

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

UNDERGRADUATE THESIS

**DEVELOPING TWITTER BOT FOR LOCATING PUBLIC
TRANSPORT ROUTE**



KEVIN THEODORUS YONATHAN

NPM: 2011730037

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2014**

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

DAFTAR ISI

| | |
|--|------------|
| DAFTAR ISI | ix |
| DAFTAR GAMBAR | xi |
| DAFTAR TABEL | xii |
| 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> | 12 |
| 2.3.2 <i>3-legged authorization</i> | 13 |
| 2.3.3 <i>PIN-based authorization</i> | 14 |
| 2.4 KIRI API | 14 |
| 2.4.1 <i>Routing Web Service</i> | 15 |
| 2.4.2 <i>Search Place Web Service</i> | 16 |
| 2.4.3 <i>Nearest Transports Web Service</i> | 17 |
| 2.5 Twitter4J | 18 |
| 2.5.1 Twitter | 18 |
| 2.5.2 TwitterFactory | 19 |
| 2.5.3 TwitterStream | 19 |
| 2.5.4 TwitterStreamFactory | 20 |
| 2.5.5 UserStreamListener | 21 |
| 2.5.6 StatusListener | 22 |
| 2.5.7 StatusUpdate | 22 |
| 2.5.8 TweetsResources | 24 |
| 2.5.9 OAuthSupport | 24 |
| 2.5.10 RequestToken | 25 |
| 2.5.11 AccessToken | 26 |
| 2.5.12 Status | 26 |
| 2.5.13 TweetsResources | 28 |
| 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> | 35 |
| 4 | PERANCANGAN PERANGKAT LUNAK | 37 |
| 4.1 | Perancangan Perangkat Lunak | 37 |
| 4.1.1 | Perancangan Kelas | 37 |
| 4.1.2 | Sequence Diagram | 40 |
| 4.1.3 | Perancangan Antar Muka | 42 |
| 5 | IMPLEMENTASI DAN PENGUJIAN APLIKASI | 43 |
| 5.1 | Lingkungan Pembangunan | 43 |
| 5.2 | Hasil Tampilan Antarmuka | 43 |
| 5.3 | Pengujian | 47 |
| 5.3.1 | Pengujian Fungsional | 47 |
| 5.3.2 | Pengujian Eksperimental | 47 |
| 6 | KESIMPULAN DAN SARAN | 55 |
| 6.1 | Kesimpulan | 55 |
| 6.2 | Saran | 55 |
| | DAFTAR REFERENSI | 57 |
| | A KODE PROGRAM KELAS MAIN | 59 |
| | B KODE PROGRAM KELAS KIRIGATEWAY | 61 |
| | C KODE PROGRAM KELAS KIRIGATEWAY | 65 |
| | D KODE PROGRAM KELAS ROUTINGRESPONSE | 67 |
| | E KODE PROGRAM KELAS STEP | 69 |
| | F KODE PROGRAM KELAS STEPS | 71 |

DAFTAR GAMBAR

| | | |
|------|--|----|
| 2.1 | Contoh PIN-based authorization | 14 |
| 3.1 | Use case Twitter Bot | 35 |
| 3.2 | <i>Class Diagram</i> Twitter Bot | 36 |
| 4.1 | Class Diagram Pembuatan <i>Twitter bot</i> untuk Mencari Jalur Transportasi Publik | 37 |
| 4.2 | Sequence Diagram <i>Twitter bot</i> untuk Mencari Jalur Transportasi Publik | 41 |
| 4.3 | Homepage Twitter yang Diakses Melalui Mobile Twitter | 42 |
| 5.1 | Antarmuka Perangkat Lunak Twitter Bot Untuk Mencari Jalur Transportasi Publik | 44 |
| 5.2 | Antar muka Twitter yang diakses melalui aplikasi Twitter di Android | 44 |
| 5.3 | Antar muka Twitter yang diakses melalui aplikasi Twitter di iOS | 45 |
| 5.4 | Antar muka Twitter yang diakses melalui aplikasi Twitter di Windows Phone | 46 |
| 5.5 | Antar muka Twitter yang diakses melalui aplikasi Twitter di Website Twitter | 47 |
| 5.6 | Tweet dari BIP menuju IP | 48 |
| 5.7 | Hasil Pencarian Rute Transportasi Publik dari BIP menuju IP | 48 |
| 5.8 | Tweet dari BIP menuju PVJ | 48 |
| 5.9 | Hasil Pencarian Rute Transportasi Publik dari BIP menuju PVJ | 49 |
| 5.10 | Hasil Pencarian Jalur Transportasi Publik dari BIP menuju IP Melalui Website KIRI | 49 |
| 5.11 | Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI | 50 |
| 5.12 | Hasil Reply Twitter Bot | 52 |
| 5.13 | Akun Twitter Bot Mendapat Banyak Mention Dalam Satu Tweet | 52 |
| 5.14 | Hasil Reply Twitter Bot | 52 |
| 5.15 | Hasil Reply Twitter Bot | 53 |
| 5.16 | Hasil Reply Twitter Bot | 53 |

