookup(long[] ids) throws TwitterException
31 Mengembalikan objek *tweet* hingga dengan 100 *tweet* per **request**.
- 32 – UploadedMedia uploadMedia(java.io.File mediaFile) throws TwitterException
33 Melakukan *upload* gambar.
- 34

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 pengguna 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 proses otentifikasi tidak perlu dilakukan kepada pengguna karena Twitter Bot yang dibuat menggunakan otentikasi 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 otentikasi yang dapat memposting *tweet* dan melakukan koneksi dengan *streaming endpoint*. Lalu untuk otentikasi *PIN-based authorization* tidak cocok karena otentikasi sudah dilakukan langsung dari developer tidak lagi meminta PIN untuk proses otentikasi.

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 *latitude* 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®ion=cgk/bdo/sub&querystring='string'&apikey=889C2C8FBB82C7E6](https://dev.kiri.travelbukitjarian/handle.php?version=2&mode=searchplace®ion=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.

- 1 2. mode : "searchplace"
- 2 Mode "searchplace" merupakan mode dari *Search Place Web Service* yang digu-
- 3 nakan untuk mencari lokasi.
- 4 3. region : bdo
- 5 *Region* berfungsi sebagai parameter untuk memberitahukan kota yang akan men-
- 6 jadi bagian dalam pencarian lokasi. Parameter yang terdapat di region ada tiga
- 7 yaitu "cgk" untuk Kota Jakarta, "bdo" untuk Kota Bandung, dan "sub" untuk
- 8 Kota Surabaya.
- 9 4. querystring
- 10 Merupakan kata kunci untuk lokasi.
- 11 5. apikey : 889C2C8FBB82C7E6
- 12 Merupakan *password* yang digunakan untuk mengakses KIRI API.

13 Penulis mencoba mencari lokasi pvj dari kata kata kunci "pvj" yang berada di Kota
 14 Bandung. Layanan dikirimkan ke URL `kiri.travel/handle.php`. Berikut adalah for-
 15 mat layanan yang dituliskan: <http://kiri.travel/handle.php?version=2&mode=searchplace®ion=bdo&querystring=pvj&apikey=889C2C8FBB82C7E6>
 16 <http://kiri.travel/handle.php?version=2&mode=searchplace®ion=bdo&querystring=pvj&apikey=889C2C8FBB82C7E6>

17 Berikut adalah hasil kembalian dari KIRI API:

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

```

18 {
19     "status": "ok",
20     "searchresult": [
21         {
22             "placename": "J.Co Donuts & Coffee",
23             "location": "-6.88929,107.59574"
24         },
25         {
26             "placename": "Pepper Lunch Bandung (PVJ)",
27             "location": "-6.88923,107.59615"
28         },
29         {
30             "placename": "Domino's Pizza Pvj",
31             "location": "-6.90348,107.61709"
32         },
33         {
34             "placename": "Outlet Alleira Batik PVJ Bandung",
35             "location": "-6.88875,107.59634"
36         },
37         {
38             "placename": "Burger King Bandung PVJ Mall",
39             "location": "-6.88894,107.59342"
40         },
41         {
42             "placename": "Killiney Kopitiam PVJ",
43             "location": "-6.88947,107.59654"
44         },
45         {
46             "placename": "Adidas Pvj",
47             "location": "-6.88909,107.59614"
48         },
49         {
50             "placename": "Crocs - PVJ",
51             "location": "-6.88894,107.59342"
52         },
53         {
54             "placename": "Cross Pvj",
55             "location": "-6.88906,107.59619"
56         },
57         {
58             "placename": "Jonas Photo - PVJ",
59             "location": "-6.88913,107.59643"
60         }
61     ],
62     "attributions": null
63 }

```

• *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 Inggris 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 penggunaanya 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"],
```

