

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

LEMBAR PENGESAHAN

PEMBUATAN *TWITTER BOT* UNTUK MENCARI JALUR
TRANSPORTASI PUBLIK

KEVIN THEODORUS YONATHAN

NPM: 2011730037

Bandung, 4 Juli 2014

Menyetujui,

Pembimbing Tunggal

Pascal Alfadian, M.Com.

Ketua Tim Penguji

Anggota Tim Penguji

Thomas Anung Basuki, Ph.D.

Dr. rer. nat. Cecilia Esti Nugraheni

Mengetahui,

Ketua Program Studi

Thomas Anung Basuki, Ph.D.

PERNYATAAN

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

PEMBUATAN *TWITTER BOT* UNTUK Mencari Jalur TRANSPORTASI PUBLIK

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

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

Dinyatakan di Bandung,
Tanggal 4 Juli 2014

Meterai

Kevin Theodorus Yonathan
NPM: 2011730037

ABSTRAK

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

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

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

Kata-kata kunci: Twitter, Rute, Transportasi Publik

ABSTRACT

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

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

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

Keywords: Twitter, Route, Public Transport

Dipersembahkan untuk diri sendiri

KATA PENGANTAR

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Bandung, Juli 2014

Penulis

DAFTAR ISI

KATA PENGANTAR	xiii
DAFTAR ISI	xv
DAFTAR GAMBAR	xvii
DAFTAR TABEL	xviii
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>	18
2.5 Twitter4J	18
2.5.1 <i>TwitterFactory</i>	19
2.5.2 <i>TwitterStream</i>	20
2.5.3 <i>TwitterStreamFactory</i>	21
2.5.4 <i>UserStreamListener</i>	21
2.5.5 <i>TweetsResources</i>	22
2.5.6 <i>OAuthSupport</i>	23
2.5.7 <i>RequestToken</i>	24
2.5.8 <i>AccessToken</i>	24
2.5.9 <i>Status</i>	25
2.5.10 <i>TweetsResources</i>	26
3 ANALISIS	29
3.1 Analisis Data	29
3.1.1 Analisis Twitter API	29

3.1.2	Analisis OAuth	30
3.1.3	Analisis KIRI API	30
3.1.4	Analisis Twitter4J	33
3.2	Analisis Perangkat Lunak	34
3.2.1	Spesifikasi Kebutuhan Fungsional	34
3.2.2	<i>Use Case Diagram</i>	35
3.2.3	<i>Class Diagram</i>	36
4	PERANCANGAN PERANGKAT LUNAK	37
4.1	Perancangan Perangkat Lunak	37
4.1.1	Perancangan Kelas	37
4.1.2	Sequence Diagram	40
5	IMPLEMENTASI DAN PENGUJIAN APLIKASI	43
5.1	Lingkungan Pembangunan	43
5.2	Pengujian	43
5.2.1	Pengujian Fungsional	44
5.2.2	Pengujian Eksperimental	44
6	KESIMPULAN DAN SARAN	51
6.1	Kesimpulan	51
6.2	Saran	51
	DAFTAR REFERENSI	53

DAFTAR GAMBAR

2.1	Ilustrasi sign in	14
2.2	Contoh PIN-based authorization	15
3.1	Use case Twitter Bot	35
3.2	<i>Class Diagram</i> Twitter Bot	36
4.1	Class Diagram Pembuatan Twitter Bot untuk Mencari Jalur Transportasi Publik . .	37
4.2	Sequence Diagram Twitter Bot untuk Mencari Jalur Transportasi Publik	41
5.1	Tweet dari BIP menuju IP	45
5.2	Hasil Pencarian Rute Transportasi Publik dari BIP menuju IP	45
5.3	Tweet dari BIP menuju PVJ	45
5.4	Hasil Pencarian Rute Transportasi Publik dari BIP menuju PVJ	46
5.5	Hasil Pencarian Jalur Transportasi Publik dari BIP menuju IP Melalui Website KIRI	47
5.6	Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI	48
5.7	Hasil Reply Twitter Bot	49
5.8	Akun Twitter Bot Mendapat Banyak Mention Dalam Satu Tweet	49
5.9	Hasil Reply Twitter Bot	50
5.10	Hasil Reply Twitter Bot	50
5.11	Hasil Reply Twitter Bot	50

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	35
5.1	Tabel Hasil pengujian fungsionalitas pada Aplikasi Twitter Bot untuk mencari jalur transportasi publik	44

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 online².

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 akun³. *Follow* adalah salah satu istilah dalam Twitter yang bertujuan untuk mengikuti aktivitas *tweet* suatu akun. Sedangkan cara seseorang untuk dapat memberi rujukan kepada akun Twitter yang lainnya adalah dengan cara melakukan *reply* atau lebih dikenal dengan nama *mention*⁴. Sebagai contoh, diketahui akun bernama @kviniink mem-*follow* @infobdg untuk mengetahui perkembangan apa saja yang terjadi di Kota Bandung. Lalu akun @kviniink ingin bertanya tentang info mall yang sedang ramai dikunjungi di Bandung, maka akun @kviniink membuat *mention tweet* kepada akun @infobdg yang berisikan "@infobdg Halo saya ingin bertanya mall apa yang sedang ramai dikunjungi di Bandung yah?".

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 kendaraan⁵. Keuntungan memakai transportasi publik sudah banyak dirasakan di seluruh dunia yaitu untuk

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

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

³Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 7

⁴Dusty Reagan, *Twitter Application Development For Dummies*, Wiley, 2010, page 9

⁵Oris 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

mengatasi kemacetan dan mengurangi pemanasan global. Seiring dengan perkembangan teknologi, menaiki transportasi publik menjadi semakin mudah. Dengan adanya KIRI di Indonesia terutama di Kota Bandung, masyarakat dapat manaiki transportasi publik tanpa harus mengetahui terlebih dahulu jalur yang harus ditempuh pengguna. Pengguna hanya perlu tahu tempat asal dan tempat tujuan untuk menaiki transportasi publik di Kota Bandung.

KIRI API adalah aplikasi pihak ketiga yang memungkinkan *programmer* mendapatkan data tentang info jalur transportasi publik. Twitter API adalah aplikasi pihak ketiga yang memungkinkan *programmer* melakukan manipulasi dan pengolahan data di Twitter. Dengan memanfaatkan KIRI API dan Twitter API peneliti akan membuat *Twitter bot* yang dapat membalas *tweet* untuk mencari jalur transportasi publik. *Twitter bot* yang dibuat akan bersifat *real time* sehingga jika seseorang melakukan mention kepada akun *Twitter bot* maka *Twitter bot* akan menangkapnya dan membalas *mention* tersebut berupa jalur yang harus ditempuh. Contoh dari jalannya *Twitter bot* adalah ketika akun bernama @kviniink melakukan *mention* kepada @kiriupdate untuk bertanya jalur transportasi publik "@kiriupdate bip to ip". Maka Twitter bot @kiriupdate akan menerima *tweet* dari akun @kviniink lalu *tweet* tersebut akan diolah oleh server dan akan di-*reply* dengan tiga buah tweet yaitu

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

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

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

1.2 Rumusan Masalah

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

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

1.3 Tujuan

Tujuan dari penelitian ini adalah:

- Membuat aplikasi *Twitter bot* untuk mencari jalur transportasi publik.

- Membuat aplikasi Twitter yang bekerja secara *real time*.
- Membuat algoritma untuk memecah instruksi dari KIRI API dan mengubahnya ke dalam bentuk *tweet*.

1.4 Batasan Masalah

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

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

1.5 Metode Penelitian

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

1. Melakukan studi literatur, antara lain:
 - KIRI API,
 - REST API Twitter (<https://dev.twitter.com/docs/api/1.1>),
 - Streaming API Twitter (<https://dev.twitter.com/docs/api/streaming>).
2. Mempelajari pembuatan server dalam bahasa Java.
3. Membuat *Twitter bot* 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 salah satu layanan jejaring sosial online yang memungkinkan pengguna melakukan *posting* pesan berbasis teks hingga 140 karakter[2]. Berikut ini adalah daftar istilah umum pada Twitter:

- *Tweet*

Posting pada Twitter disebut sebagai *tweet*. *Tweet* ini akan meneruskan pesan singkat yang ditujukan ke semua *follower* suatu akun¹. Contohnya adalah seorang akun @kviniink ingin menuliskan bahwa hari ini cuaca cerah, maka akun @kviniink akan melakukan *tweet* 'Hari ini cerah yah..'. *Tweet* juga bisa menyertakan *link* untuk video, foto, atau media lain di internet selain teks biasa. URL *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, contohnya pada saat pengguna memasukkan link <http://www.chacha.com/gallery/7253/15-movies-that-make-guys-cry>, maka link tersebut akan dibuat menjadi bit.ly/1uRi8vV.