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> | 15 |
| 2.8 | Tabel parameter <i>Search Place Web Service</i> | 16 |
| 2.9 | Tabel parameter <i>Nearest Transports Web Service</i> | 17 |
| 3.1 | Skenario <i>Tweet</i> mencari informasi transportasi | 36 |
| 5.1 | Tabel Hasil pengujian fungsionalitas pada Aplikasi <i>Twitter bot</i> untuk mencari jalur transportasi publik | 51 |

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 mengatasi kemacetan dan mengurangi pemanasan global. Seiring dengan

¹Kominfo bint005 , Pengguna Internet di Indonesia Capai 82 Juta, http://kominfo.go.id/index.php/content/detail/3980/Kominfo%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

perkembangan teknologi, menaiki transportasi publik menjadi semakin mudah. Dengan adanya KIRI di Indonesia terutama di Kota Bandung, masyarakat dapat menaiki 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:

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

5 1.4 Batasan Masalah

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

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

13 1.5 Metode Penelitian

14 Pada perangkat lunak yang dibuat ini digunakan beberapa metode dalam penyelesaian ma-
15 salah yang menjadi topik pada penelitian ini, antara lain:

- 16 1. Melakukan studi literatur, antara lain:
 - 17 • KIRI API,
 - 18 • REST API Twitter (<https://dev.twitter.com/docs/api/1.1>),
 - 19 • Streaming API Twitter (<https://dev.twitter.com/docs/api/streaming>).
- 20 2. Mempelajari pembuatan server dalam bahasa Java.
- 21 3. Membuat *Twitter bot* sederhana.
- 22 4. Melakukan analisis terhadap teori-teori yang sudah dipelajari, guna membangun per-
23 angkat lunak yang dimaksud.
- 24 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 (*Uniform resource locator*) *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* Uniform resource locator

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 Bandung, maka akun @kviniink membuat

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

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

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" [2].

- *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(*Application programming interface*) adalah aplikasi pihak ketiga yang memungkinkan pengembang perangkat lunak melakukan manipulasi dan pengolahan data di Twitter. Twitter API adalah salah satu bentuk pendekatan dari Twitter yang berfokus pada jaringan dan memungkinkan pengembang perangkat lunak memiliki hak untuk berpikir '*out of the box*' untuk membuat aplikasi yang mereka inginkan[1]. 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[1]. Ini berarti bahwa ada beberapa *Tweet* atau akun akan hilang dari hasil pencarian.

1 **Bagaimana cara membuat sebuah *query*** Cara terbaik dalam membuat sebuah *qu-*
 2 *ery* adalah melakukan percobaan yang valid dan mengembalikan *tweet* yang sesuai. Cara
 3 mencobanya dapat dilakukan pada twitter.com/search. URL yang ditampilkan pada bro-
 4 wser akan berisi sintaks *query* yang sesuai agar dapat digunakan kembali pada Twitter API.
 5 Berikut adalah contohnya:

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

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

| Operator | Finds <i>tweets</i> |
|-----------------------------------|--|
| <i>watching now</i> | Mengandung kata " <i>watching</i> " dan " <i>now</i> ". |
| <i>"happy hour"</i> | Mengandung frase " <i>happy hour</i> " yang tepat. |
| <i>love OR hate</i> | Mengandung kata " <i>love</i> " atau " <i>hate</i> " atau keduanya. |
| <i>beer -root</i> | Mengandung kata " <i>beer</i> " tanpa adanya kata " <i>root</i> ". |
| <i>#haiku</i> | Mengandung <i>hashtag</i> " <i>haiku</i> ". |
| <i>from:alexiskold</i> | Dikirim melalui akun "alexiskold". |
| <i>to:techcrunch</i> | Dikirimkan kepada akun "techcrunch". |
| <i>@mashable</i> | Mereferensi kepada akun "mashable". |
| <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 perta-nyaan. |
| <i>hilarious filter:links</i> | Mengandung kata " <i>hilarious</i> " yang di sambungkan de-ngan URL. |
| <i>news source:twitterfeed</i> | Mengandung kata " <i>news</i> " yang di- <i>posting</i> melalui <i>twit-terfeed</i> . |

Tabel 2.1: Contoh berbagai macam pencarian *tweet*

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

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

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

1 **Additional parameters** Terdapat parameter tambahan yang dapat digunakan untuk
 2 menghasilkan pencarian yang lebih baik. Berikut adalah penjelasan dari parameter tam-
 3 bahan tersebut :

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

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

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

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

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

- 32 • Anda ingin mencari *tweet* yang paling baru dalam Bahasa Portugal, yang lokasinya
 33 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%3Akvinink%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*[1].