```

1           "Walk about 92 meter from your starting point \\\%fromicon to Jalan
2             Sukajadi \\\%toicon.",
3           null
4         ],
5         [
6           "angkot",
7           "kalapakarangsetra",
8           ["-6.88958,107.59691","-6.89052,107.59696","-6.89146,107.59701","-6.89239,107.59706"],
9
10          "Take angkot Kalapa - Karang Setra at Jalan Sukajadi \\\%fromicon, and
11            alight at Jalan Pajajaran \\\%toicon about 2.6 kilometer later
12            .",
13          null
14        ],
15        [
16          "angkot",
17          "ciroyomantapani",
18          ["-6.90713,107.60441","-6.90713,107.60441","-6.90679,107.60440","-6.90563,107.60438"],
19
20          "Take angkot Ciroyom - Antapani at Jalan Pajajaran \\\%fromicon, and
21            alight at Jalan Aceh \\\%toicon about 1.7 kilometer later.",
22          null
23        ],
24        [
25          "walk",
26          "walk",
27          ["-6.90974,107.61091","-6.90864,107.61108"],
28          "Walk about 124 meter from Jalan Aceh \\\%fromicon to your destination
29            \\\%toicon.",
30          null
31        ]
32      ],
33      "traveltime ":"25 minutes"
34    }
35  }
36

```

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

```

53  {
54      debug=true
55      oauth . consumerKey=3iT8duMItTTTrdaU1qTHxwDIU1
56      oauth . consumerSecret=YUIgJtBQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****

```

```

1      |      oauth . accessToken = 313287708 - NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
2      |      oauth . accessTokenSecret = LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYospYoOptc****
3      |      }

```

4 2. via *ConfigurationBuilder*

5 Menggunakan *ConfigurationBuilder* class untuk melakukan konfigurasi Twitter4J.

Listing 3.4: isi dari twitter4j.properties

```

6      |      {
7      |      ConfigurationBuilder cb = new ConfigurationBuilder();
8      |      cb.setDebugEnabled(true)
9      |      .setOAuthConsumerKey("3iT8duMItTTrdaU1qTHxwDIU1")
10     |      .setOAuthConsumerSecret("YUlgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****")
11     |      .setOAuthAccessToken("313287708 - NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ")
12     |      .setOAuthAccessTokenSecret("LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYospYoOptc****");
13     |      TwitterFactory tf = new TwitterFactory(cb.build());
14     |      Twitter twitter = tf.getInstance();
15     |      }

```

16 3. via *System Properties*

17 Menggunakan *System Properties* untuk melakukan konfigurasi Twitter4J.

Listing 3.5: isi dari twitter4j.properties