- *Follow*

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

- *Reply*

Reply adalah cara seseorang untuk dapat memberi rujukan kepada akun Twitter yang lainnya atau lebih dikenal dengan nama *mention*². 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 sedang ramai dikunjungi di Kota

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

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

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

- *Retweet*

Retweet ini merupakan salah satu istilah penting dari Twitter. *Retweet* ini berguna ketika pengguna menemukan *tweet* menarik dan ingin berbagi *tweet* tersebut dengan *follower* akun tersebut. *Retweet* ini juga secara tidak langsung mengatakan bahwa "saya menghormati anda dan pesan yang anda buat" ³.

- *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 akun Twitter. Syarat agar dapat melakukan *direct message* adalah melakukan aksi follow terhadap akun yang akan dikirimkan *direct message*.

- *Timeline*

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

2.2 Twitter API

Twitter API adalah aplikasi pihak ketiga yang memungkinkan *programmer* melakukan manipulasi dan pengolahan data di Twitter. Twitter API 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 melakukan *tweet* 1000 per harinya, baik melalui handphone, website, API, dan sebagainya.
- Total pesan hanya bisa sebanyak 250 per harinya, pada setiap dan semua perangkat.
- 150 permintaan API per jam.
- 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 ada beberapa *Tweet* atau akun akan hilang dari hasil pencarian.

³Tim O'Reilly, *The Twitter Book*, oreilly, 2009, page 47

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

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

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

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 adanya kata " <i>root</i> ".
<i>#haiku</i>	Mengandung <i>hashtag</i> " <i>haiku</i> ".
<i>from:alexiskold</i>	Dikirim melalui akun " <i>alexiskold</i> ".
<i>to:techcrunch</i>	Dikirimkan kepada akun " <i>techcrunch</i> ".
<i>@mashable</i>	Mereferensi kepada akun " <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 adanya 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 di- <i>posting</i> melalui <i>twitterfeed</i> .

Tabel 2.1: Contoh berbagai macam pencarian *tweet*

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

<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

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

- **Result Type.** Seperti hasil yang terdapat pada twitter.com/search, parameter *result_type* memungkinkan hasil pencarian akan berdasarkan *tweet* yang paling baru atau *tweet* yang paling populer atau bahkan gabungan dari keduanya.
- **Geolocatization.** Pencarian tempat tidak tersedia pada API, tetapi ada beberapa cara yang tepat untuk membatasi *query* dengan cara menggunakan parameter geocode lalu menentukan "*latitude, longitude, radius*". Contohnya adalah "37.781157,-122.398720,1mi". Ketika pencarian lokasi, pencarian API akan mencoba menemukan *tweet* yang memiliki *latitude* dan *longitude* yang sudah dimasukan kedalam *query geocode*, jika tidak berhasil maka API akan mencoba menemukan *tweet* yang dibuat oleh pengguna yang lokasi profilnya terdapat pada *latitude* dan *longitude* tersebut. Kesimpulannya adalah hasil pencarian dapat menerima *tweet* yang tidak mencakup informasi *latitude* atau *longitude*.
- **Language.** Bahasa dapat dijadikan parameter untuk mencari tweet yang sesuai dengan bahasa tersebut.
- **Iterating in a result set.** Parameter seperti *count*, *until*, *since_id*, *max_id* memungkinkan untuk melakukan kontrol bagaimana iterasi melalui hasil pencarian.

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

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

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

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

- Anda ingin mencari *tweet* yang paling baru dalam Bahasa Portugal, yang lokasinya dekat Maracana soccer stadium yang terletak di Rio de Janeiro.

- Maka search URL akan seperti ini: https://api.twitter.com/1.1/search/tweets.json?q=&geocode=-22.912214,-43.230182,1km&lang=pt&result_type=recent

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

- Anda ingin mencari *tweet* yang populer yang berasal dari *user* @kviniink yang terdapat *hashtag* #nasa.
- Maka *search* URL akan seperti ini: https://api.twitter.com/1.1/search/tweets.json?q=from%3Akviniink%20%23nasa&result_type=popular

2.2.2 Streaming API

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

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

- *Public stream*

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

- *User Stream*

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

- *Site Stream*

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

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

Endpoints

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

POST statuses/*filter* POST *filter* ini mengembalikan status publik yang sesuai dengan satu atau lebih predikat yang telah di filter. *Multiple parameter* memungkinkan klien untuk menggunakan koneksi tunggal untuk ke *Streaming* API. Antara GET *request* dan POST *request* keduanya didukung oleh POST statuses / *filter* tetapi untuk GET *request* yang memiliki parameter yang

terlalu banyak mungkin akan ditolak karena URL yang terlalu panjang. Gunakanlah POST request untuk menghindari URL yang panjang. *Track*, *follow*, dan lokasi harus dipertimbangkan untuk dapat digabungkan dengan operator OR. *track=foo&follow=1234* ini mengembalikan *tweet* yang memiliki kata "foo" atau dibuat oleh *user* 1234. Akses standar mengizinkan pencarian hingga 400 kata kunci, dan 5000 *follow userids*. Perintah ini dikembalikan dalam format JSON, memerlukan otentifikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk POST *statuses/filter* dapat dilihat pada tabel 2.3

<i>follow</i>	Menentukan pencarian <i>tweet</i> dari suatu akun
<i>track</i>	Kata kunci pencarian untuk di- <i>track</i> .
<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*.

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 sa-

ma akan menyebabkan koneksi terlalu lama akan putus. Klien yang membuat koneksi secara berlebihan baik berhasil ataupun tidak maka IP mereka otomatis akan di *banned*.

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

Endpoints

- GET *user*

<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 diotentikasi atau dikirim juga kepada akun yang difollow oleh akun yang sudah diotentikasi 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 Jika suatu perangkat lunak menggunakan *user stream* maka sebisa mungkin untuk meminimalkan jumlah koneksi suatu perangkat lunak. Setiap akun Twitter terbatas hanya untuk beberapa koneksi *user stream* per otentikasi perangkat lunak, terlepas dari IP. Setelah mencapai batasnya maka koneksi tertua atau terlalu lama akan diberhentikan secara otomatis. *User login* dari beberapa instansi dari otentikasi perangkat lunak 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 akun yang seperti itu akan secara otomatis di *banned* dari *User Stream* untuk tingkat *login* yang berlebihan. Untuk memulihkan akses *streaming user* harus menutup perangkat lunak tambahan yang ada, mungkin berjalan di perangkat atau *device* yang berbeda.

Perhatikan bahwa setiap aplikasi memiliki alokasinya masing-masing, sehingga *login* dari perangkat lunak yang pertama tidak akan mempengaruhi koneksi untuk perangkat lunak ke dua begitu juga sebaliknya. Tetapi menjalankan terlalu banyak salinan perangkat lunak yang pertama maupun

ke dua akan menimbulkan masalah. Perhatikan juga bahwa jumlah koneksi yang serentak per alamat IP masih terbatas terlepas dari perangkat lunak yang ada. Jika anda perlu membuat koneksi atas nama beberapa user dari mesin yang sama maka lebih baik menggunakan *site stream*.

2.3 OAuth

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

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

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

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

Twitter OAuth yang diberikan memiliki fitur

- *Secure*

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

- *Standard*

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

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

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

Application-only authentication *Application-only authentication* adalah bentuk dari otentifikasi dimana aplikasi anda membuat *API request* atas nama aplikasi itu sendiri tanpa adanya konteks dari pengguna. Pemanggilan API masih terbatas dalam setiap *API method* .

2.3.1 Application-only authentication

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

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

Tetapi *application-only authentication* tidak bisa melakukan :

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

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

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

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

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

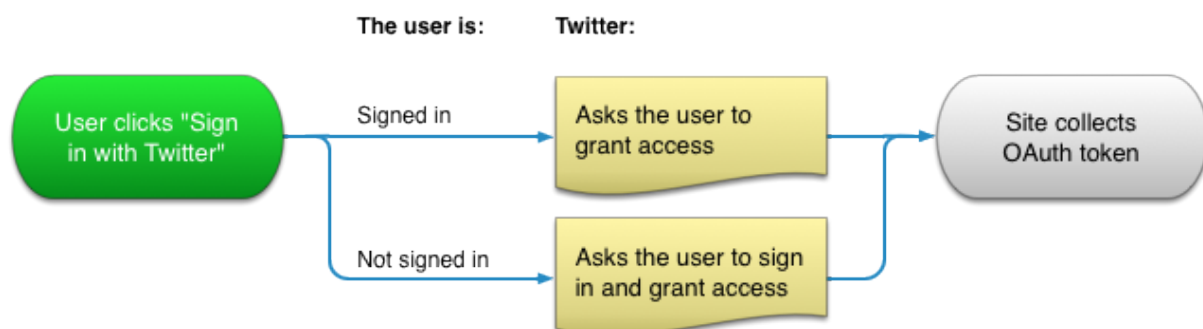
Rate limiting. *Request* yang dibuat atas nama pengguna tidak akan menguras ketersediaan *rate limit* dan *request* tidak akan menguras batas penggunaan **limit** dalam *user-based auth*.