```

18     |      $ export twitter4j.debug=true
19     |      $ export twitter4j.oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIU1
20     |      $ export twitter4j.oauth.consumerSecret=
21     |      YUlgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****
22     |      $ export twitter4j.oauth.accessToken=313287708 -
23     |      NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
24     |      $ export twitter4j.oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYospYoOptc
25     |      ****
26     |      $ java -cp twitter4j-core-4.0.2.jar:yourApp.jar yourpackage.Main

```

27 3.2 Analisis Perangkat Lunak

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

35 3.2.1 Spesifikasi Kebutuhan Fungsional

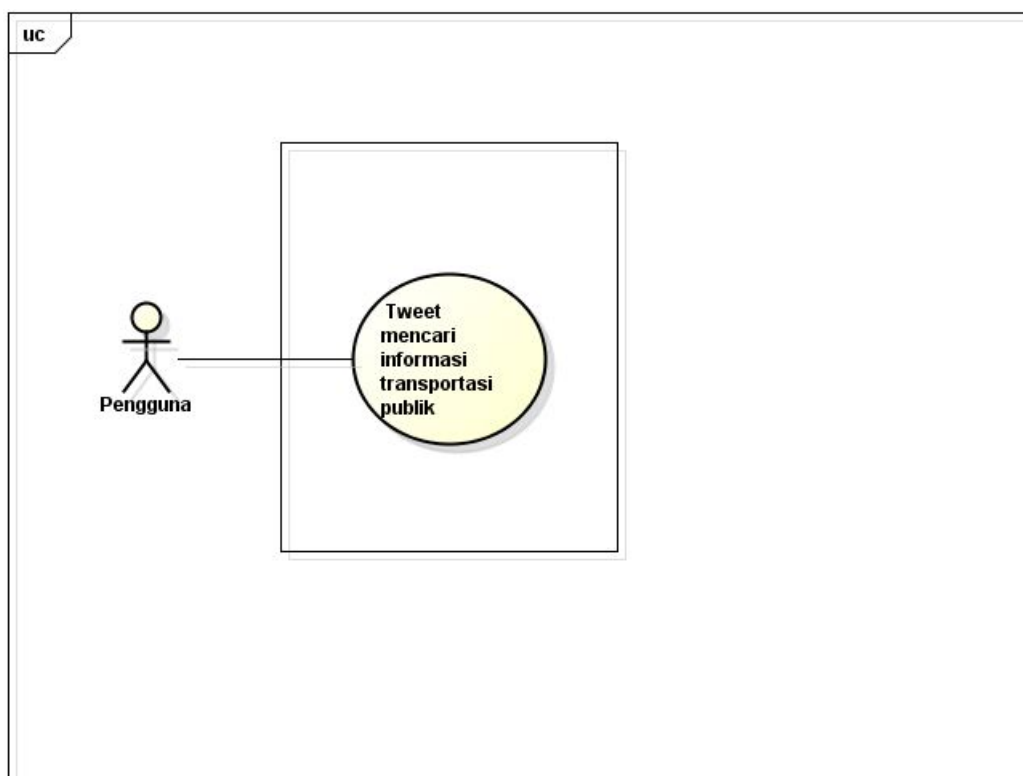
36 Spesifikasi kebutuhan perangkat lunak yang akan dibangun untuk membuat Twitter Bot
 37 adalah

- 38 1. Perangkat lunak dapat melakukan otentikasi untuk akun Twitter Bot yang digunakan.
- 39 2. ??Perangkat lunak?? dapat menerima dan membaca *tweet* yang di *mention* kepada
 40 user @kviniink
- 41 3. Dapat Melakukan proses pencarian koordinat suatu lokasi
- 42 4. Dapat melakukan proses pencarian jalur transportasi publik dari lokasi awal menuju
 43 lokasi tujuan

- 1 5. Dapat membalas *tweet* pencarian jalur transportasi publik yang diterima oleh Twitter
- 2 bot dengan melakukan *reply* tweet yang berisikan hasil pencarian jalur transportasi
- 3 publik dengan format yang sudah ditentukan.

4 3.2.2 Use Case Diagram

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



Gambar 3.1: Use case Twitter Bot

- 7 **Skenario Use Case** Skenario ini hanya memiliki satu aktor yaitu pengguna. *Tweet*
- 8 mencari informasi transportasi publik pada skenario ini dilakukan dengan melakukan *tweet*
- 9 kepada user @kiriupdate berisikan format yang sesuai untuk pencarian rute transportasi.

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

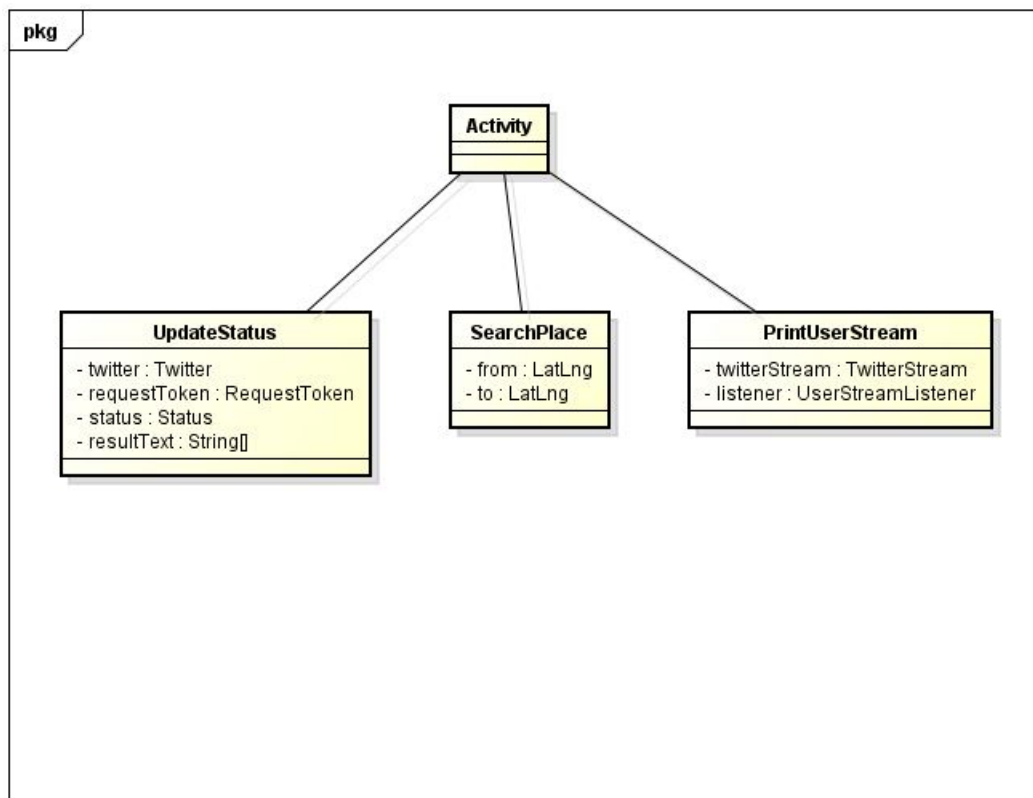
Tabel 3.1: Skenario *Tweet* mencari informasi transportasi

3.2.3 Class Diagram

Untuk membuat *class diagram* Twitter Bot untuk mencari jalur transportasi publik, dibutuhkan kebutuhan kelas dari skenario. 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. Melakukan *reply* kepada pengguna berupa jalur transportasi publik yang harus ditempuh.

Berikut adalah *class diagram* sederhana:



Gambar 3.2: *Class Diagram* Twitter Bot

BAB 4

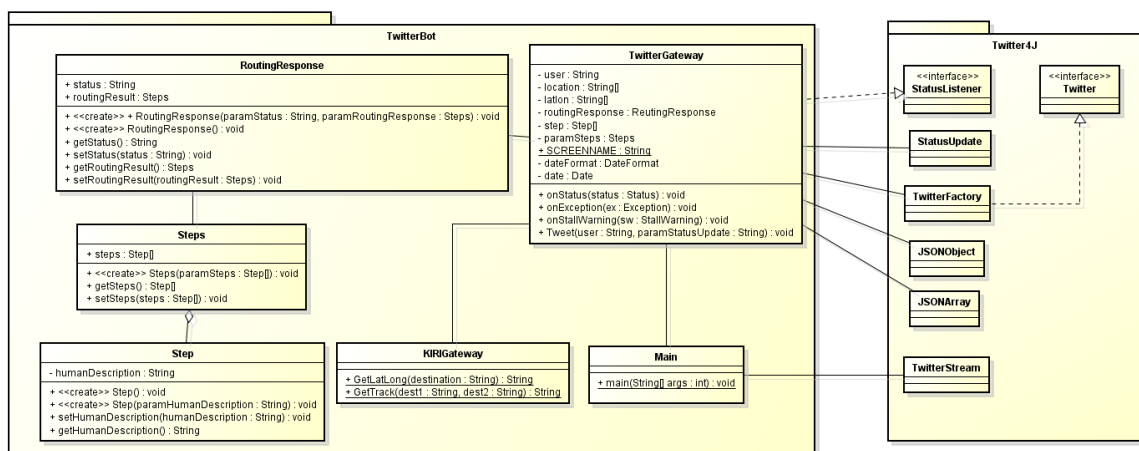
PERANCANGAN PERANGKAT LUNAK

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 dimasukkan 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 dimasukkan 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 melakukan pembagian *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

- 1 * public static String GetLatLong(String destination), merupakan *method* yang
- 2 digunakan untuk mencari koordinat dari suatu lokasi. Hasil kembalian dari
- 3 *method* ini berupa *latitude* and *longitude* yang diberikan oleh KIRI API lalu
- 4 diubah ke dalam bentuk *String*.
- 5 * public static String GetTrack(String dest1, String dest2), merupakan *method*
- 6 yang digunakan untuk mencari jalur transportasi publik dari lokasi awal ke
- 7 lokasi tujuan. Hasil kembalian dari *method* ini adalah langkah-langkah per-
- 8 jalanan dari lokasi awal ke lokasi tujuan dengan menggunakan transportasi
- 9 publik.
- 10 • Kelas RoutingResult, merupakan kelas untuk menampung hasil kembalian dari KIRI
- 11 API
 - 12 – Atribut
 - 13 * status, digunakan untuk menyimpan apakah status dari hasil pencarian.
 - 14 * routingResult, digunakan untuk menyimpan langkah-langkah perjalanan.
 - 15 – Method
 - 16 * public RoutingResponse(String paramStatus, Steps paramRoutingResult),
 - 17 merupakan *constructor* dari kelas RoutingResult.
 - 18 * public RoutingResponse(), merupakan *constructor* dari kelas RoutingResult.
 - 19 * public String getStatus(), merupakan *getter* dari atribut status.
 - 20 * public void setStatus(String status), merupakan *setter* dari atribut status.
 - 21 * public Steps getRoutingResult(), merupakan *getter* dari atribut routingRe-
 - 22 sult.
 - 23 * public void setRoutingResult(Steps routingResult), merupakan *setter* dari
 - 24 atribut routingResult.
- 25 • Kelas Step, merupakan kelas untuk menampung jalur perjalanan dari lokasi awal ke