2.3.2 3-legged authorization

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

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

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



Gambar 2.1: Ilustrasi sign in

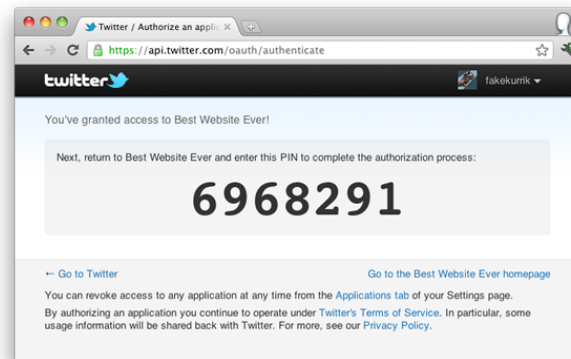
2.3.3 PIN-based authorization

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

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

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

Ketika *callback oob* diminta, akun tidak akan dipindahkan secara otomatis ke aplikasi setelah menyetujui akses. Sebaliknya, mereka akan melihat kode PIN untuk dikembalikan kepada perangkat lunak tersebut dengan cara memasukkan nilai dari kode PIN tersebut.



Gambar 2.2: Contoh PIN-based authorization

Aplikasi anda harus memungkinkan akun untuk memasukkan *PIN code* ini untuk menyelesaikan *flow* tersebut. Nilai dari *PIN code* harus lolos sebagai *oauth_verifier* untuk *POST oauth/access_token request*. Semua *request* akan berjalan normal kedepannya.

2.4 KIRI API

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

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

2.4.1 *Routing Web Service*

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

Berikut ini adalah parameter *request* yang diperlukan:

Listing 2.1: code *respond* pencarian rute

```

1 {
2   "status": "ok" or "error"
3   "routingresults": [
4     {
5       "steps": [
6         {
7           "walk" or "none" or others,
8           "walk" or vehicle_id or "none",
9           ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
10          "human readable description, dependant on locale",
11          URL for ticket booking or null (future)

```

<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 keluaran. Contoh, jika tipe presentasi "mobile", maka link "tel:" akan ditambahkan di hasil.
<i>apikey</i>	16-digit <i>hexadecimals</i>	API key yang digunakan

Tabel 2.7: Parameter *Routing Web Service*

```

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

```

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 menambikan 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

```

1 {
2   "status": "ok" or "error"
3   "searchresult": [
4     {
5       "placename": "place name"
6       "location": "lat,lon"
7     },
8     {
9       "placename": "place name"
10      "location": "lat,lon"
11    },
12    ...
13  ]
14  "attributions": [
15    "attribution_1", "attribution_2", ...
16  ]
17 }
```

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:

<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*

Berikut format kembalian dari Kiri API:

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.
- void clearListeners()
Menghilangkan *status listener*.
- void replaceListener(StreamListener toBeRemoved, StreamListener toBeAdded)
Menimpa listener yang sudah ada sebelumnya.
- void firehose(int count)
Mendengarkan semua status publik.
- void links(int count)
Mendengarkan semua status publik yang mengandung link.
- void retweet()
Mendengarkan semua retweet.
- void sample()
Mendengarkan status publik secara acak.
- void user()
User Streams menyediakan update dari semua data secara *real-time*.
- void user(java.lang.String[] track)
User Streams menyediakan update dari semua data secara *real-time*. Parameter track merupakan kata kunci untuk kata yang akan ditampilkan.
- StreamController site(boolean withFollowings, long[] follow)
Menerima update secara *real-time* untuk sejumlah pengguna tanpa perlu kerepotan dalam mengelola REST API *rate limits*.
- void filter(FilterQuery query)
Menerima status publik yang telah di *filter* dari satu atau lebih kata kunci.
- void cleanUp()
Menon-aktifkan penggunaan *thread stream*.
- void shutdown()
Menon aktifkan *dispatcher thread* bersama dengan semua instansi TwitterStream.

2.5.3 TwitterStreamFactory

- *Constant*

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

- *Constructor*

- TwitterStreamFactory() Membuat TwitterStreamFactory dengan konfigurasi dari sumber.
- TwitterStreamFactory(Configuration conf) Membuat TwitterStreamFactory dengan konfigurasi yang diberikan.
- TwitterStreamFactory(java.lang.String configTreePath) Membuat TwitterStreamFactory yang berasal dari *config tree* yang spesifik.

- *Methods*

- public TwitterStream getInstance()
Mengembalikan contoh yang terkait dengan konfigurasi.
- public TwitterStream getInstance(AccessToken accessToken)
Mengembalikan OAuth yang sudah diotentifikasi.
- public TwitterStream getInstance(Authorization auth)
Mengembalikan *instance*.
- private TwitterStream getInstance(Configuration conf, Authorization auth)
Mengembalikan *instance* dengan konfigurasi dan otorisasi yang sesuai.
- public static Twitter getSingleton()
Mengembalikan *singleton* standar Twitter *instance*.

2.5.4 UserStreamListener

- *Constant*

- public interface UserStreamListener extends StatusListener

- *Methods*

- void onDeleteNotice(long directMessageId, long userId)
- void onFriendList(long[] friendIds)
- void onFavorite(User source, User target, Status favoritedStatus)
- void onUnfavorite(User source, User target, Status unfavoritedStatus)
- void onFollow(User source, User followedUser)

- void onUnfollow(User source, User unfollowedUser)
- void onDirectMessage(DirectMessage directMessage)
- void onUserListMemberAddition(User addedMember, User listOwner, UserList list)
- void onUserListMemberDeletion(User deletedMember, User listOwner, UserList list)
- void onUserListSubscription(User subscriber, User listOwner, UserList list)
- void onUserListUnsubscription(User subscriber, User listOwner, UserList list)
- void onUserListCreation(User listOwner, UserList list)
- void onUserListUpdate(User listOwner, UserList list)
- void onUserListDeletion(User listOwner, UserList list)
- void onUserProfileUpdate(User updatedUser)
- void onBlock(User source, User blockedUser)
- void onUnblock(User source, User unblockedUser)

Tidak ada penjelasan yang diberikan oleh Twitter4J

2.5.5 TweetsResources

- *Constant*

- public interface TweetsResources

- *Methods*

- ResponseList<Status> getRetweets(long statusId) throws TwitterException
Mengembalikan sampai dengan 100 retweet pertama yang diberikan.
- IDs getRetweeterIds(long statusId, long cursor) throws TwitterException
Mengembalikan sampai dengan 100 ID pengguna yang telah melakukan retweet oleh parameter ID tertentu
- IDs getRetweeterIds(long statusId, int count, long cursor) throws TwitterException
Mengembalikan sampai dengan "*count*" ID pengguna yang telah melakukan retweet oleh parameter ID tertentu
- Status showStatus(long id) throws TwitterException
Mengembalikan *single status* yang ditentukan oleh parameter ID yang telah ditentukan.
- Status destroyStatus(long statusId) throws TwitterException
Menghapus status yang ditentukan oleh parameter ID yang telah ditentukan.
- Status updateStatus(java.lang.String status) throws TwitterException
Melakukan update status oleh user yang telah diotentifikasi
- Status updateStatus(StatusUpdate latestStatus) throws TwitterException
Melakukan update status oleh user yang telah diotentifikasi.

- Status retweetStatus(long statusId) throws TwitterException
Melakukan retweet.
- OEmbed getOEmbed(OEmbedRequest req) throws TwitterException Mengembalikan informasi yang dapat merepresentasikan *third party* Tweet
- ResponseList<Status> lookup(long[] ids) throws TwitterException
Mengembalikan *fully-hydrated tweet objects* sampai dengan 100 tweet setiap *requestnya*.
- UploadedMedia uploadMedia(java.io.File mediaFile) throws TwitterException
Melakukan *upload* media gambar yang telah di dilampirkan via updateStatus(twitter4j.StatusUpdate)

2.5.6 OAuthSupport

- *Constant*
 - public interface OAuthSupport
- *Methods*
 - void setOAuthConsumer(java.lang.String consumerKey, java.lang.String consumerSecret)
Melakukan pengaturan terhadap *consumer key* dan *consumer secret*.
 - RequestToken getOAuthRequestToken() throws TwitterException
Mengambil *request token*.
 - RequestToken getOAuthRequestToken(java.lang.String callbackURL) throws TwitterException
Mengambil *request token*.
 - RequestToken getOAuthRequestToken(java.lang.String callbackURL, java.lang.String xAuthAccessType) throws TwitterException
Mengambil *request token*.
 - AccessToken getOAuthAccessToken() throws TwitterException
Mengembalikan *access token* yang terkait dengan instansi ini. Jika tidak ada instansi pada *access token* maka akan mengambil *access token* yang baru.
 - AccessToken getOAuthAccessToken(java.lang.String oauthVerifier) throws TwitterException
Mengambil *request token*.
 - AccessToken getOAuthAccessToken(RequestToken requestToken) throws TwitterException
Mengambil *access token* yang terkait dengan *request token* dan *userId* yang telah diberikan
 - AccessToken getOAuthAccessToken(RequestToken requestToken, java.lang.String oauthVerifier) throws TwitterException
Mengambil *access token* yang terkait dengan *request token* dan *userId* yang telah diberikan