- 26 lokasi tujuan dengan menggunakan transportasi publik yang diberikan oleh KIRI API.
 - 27 – Atribut
 - 28 * String humanDescription, merupakan atribut untuk menjelaskan cara perja-
 - 29 lanan yang bahasanya dimengerti oleh pengguna.
 - 30 – Method
 - 31 * public Step(), merupakan *constructor* dari kelas Step.
 - 32 * public Step(String paramHumanDescription), merupakan *constructor* dari
 - 33 kelas Step.
 - 34 * public String getHumanDescription(), merupakan *getter* dari atribut human-
 - 35 Description.
 - 36 * public void setHumanDescription(String humanDescription), merupakan *set-*
 - 37 *ter* dari atribut humanDescription.
- 38 • Kelas Steps, merupakan kelas untuk menampung kumpulan step.

- 1 – Atribut
- 2 * Step[] steps, merupakan atribut yang berisi *array* step
- 3 – Method
- 4 * public Steps(Step[] paramSteps), merupakan konstruktor dari kelas Steps.
- 5 * public Step[] getSteps(), merupakan *getter* dari atribut steps.
- 6 * public void setSteps(Step[] steps), merupakan *setter* dari atribut steps.

7 4.1.2 Sequence Diagram

8 Pada sub bab ini, akan dijelaskan alur program dengan menggunakan *sequence diagram* pada
9 4.2

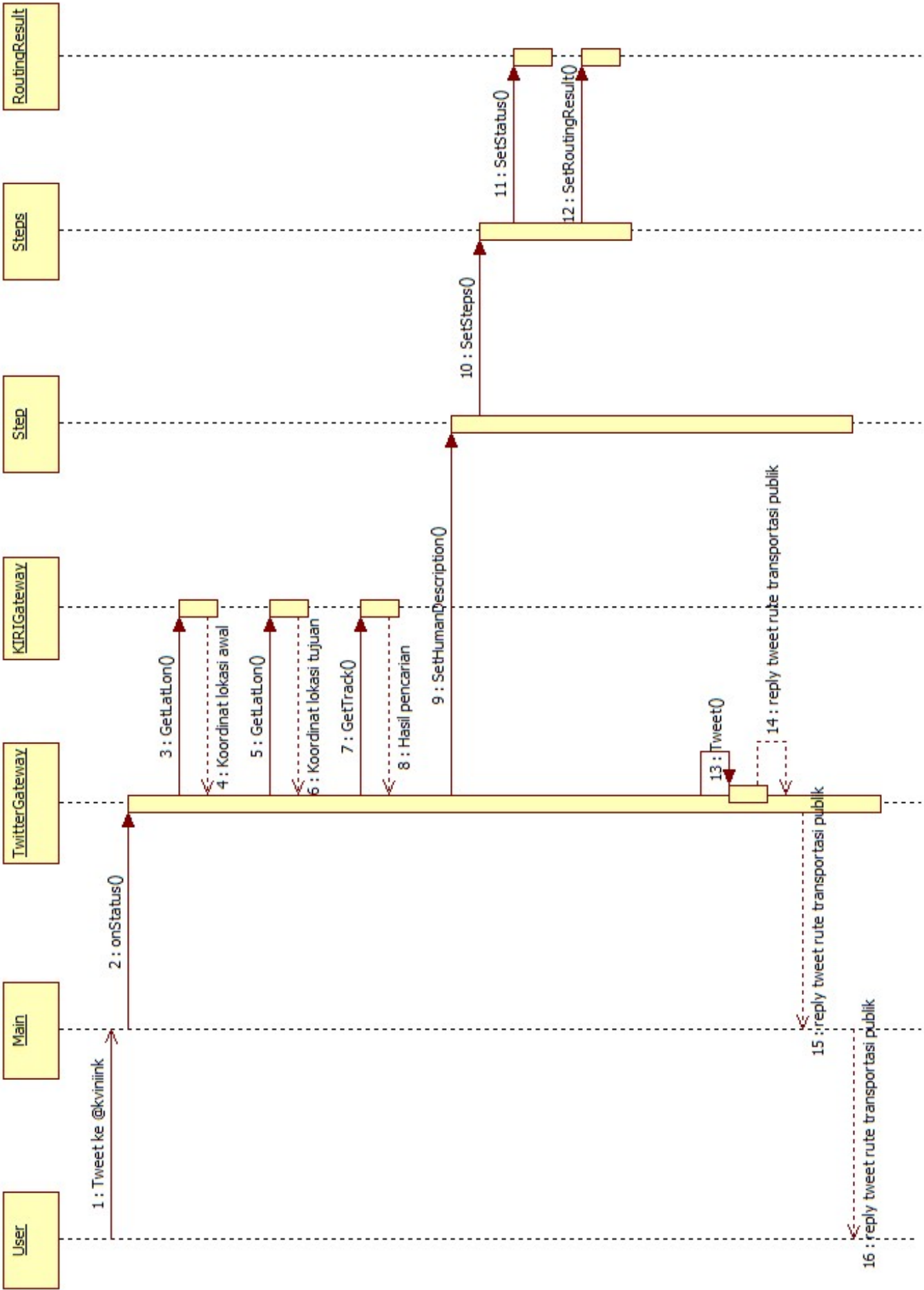
10 Pertama, program akan melakukan *streaming* pada saat kelas *main* dijalankan. Kelas
11 *main* akan membuka gerbang untuk mengakses *Twitter API*, dengan menggunakan *Strea-*
12 *ming* API aplikasi akan menangkap semua *tweet* yang memiliki kata kunci @kviniink. Apli-
13 kasi akan terus melakukan *streaming tweet* hingga aplikasi dinon-aktifkan.

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

20 Kelas KIRIGateway akan memanggil *method GetLatLon* untuk mencari koordinat suatu
21 lokasi. Setelah didapatkan koordinat dari masing-masing lokasi maka kelas TwitterGate-
22 way akan mengolahnya terlebih dahulu dikarenakan hasil dari *method GetLatLon* ini berupa
23 JSON. Setelah itu maka hasilnya akan dikembalikan kepada kelas KIRIGateway untuk di-
24 cari jalur transportasi publik dari lokasi awal menuju lokasi tujuan menggunakan *method*
25 *GetTrack*. Hasil dari *method GetTrack* akan disimpan pada atribut *step*, *steps*, dan *routi-*