- AccessToken getOAuthAccessToken(java.lang.String screenName, java.lang.String password) throws TwitterException
Mengambil *access token* yang terkait dengan *screen name* dan *password* yang telah diberikan
- void setOAuthAccessToken(AccessToken accessToken)
Melakukan pengaturan pada *access token*

2.5.7 RequestToken

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

2.5.8 AccessToken

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

2.5.9 Status

- *Constant*

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

- *Methods*

- java.util.Date getCreatedAts()
Mengembalikan *created_at*
- public long getUserId()
Mengembalikan *user id*
- java.lang.String getText()
Mengembalikan teks dari status
- java.lang.String getSource()
Mengembalikan *source*
- boolean isTruncated()
Menguji apakah sebuah status terpotong atau tidak
- long getInReplyToStatusId()
Mengembalikan *in_reply_to_status_id*
- long getInReplyToUserId()
Mengembalikan *in_reply_user_id*
- java.lang.String getInReplyToScreenName()
Mengembalikan *in_reply_to_screen_name*
- GeoLocation getGeoLocation()
Mengembalikan lokasi dari suatu *tweet* jika tersedia.
- Place getPlace()
Mengembalikan tempat yang terdapat pada sebuah status.
- boolean isFavorited()
Menguji apakah status tersebut *favorite* atau tidak
- boolean isRetweeted()
Menguji apakah status tersebut *retweet* atau tidak
- int getFavoriteCount()
Menunjukkan berapa kali Tweet telah menjadi *favorite*
- User getUser()
Mengembalikan *user* yang terdapat pada sebuah status.
- boolean isRetweet()
- Status getRetweetedStatus()

- long[] getContributors()
Mengembalikan array yang berisi kontributor atau mengembalikan *null* jika tidak ada kontributor yang terkait dengan status ini
- int getRetweetCount()
Menunjukkan berapa kali Tweet telah di *retweet*, jika belum terdapat maka akan mengembalikan nilai -1
- boolean isRetweetedByMe()
Mengembalikan nilai *true* jika *user* yang telah diotentifikasi melakukan *retweet* terhadap suatu *tweet*, atau mengembalikan nilai *false* jika tidak.
- long getCurrentUserRetweetId()
Mengembalikan *retweet id* sebuah *tweet* dari *user* yang telah diotentifikasi, jika belum terdapat maka akan mengembalikan nilai -1L
- boolean isPossiblySensitive()
Mengembalikan nilai *true* jika pada status terdapat *sensitive links*
- java.lang.String getLang()
Mengembalikan *lang* dari sebuah status teks jika tersedia
- Scopes getScopes()
Mengembalikan target dari *scopes* yang diaplikasikan kepada sebuah status.

2.5.10 TweetsResources

- *Constant*
 - public interface TweetsResources
- *Methods*
 - ResponseList<Status> getRetweets(long statusId) throws TwitterException
Mengembalikan hingga dengan seratus *retweet* pertama
 - IDs getRetweeterIds(long statusId, long cursor) throws TwitterException
Mengembalikan hingga dengan 100 *user ID* yang melakukan *retweet* terhadap *tweet* ditentukan dari *id parameter*
 - IDs getRetweeterIds(long statusId, int count, long cursor) throws TwitterException
Mengembalikan hingga dengan "*count*" *user ID* yang melakukan *retweet* terhadap *tweet* ditentukan dari *id parameter*
 - Status showStatus(long id) throws TwitterException
Mengembalikan *status* yang ditentukan dari parameter *id*.
 - Status destroyStatus(long statusId) throws TwitterException
Menghapus *status* yang ditentukan dari parameter *id*.
 - Status updateStatus(java.lang.String status) throws TwitterException
Melakukan *update status* terhadap *user* yang telah diotentifikasi.

-
- Status `updateStatus(StatusUpdate latestStatus)` throws `TwitterException`
Melakukan *update status* terhadap *user* yang telah diotentifikasi.
 - Status `retweetStatus(long statusId)` throws `TwitterException`
Melakukan *retweet* terhadap sebuah *tweet*.
 - OEmbed `getOEmbed(OEmbedRequest req)` throws `TwitterException`
Mengembalikan informasi yang mengizinkan terciptanya *embedded representation* dari tweet yang berada di *third party sites*
 - `ResponseList<Status> lookup(long[] ids)` throws `TwitterException`
Mengembalikan objek *tweet* hingga dengan 100 *tweet* per **request**.
 - `UploadedMedia uploadMedia(java.io.File mediaFile)` throws `TwitterException`
Melakukan *upload* gambar.

BAB 3

ANALISIS

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

3.1 Analisis Data

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

3.1.1 Analisis Twitter API

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

- Streaming API adalah *real-time* API, sedangkan Search API hanya dapat menangkap *tweet* setiap beberapa waktu sekali. Pada aplikasi yang akan dibuat skenarionya adalah pengguna akan menanyakan rute transportasi publik dalam bentuk *tweet* yang dikirimkan kepada user @kiriupdate, dalam skenario seperti ini dibutuhkanlah jawaban yang *real-time*.
- Menggunakan *Public Stream* dalam *endpoint streaming*. *Public Stream* mengambil semua data publik, sehingga semua *tweet* bisa ditangkap menggunakan *Public Stream*. Dalam pembuatan *Twitter Bot* untuk mencari jalur transportasi publik 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/kiri.travel/handle.php?version=2&mode=searchplace®ion=cgk/bdo/sub&querystring=)

Parameter yang dikirimkan adalah :

1. version : 2

Memberitahukan versi KIRI API, mengikuti versi yang paling baru oleh karena itu penulis akan menuliskan parameter version dengan nilai 2.

2. mode : "searchplace"

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

3. region : bdo

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

4. querystring

Merupakan kata kunci untuk lokasi.

5. apikey : 889C2C8FBB82C7E6

Merupakan *password* yang digunakan untuk mengakses KIRI API.

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

Berikut adalah hasil kembalian dari KIRI API:

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

```

1  {
2      "status":"ok",
3      "searchresult":[
4          {
5              "placename":"J.Co Donuts & Coffee",
6              "location":"-6.88929,107.59574"
7          },
8          {
9              "placename":"Pepper Lunch Bandung (PVJ)",
10             "location":"-6.88923,107.59615"
11         },
12         {
13             "placename":"Domino's Pizza Pvj",
14             "location":"-6.90348,107.61709"
15         },
16         {
17             "placename":"Outlet Alleira Batik PVJ Bandung",
18             "location":"-6.88875,107.59634"
19         },
20         {
21             "placename":"Burger King Bandung PVJ Mall",
22             "location":"-6.88894,107.59342"
23         },
24         {
25             "placename":"Killiney Kopitiam PVJ",
26             "location":"-6.88947,107.59654"
27         },
28         {
29             "placename":"Adidas Pvj",
30             "location":"-6.88909,107.59614"
31         },
32         {
33             "placename":"Crocs - PVJ",
34             "location":"-6.88894,107.59342"
35         },
36         {
37             "placename":"Cross Pvj",
38             "location":"-6.88906,107.59619"
39         },
40         {
41             "placename":"Jonas Photo - PVJ",
42             "location":"-6.88913,107.59643"
43         }
44     ],
45     "attributions": null
46 }
```

- *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 penggunanya menggunakan perangkat bergerak maka parameter dari *presentation* yang cocok adalah "mobile".

7. apikey : 889C2C8FBB82C7E6

Merupakan password yang digunakan untuk mengakses KIRI API.

Penulis mencoba mencari langkah perjalanan dari pvj menuju bip. Layanan dikirimkan ke URL kiri.travel/handle.php. Berikut adalah format layanan yang dituliskan: <http://kiri.travel/handle.php?version=2&mode=findroute&locale=en&start=-6.88923,107.59615&finish=-6.90864,107.61108&presentation=mobile&apikey=889C2C8FBB82C7E6>.

Berikut adalah hasil kembalian dari KIRI API:

Listing 3.2: hasil kembalian dari *Routing Web Service*

```

1 | {
2 |     "status": "ok",
3 |     "routingresults": [
4 |         {
5 |             "steps": [
6 |                 [
7 |                     "walk",
8 |                     "walk",
9 |                     ["-6.88923,107.59615", "-6.88958,107.59691"],

```

```

10         "Walk about 92 meter from your starting point \\\%fromicon to Jalan Sukajadi
11         \\\%toicon.",
12         null
13     ],
14     [
15         "angkot",
16         "kalapakarangsetra",
17         ["-6.88958,107.59691","-6.89052,107.59696","-6.89146,107.59701","-6.89239,107.59706","-6.
18
19         "Take angkot Kalapa – Karang Setra at Jalan Sukajadi \\\%fromicon, and alight
20         at Jalan Pajajaran \\\%toicon about 2.6 kilometer later.",
21         null
22     ],
23     [
24         "angkot",
25         "ciroyomantapani",
26         ["-6.90713,107.60441","-6.90713,107.60441","-6.90679,107.60440","-6.90563,107.60438","-6.
27
28         "Take angkot Ciroyom – Antapani at Jalan Pajajaran \\\%fromicon, and alight at
29         Jalan Aceh \\\%toicon about 1.7 kilometer later.",
30         null
31     ],
32     [
33         "walk",
34         "walk",
35         ["-6.90974,107.61091","-6.90864,107.61108"],
36         "Walk about 124 meter from Jalan Aceh \\\%fromicon to your destination \\\%
37         toicon.",
38         null
39     ],
40     ],
41     "traveltime":"25 minutes"
42 }
43 ]
44 }
```

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

```

1 | {
```

```

2 |         debug=true
3 |         oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIU1
4 |         oauth.consumerSecret=YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****
5 |         oauth.accessToken=313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ
6 |         oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptc****
7 |     }

```

2. via *ConfigurationBuilder*

Menggunakan *ConfigurationBuilder* class untuk melakukan konfigurasi Twitter4J.

Listing 3.4: isi dari twitter4j.properties

```

1 | {
2 |     ConfigurationBuilder cb = new ConfigurationBuilder();
3 |     cb.setDebugEnabled(true)
4 |         .setOAuthConsumerKey("3iT8duMItTTrdaU1qTHxwDIU1")
5 |         .setOAuthConsumerSecret("YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****")
6 |         .setOAuthAccessToken("313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ")
7 |         .setOAuthAccessTokenSecret("LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptc****");
8 |     TwitterFactory tf = new TwitterFactory(cb.build());
9 |     Twitter twitter = tf.getInstance();
10 | }

```

3. via *System Properties*

Menggunakan *System Properties* untuk melakukan konfigurasi Twitter4J.

Listing 3.5: isi dari twitter4j.properties

```

1 | $ export twitter4j.debug=true
2 | $ export twitter4j.oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIU1
3 | $ export twitter4j.oauth.consumerSecret=YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****
4 | $ export twitter4j.oauth.accessToken=313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ
5 | $ export twitter4j.oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptc****
6 | $ java -cp twitter4j-core-4.0.2.jar:yourApp.jar yourpackage.Main

```

3.2 Analisis Perangkat Lunak

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

3.2.1 Spesifikasi Kebutuhan Fungsional

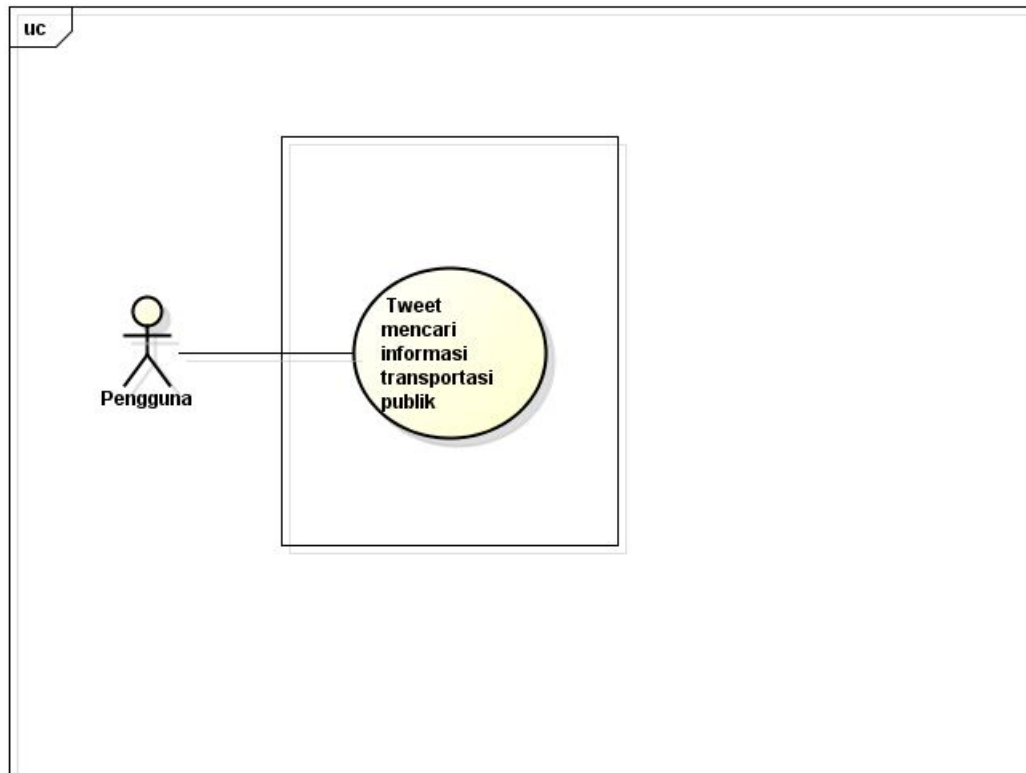
Spesifikasi kebutuhan perangkat lunak yang akan dibangun untuk membuat Twitter Bot adalah

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

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

3.2.2 Use Case Diagram

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



Gambar 3.1: Use case Twitter Bot

Skenario Use Case Skenario ini hanya memiliki satu aktor yaitu pengguna. *Tweet* mencari informasi transportasi publik pada skenario ini dilakukan dengan melakukan *tweet* kepada 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

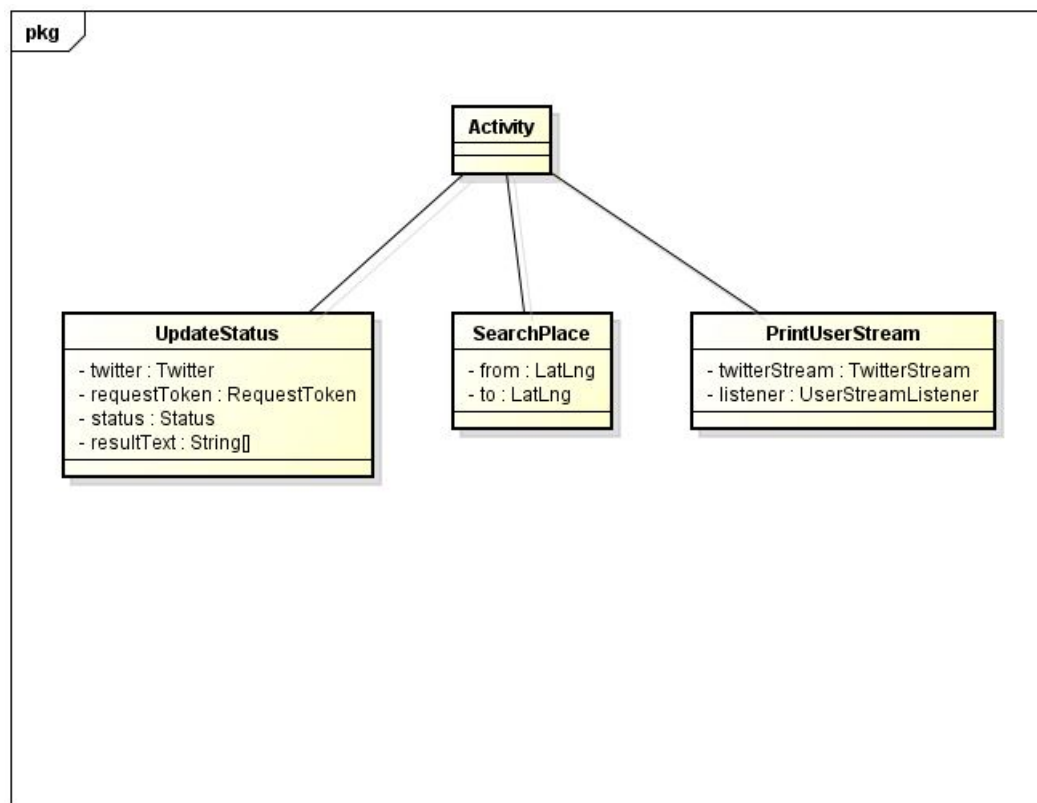
Tabel 3.1: Skenario *Tweet* mencari informasi transportasi