26 *ngResult*.

27 Setelah selesai, langkah-langkah jalur transportasi publik siap di *reply*. Proses *reply*
28 dilakukan oleh *method tweet* yang terdapat pada kelas TwitterGateway. Tweet tersebut
29 berisi tentang jalur transportasi publik dari lokasi awal menuju lokasi tujuan. *Tweet* akan di-
30 *reply* satu per satu sesuai dengan banyaknya *step* yang ada. Aplikasi akan terus melakukan
31 proses tersebut hingga aplikasi dinon-aktifkan.

Diagram Final.jpg



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

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 : NVIDIA GeForce GT 540M
- Sistem operasi: Windows 7 Professional
- Platform: NetBeans: IDE 8.0.2
- Akun Twitter Bot
 - Nama akun: kviniink
 - ConsumerKey : 3iT8duMiTTTrdaU1qTHxwDIU1
 - ConsumerSecret : YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ*****
 - AccessToken : 313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
 - AccessTokenSecret : LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOpt*****
- Akun Twitter penguji : kviniinktest123

5.2 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.

5.2.1 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 ditunjukkan pada tabel 5.1.

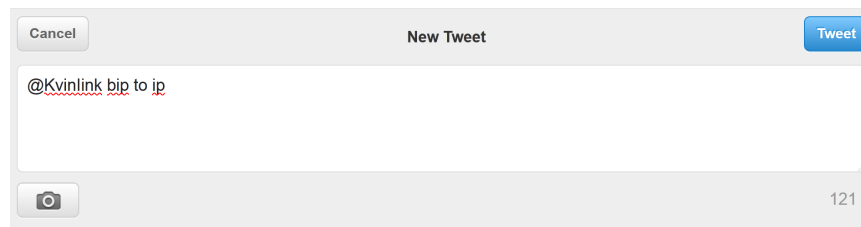
No	Pengujian	Reaksi yang Diharapkan	Reaksi Aplikasi
1	Melakukan otentikasi terhadap akun Twitter Bot	Otentikasi berhasil dilakukan antara Twitter dengan akun Twitter Bot. Otentikasi dilakukan dengan melakukan pemeriksaan terhadap <i>ConsumerKey</i> , <i>CustomerSecret</i> , <i>AccessToken</i> , dan <i>AccessTokenSecret</i>	<i>ConsumerKey</i> , <i>CustomerSecret</i> , <i>AccessToken</i> , dan <i>AccessTokenSecret</i> yang diberikan Twitter berhasil diotentikasi oleh aplikasi
2	Melakukan <i>streaming tweet</i>	Menangkap semua <i>tweet</i> yang <i>dimention</i> kepada akun @kvinink	Setiap <i>tweet</i> yang <i>dimention</i> kepada akun @kvinink dapat diterima secara <i>realtime</i>