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

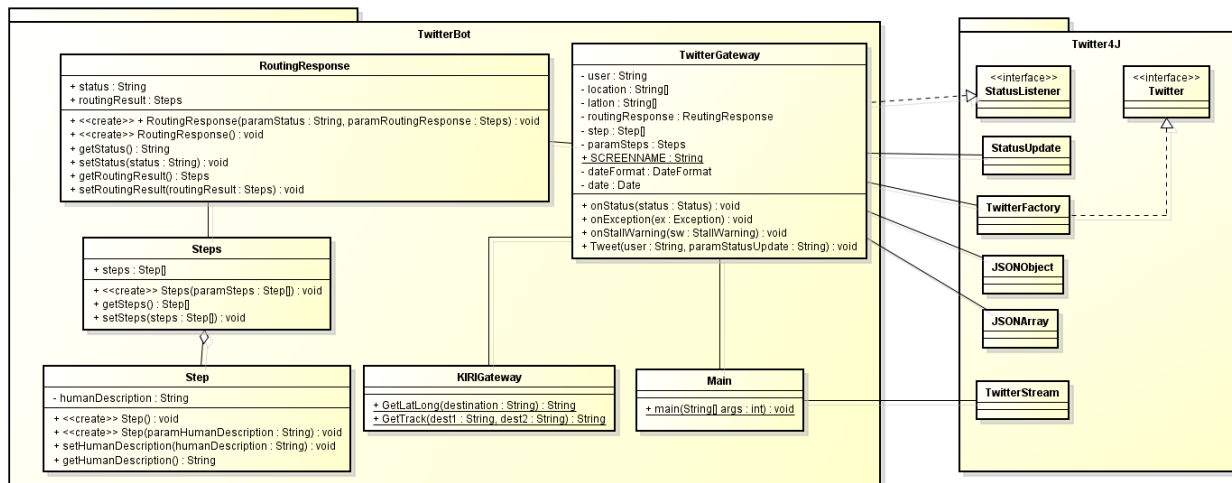
PERANCANGAN PERANGKAT LUNAK

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 perangkat lunak *Twitter bot* untuk mencari jalur transportasi publik. Untuk lebih jelas mengenai kelas yang ada pada aplikasi ini, penulis menyajikan gambar kelas diagram yang dapat dilihat pada gambar 4.1



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

- Kelas Main, merupakan kelas yang berfungsi untuk membuat koneksi dengan Twitter ketika perangkat lunak dijalankan.

– Method

* `public static void main(String[] args)`, merupakan method main untuk menjalankan program.

- Kelas Twitter Gateway, merupakan kelas untuk menangkap dan membalas *tweet*. Kelas Twitter Gateway ini mengimplementasikan *StatusListener*.

– Atribut

- * String user, digunakan untuk menampung nama akun pengguna *Twitter bot*.
- * String location[], berupa *array* yang digunakan untuk menampung lokasi awal dan lokasi tujuan.
- * String latlon[], berupa *array* yang digunakan untuk menampung koordinat lokasi awal dan koordinat lokasi tujuan.
- * RoutingResponse routingResponse, merupakan atribut yang digunakan untuk menampung hasil yang diberikan oleh KIRI API.
- * Step[] step, berupa *array* yang berguna untuk menampung langkah-langkah informasi perjalanan.
- * Steps steps, merupakan atribut yang berguna untuk menampung semua step.

– Method

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

- Kelas KIRIGateway, merupakan kelas untuk memanggil KIRI API. Pemanggilan KIRI API ini digunakan untuk mendapatkan koordinat suatu lokasi dan mencari jalur transportasi publik.
 - Method
 - * public static String GetLatLong(String destination), merupakan *method* yang digunakan untuk mencari koordinat dari suatu lokasi. Hasil kembalian dari *method* ini berupa *latitude* and *longitude* yang diberikan oleh KIRI API lalu diubah ke dalam bentuk *String*.
 - * public static String GetTrack(String dest1, String dest2), merupakan *method* yang digunakan untuk mencari jalur transportasi publik dari lokasi awal ke lokasi tujuan. Hasil kembalian dari *method* ini adalah langkah-langkah perjalanan dari lokasi awal ke lokasi tujuan dengan menggunakan transportasi publik.
- Kelas RoutingResult, merupakan kelas untuk menampung hasil kembalian dari KIRI API
 - Atribut
 - * status, merupakan atribut yang digunakan untuk menyimpan status dari hasil pencarian.
 - * routingResult, merupakan atribut yang digunakan untuk menyimpan langkah-langkah perjalanan.
 - Method
 - * public RoutingResponse(String paramStatus, Steps paramRoutingResult), merupakan *constructor* dari kelas RoutingResult.
 - * public RoutingResponse(), merupakan *constructor* dari kelas RoutingResult.
 - * public String getStatus(), merupakan *getter* dari atribut status.
 - * public void setStatus(String status), merupakan *setter* dari atribut status.
 - * public Steps getRoutingResult(), merupakan *getter* dari atribut routingResult.
 - * public void setRoutingResult(Steps routingResult), merupakan *setter* dari atribut routingResult.
- Kelas Step, merupakan kelas untuk menampung jalur perjalanan dari lokasi awal ke lokasi tujuan dengan menggunakan transportasi publik yang diberikan oleh KIRI API.
 - Atribut
 - * String humanDescription, merupakan atribut untuk menjelaskan cara perjalanan yang bahasanya dimengerti oleh pengguna.
 - Method
 - * public Step(), merupakan *constructor* dari kelas Step.
 - * public Step(String paramHumanDescription), merupakan *constructor* dari kelas Step.
 - * public String getHumanDescription(), merupakan *getter* dari atribut humanDescription.

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

4.1.2 Sequence Diagram

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

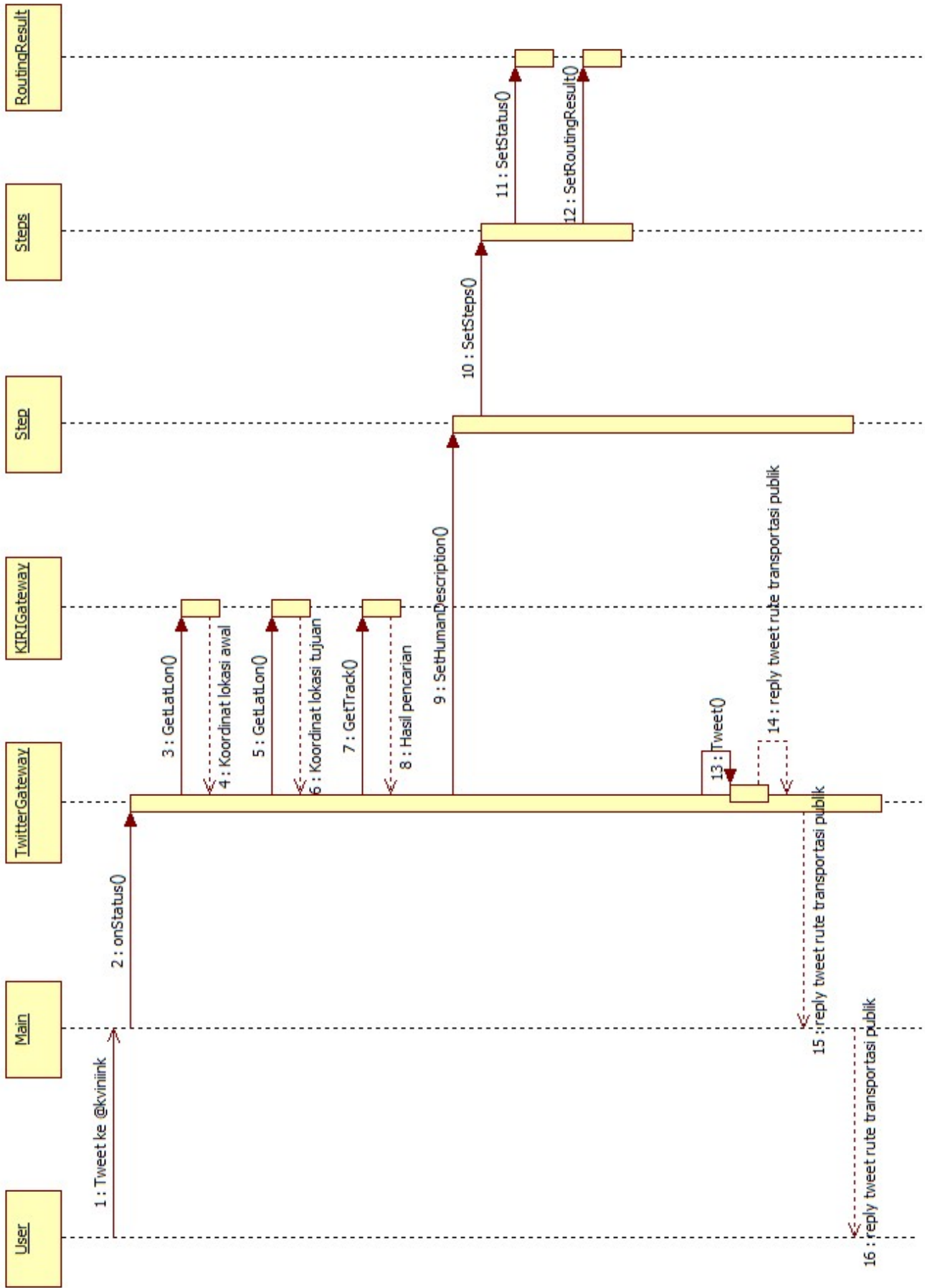
Pertama, program akan melakukan *streaming* pada saat kelas *main* dijalankan. Kelas *main* akan membuka gerbang untuk mengakses *Twitter API*, dengan menggunakan *Streaming API* aplikasi akan menangkap semua *tweet* yang memiliki kata kunci @kviniink. Aplikasi akan terus melakukan *streaming tweet*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

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

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

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

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-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ
 - 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

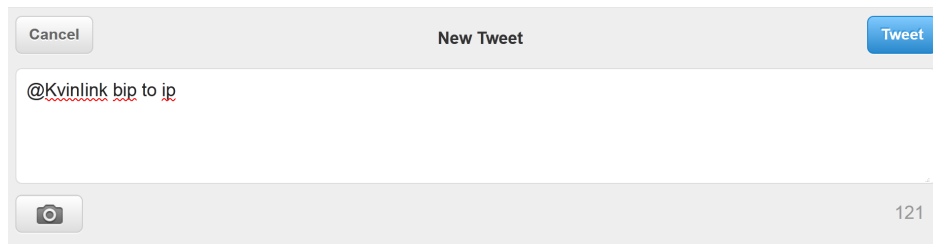
Pengujian fungsional dilakukan pada fungsionalitas yang tersedia pada aplikasi yang dibangun. Pengujian ini dilakukan untuk mengetahui kesesuaian reaksi nyata dengan reaksi yang diharapkan dari aplikasi yang dibangun. Hasil pengujian ditunjukkan pada tabel 5.1.

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 <i>reply</i> kepada akun penguji @kvininktest123, <i>reply</i> tersebut berisikan jalur transportasi publik yang harus ditempuh dari lokasi awal menuju lokasi tujuan.

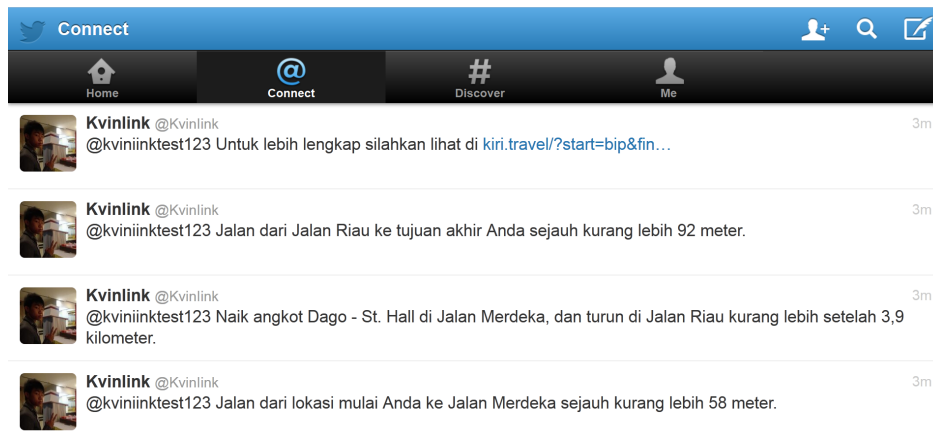
Tabel 5.1: Tabel Hasil pengujian fungsionalitas pada Aplikasi Twitter Bot untuk mencari jalur transportasi publik

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



Gambar 5.1: Tweet dari BIP menuju IP



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

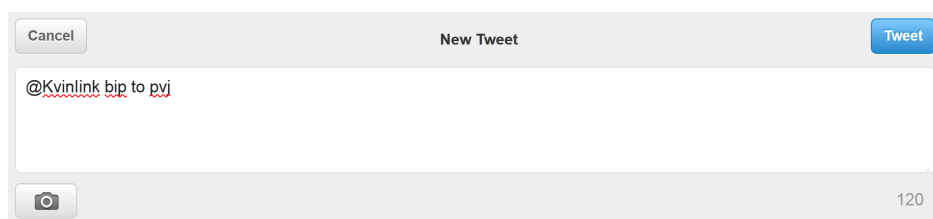
melakukan tweet pencarian melalui akun @kvininktest123.

1. Pengujian 1

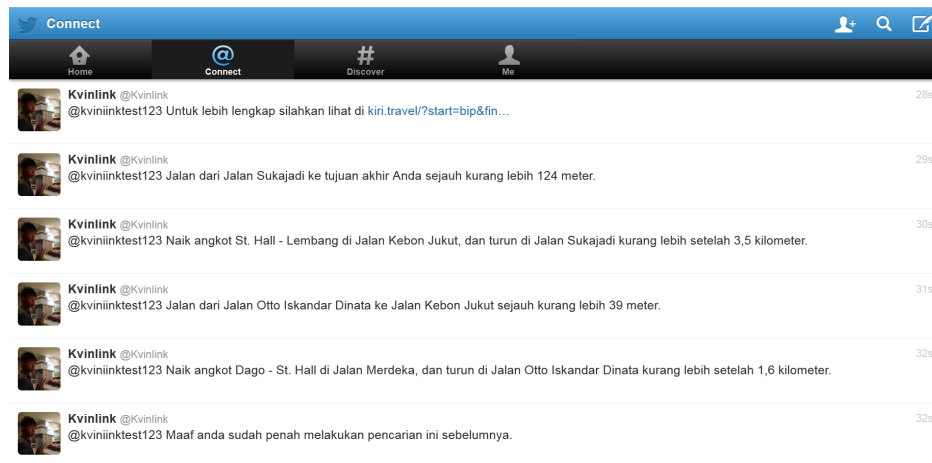
Pada pengujian 1, peneliti mencoba untuk mencari jalur transportasi publik untuk lokasi yang umum dikunjungi yaitu *mall*. Pencarian dilakukan dengan lokasi awal adalah BIP (Bandung Indah Plaza) menuju lokasi tujuan adalah IP (Istana Plaza). Akun penguji @kvininktest123 melakukan tweet kepada Twitter Bot akun @kvinink yang dapat dilihat pada gambar 5.1.

Setelah proses *tweet* dilakukan, Twitter Bot akan menangkap tweet tersebut dan memproses *tweet*. Setelah proses pencarian selesai dilakukan, lalu Twitter Bot akun @kvinink melakukan *reply* kepada akun @kvininktest yang dapat dilihat pada gambar 5.2.

Pencarian ke dua dilakukan pencarian dengan lokasi awal adalah BIP (Bandung Indah Plaza) dan lokasi tujuan ada PVJ (Paris van Java). Dapat dilihat pada gambar 5.3, akun @kvininktest123 melakukan *tweet* pencarian jalur transportasi publik yang di *mention* kepada akun @kvinink dengan lokasi awal adalah BIP dan lokasi tujuan adalah PVJ.



Gambar 5.3: Tweet dari BIP menuju PVJ




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

Setelah itu tweet tersebut proses oleh aplikasi untuk dicari jalur transportasi publiknya, lalu Twitter Bot akun @kviniiink melakukan *reply* kepada akun @kviniiinktest. Reply tweet tersebut merupakan jalur transportasi publik yang harus ditempuh, *reply tweet* tersebut dapat dilihat pada gambar 5.4.

Pada pencarian ke dua dapat dilihat pada tweet pertama terjadi ketidak sesuaian hasil dari KIRI API dengan hasil tweet. Peneliti lalu melakukan pencarian melalui website KIRI yaitu <http://kiri.travel>. Pencarian pertama dilakukan dengan lokasi awal adalah BIP dan lokasi tujuan adalah IP. Hasil pencarian KIRI tersebut dapat dilihat pada gambar 5.5.

Lalu pencarian kedua dilakukan dengan lokasi awal adalah BIP dan lokasi tujuan adalah PVJ. Hasil pencarian KIRI dari BIP menuju PVJ dapat dilihat pada gambar 5.6. Setelah dilihat dari hasil keduanya, Twitter Bot melakukan *duplicate tweet* pada *tweet* pertama pencarian ke dua. Untuk menghindari adanya duplicate tweet, peneliti menaruh waktu untuk jam, menit, dan detik di setiap tweet yang dilakukan oleh Twitter Bot agar membuat setiap tweet tersebut bersifat unik.



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.



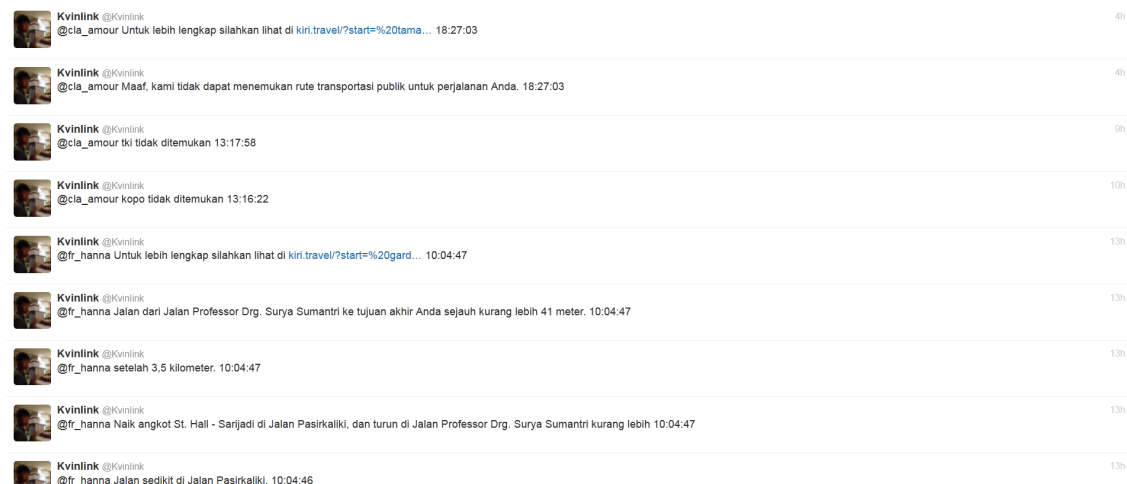
Take angkot St. Hall - Lembang at Jalan Kebon Jukut, and alight at Jalan Sukajadi about 3.5 kilometer later.