3	Membaca <i>tweet</i> yang ditangkap	Melakukan pemeriksaan terhadap <i>tweet</i> yang ditangkap, apakah <i>tweet</i> tersebut merupakan <i>tweet</i> untuk mencari transportasi publik atau bukan	Perangkat lunak dapat membedakan <i>tweet</i> untuk mencari jalur transportasi publik dengan <i>tweet</i> yang bukan bertujuan untuk mencari jalur transportasi publik
4	Melakukan pencarian koordinat suatu lokasi menggunakan KIRI API	Mendapatkan hasil koordinat <i>latitude</i> dan <i>longitude</i> dari lokasi yang dicari	Perangkat lunak mendapatkan koordinat <i>latitude</i> dan <i>longitude</i> 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 <i>tweet</i> balasan	Membalas <i>tweet</i> dengan memberikan hasil pencarian jalur transportasi publik dengan format yang sudah ditentukan	Akun @kvinink melakukan reply kepada akun penguji @kvininktest123, reply tersebut berisikan jalur transportasi publik yang harus ditempuh dari lokasi awal menuju lokasi tujuan.

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

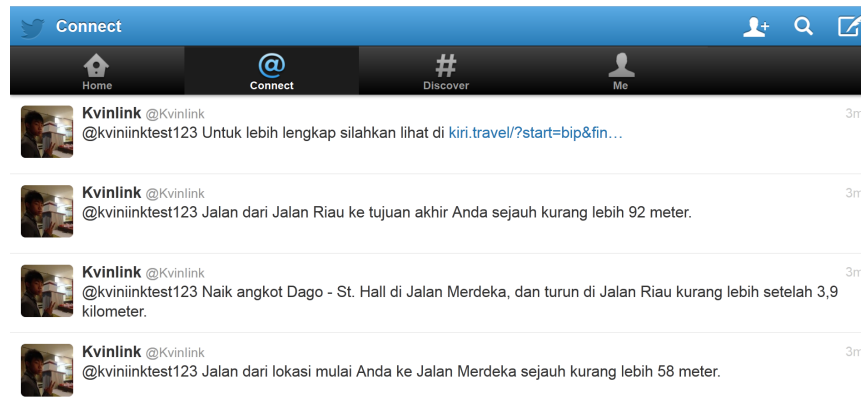
5.2.2 Pengujian Eksperimental

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 @kvininktest123.

1. Pengujian 1

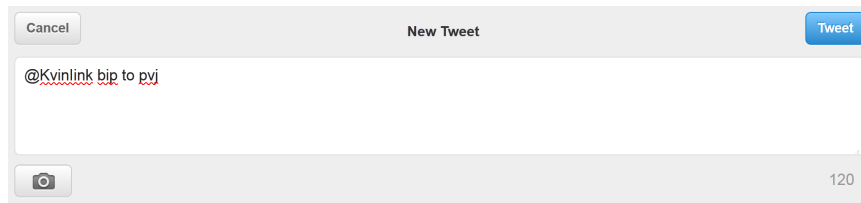


Gambar 5.1: Tweet dari BIP menuju IP

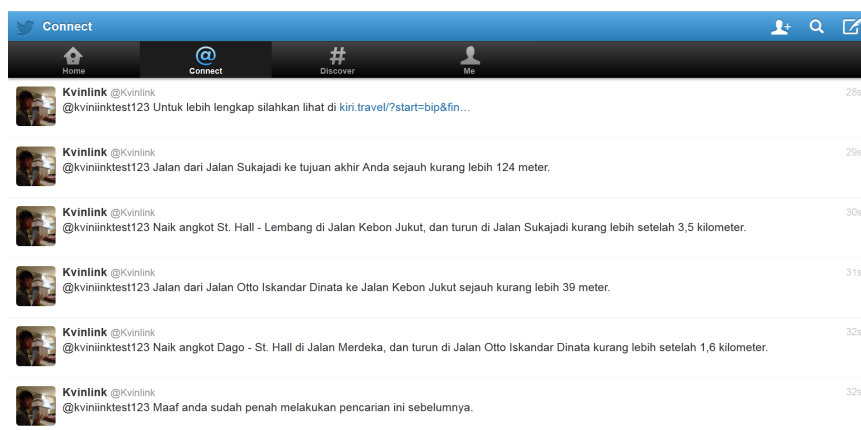


Gambar 5.2: Hasil Pencarian Rute Transportasi Publik dari BIP menuju IP


1 Pada pengujian 1, peneliti mencoba untuk mencari jalur transportasi publik untuk
 2 lokasi yang umum dikunjungi yaitu *mall*. Pencarian dilakukan dengan lokasi awal
 3 adalah BIP (Bandung Indah Plaza) menuju lokasi tujuan adalah IP (Istana Plaza).
 4 Akun penguji @kvininkttest123 melakukan tweet kepada Twitter Bot akun @kvinlink
 5 yang dapat dilihat pada gambar 5.1. Setelah proses *tweet* dilakukan, Twitter Bot akan
 6 menangkap tweet tersebut dan memproses *tweet*. Setelah proses pencarian selesai di-
 7 lakukan, lalu Twitter Bot akun @kvinlink melakukan *reply* kepada akun @kvininkttest
 8 yang dapat dilihat pada gambar 5.2. Pencarian ke dua dilakukan pencarian dengan
 9 lokasi awal adalah BIP (Bandung Indah Plaza) dan lokasi tujuan ada PVJ (Paris
 10 van Java). Dapat dilihat pada gambar 5.3, akun @kvininkttest123 melakukan *tweet*
 11 pencarian jalur transportasi publik yang di *mention* kepada akun @kvinlink dengan