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



Gambar 5.7: Hasil Reply Twitter Bot

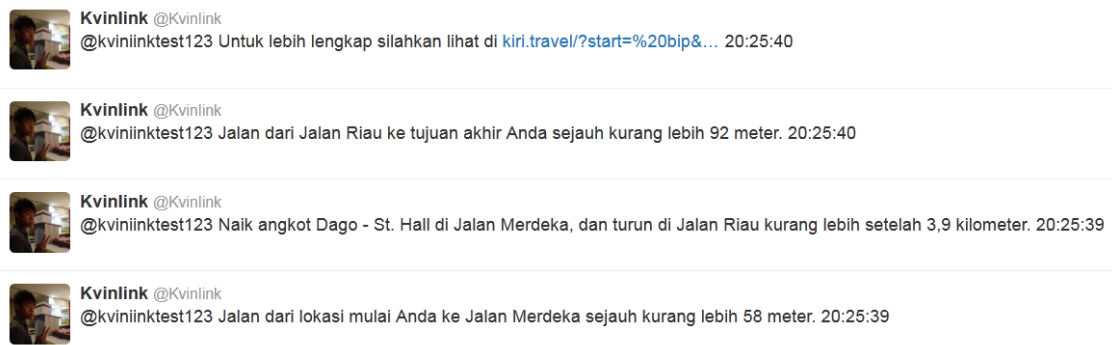


Gambar 5.8: Akun Twitter Bot Mendapat Banyak Mention Dalam Satu Tweet

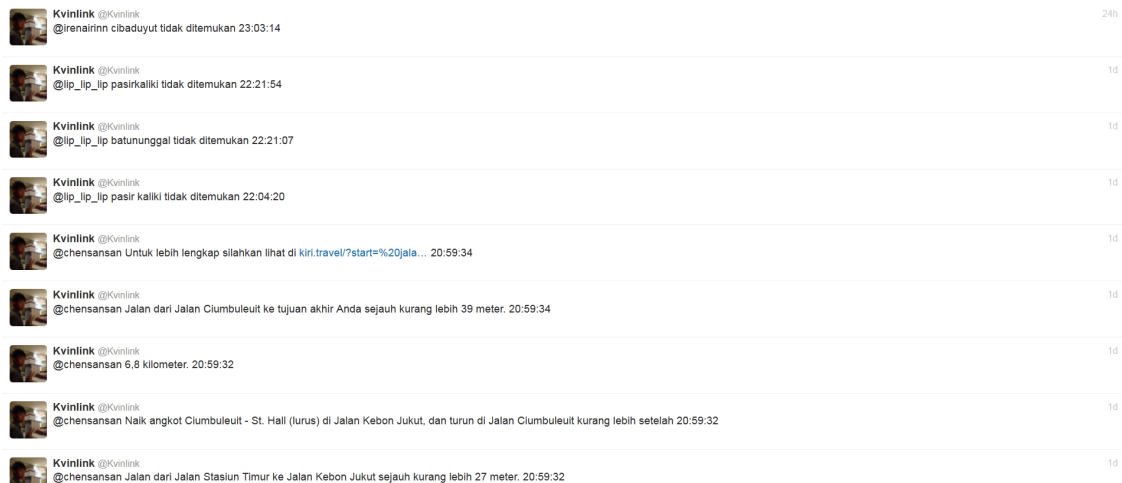
2. Pengujian 2

Pada pengujian 2, peneliti mencoba menjalankan aplikasi selama 24 jam dan meminta bantuan orang lain untuk melakukan *tweet* pencarian jalur transportasi publik. Setelah ditambahkan waktu jam, menit, dan detik pada setiap *tweet*, aplikasi *Twitter bot* berjalan dengan lancar. *Twitter bot* dapat memberitahu bahwa suatu lokasi pencarian tidak ditemukan yang dapat dilihat pada gambar 5.7, akun @cla_amour mencari lokasi tki dan kopo tetapi lokasi pencarian tidak ditemukan. Akun @cla_amour juga mencari jalur transportasi publik yang lokasinya ditemukan tetapi tidak ada rute transportasi publiknya, hasil *reply Twitter bot* dapat dilihat pada salah satu reply yang terdapat pada gambar 5.7. Twitter bot tidak akan mendapatkan error ketika akun Twitter Bot @kvinlink mendapat banyak mention dalam satu tweet seperti pada gambar 5.8, jika format penulisan benar maka Twitter bot akan tetap mencari jalur transportasi publiknya yang dapat dilihat pada gambar 5.9.

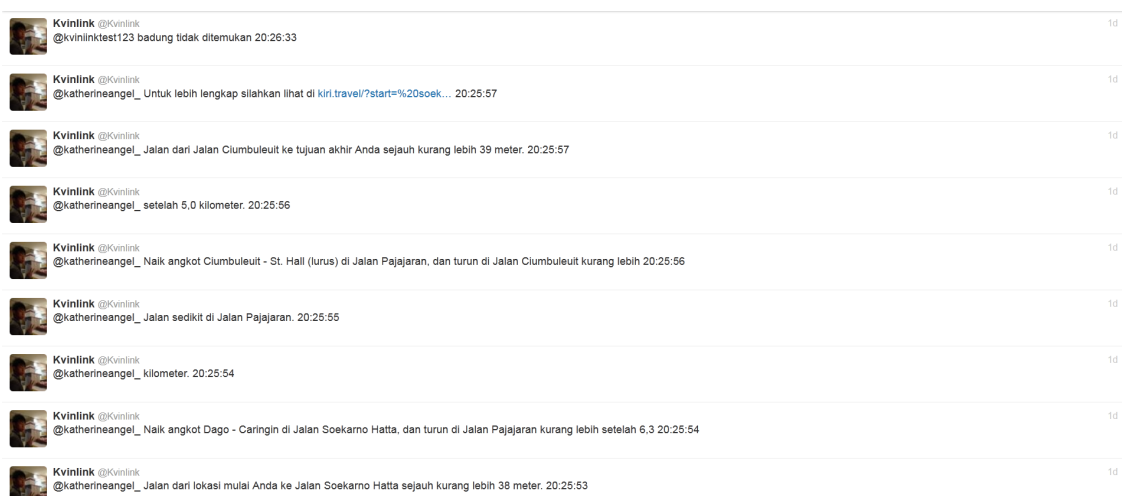
Gambar 5.7, gambar 5.10, dan gambar 5.11 merupakan beberapa hasil reply dari Twitter Bot.



Gambar 5.9: Hasil Reply Twitter Bot



Gambar 5.10: Hasil Reply Twitter Bot



Gambar 5.11: Hasil Reply Twitter Bot

BAB 6

KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dan saran dari penelitian yang dilakukan.

6.1 Kesimpulan

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

1. Pengguna melakukan tweet dengan format lokasi awal *to* lokasi tujuan kepada *Twitter bot*.
2. Twitter bot sudah berjalan dengan lancar dan sudah dapat diimplementasikan kepada user @kiriupdate. Twitter bot dapat menerima tweet yang di mention kepada akun *Twitter bot* dan sudah dapat melakukan reply dengan benar sesuai hasil yang diberikan oleh KIRI API.
3. Akses internet mempengaruhi performansi *Twitter bot*.

6.2 Saran

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

1. Ketika pengguna ingin mencari lokasi jalan, penulisan harus lengkap. Sebagai contoh adalah jalan mekar wangi, jalan kopo. Jika penulisan hanya kopo saja maka pencarian lokasi tidak akan ditemukan.
2. Membuat akun *Twitter bot* menjadi premium agar tidak mengalami adanya keterbatasan tweet per harinya.
3. Membuat format *tweet* baru agar lebih mudah dimengerti pengguna.
4. Pengguna tidak harus melakukan *mention* kepada akun *Twitter Bot*, pengguna dapat memanfaatkan fungsi *hashtag* yang telah diberikan Twitter.
5. Mengatasi format *tweet* agar tidak terlihat seperti *spam*.

DAFTAR REFERENSI

- [1] 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/>