 12 lokasi awal adalah BIP dan lokasi tujuan adalah PVJ. Setelah itu tweet tersebut proses
 13 oleh aplikasi untuk dicari jalur transportasi publiknya, lalu Twitter Bot akun @kvi-
 14 niink melakukan *reply* kepada akun @kvininkttest. Reply tweet tersebut merupakan
 15 jalur transportasi publik yang harus ditempuh, *reply tweet* tersebut dapat dilihat pada
 16 gambar 5.4. Pada pencarian ke dua dapat dilihat pada tweet pertama terjadi ketidak-
 17 sesuaian hasil dari KIRI API dengan hasil tweet. Peneliti lalu melakukan pencarian
 18 melalui website KIRI yaitu <http://kiri.travel>. Pencarian pertama dilakukan dengan
 19 lokasi awal adalah BIP dan lokasi tujuan adalah IP. Hasil pencarian KIRI tersebut
 20 dapat dilihat pada gambar 5.5. Lalu pencarian kedua dilakukan dengan lokasi awal
 21 adalah BIP dan lokasi tujuan adalah PVJ. Hasil pencarian KIRI dari BIP menuju
 22 PVJ dapat dilihat pada gambar 5.6.



Gambar 5.3: Tweet dari BIP menuju PVJ



Gambar 5.4: Hasil Pencarian Rute Transportasi Publik dari BIP menuju PVJ



Bandung English

From: bip

Hypermart - BIP Plaza


To: ip



IP Computer


Find!



↕ ↺

25 minutes 5 minutes


 Walk about 58 meter from your starting point to Jalan Merdeka.

 Take angkot Dago - St. Hall at Jalan Merdeka, and alight at Jalan Riau about 3.9 kilometer later. 

 Walk about 92 meter from Jalan Riau to your destination.

Gambar 5.5: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju IP Melalui Website KIRI



Bandung

English

From:

bip


Hypermart - BIP Plaza


To:

pvj


J.Co Donuts & Coffee

Find!








30 minutes




Walk about 58 meter from your starting point to Jalan Merdeka.




Take angkot Dago - St. Hall at Jalan Merdeka, and alight at Jalan Otto Iskandar Dinata about 1.6 kilometer later. 





Walk about 39 meter from Jalan Otto Iskandar Dinata to Jalan Kebon Jukut.



Take angkot St. Hall - Lembang at Jalan Kebon Jukut, and alight at Jalan Sukajadi about 3.5 kilometer later.



Walk about 124 meter from Jalan Sukajadi to your destination.



Gambar 5.6: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI

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/>