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

- *Public stream*

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

- *User Stream*

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

- *Site Stream*

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

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

Endpoints

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

1 **POST statuses/filter** POST *filter* dapat mengembalikan status publik yang sesuai de-
 2 ngan satu atau lebih predikat yang telah difilter. *Multiple parameter* memungkinkan klien
 3 untuk menggunakan koneksi tunggal untuk ke *Streaming* API. Antara GET *request* dan
 4 POST *request* keduanya didukung oleh POST statuses / *filter* tetapi untuk GET *request*
 5 yang memiliki parameter yang terlalu banyak mungkin akan ditolak karena URL yang ter-
 6 lalu panjang. Gunakanlah POST request untuk menghindari URL yang panjang. *Track*,
 7 *follow*, dan lokasi harus dipertimbangkan untuk dapat digabungkan dengan operator OR.
 8 *track=foo&follow=1234* ini mengembalikan *tweet* yang memiliki kata "foo" atau dibuat oleh
 9 *user* 1234. Akses standar mengizinkan pencarian hingga 400 kata kunci, dan 5000 *follow*
 10 *userids*. Perintah ini dikembalikan dalam format JSON, memerlukan otentikasi *user context*,
 11 dan frekuensi pemakaiannya dibatasi. Parameter untuk POST statuses/*filter* dapat dilihat
 12 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*

13 **GET statuses/sample** Mengembalikan *random* sampel dari semua status publik. Jika
 14 terdapat dua *client* yang terhubung dengan *endpoint* ini, maka kedua *client* tersebut akan
 15 melihat *tweet* yang sama. Perintah ini dikembalikan dalam format JSON, memerlukan oten-
 16 tikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter GET statuses/*sample*
 17 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*

18 **GET statuses/firehose** Mengembalikan semua status publik. Beberapa aplikasi mem-
 19 butuhan akses ini. Teknik ini diolah secara kreatif dengan cara menggabungkan sumber
 20 informasi yang ada dengan berbagai sumber lainnya untuk dapat memuaskan pengguna.
 21 Perintah ini dikembalikan dalam format JSON, memerlukan otentikasi *user context*, dan fre-
 22 kuensi pemakaiannya dibatasi. Parameter GET statuses/*firehose* dapat dilihat pada tabel
 23 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*

24 **Menggunakan Streaming API** Proses menggunakan *streaming* API adalah dengan cara
 25 menghubungkan *endpoint* yang sudah tercantum di atas dengan parameter yang sudah di-*list*

1 kepada *streaming endpoint* dan juga *request* parameter *streaming* API.

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

7 **User Streams** *User Stream* memberikan aliran(*stream*) data dan event yang spesifik un-
8 tuk akun yang sudah diotentikasi. Perintah ini dikembalikan dalam format JSON, me-
9 merlukan otentikasi *user context*, dan frekuensi pemakaiannya dibatasi. Parameter untuk
10 parameter ini dapat dilihat pada tabel 2.6

11 Endpoints

12 • GET *user*

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

Tabel 2.6: Parameter GET *user*

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

20 Sebuah aplikasi harus dapat mengatasi HTTP(*The Hypertext Transfer Protocol*) 420
21 *error code* yang memberitahukan bahwa suatu akun sudah terlalu sering melakukan *login*.
22 Oleh karena itu, akun yang seperti itu akan secara otomatis di-*banned* dari *user stream*
23 untuk tingkat *login* yang berlebihan. Perhatikan bahwa setiap perangkat lunak memiliki
24 alokasinya masing-masing, sehingga *login* dari perangkat lunak yang pertama tidak akan

1 mempengaruhi koneksi untuk perangkat lunak yang kedua, begitu juga sebaliknya. Tetapi
 2 akan menimbulkan masalah apabila menjalankan terlalu banyak salinan perangkat lunak
 3 yang pertama maupun kedua. Jika anda perlu membuat koneksi atas nama beberapa akun
 4 dari perangkat lunak yang sama, maka akan lebih baik jika menggunakan *site stream*.

5 2.3 OAuth

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

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

17 Agar *client* dapat mengakses *resource* mereka, pertama-tama ia harus mendapatkan izin
 18 dari si pemilik *resource*. Izin ini dinyatakan dalam bentuk token dan juga digunakan untuk
 19 mencocokkan *shared-secret*. Tujuan dari token ini adalah untuk membuat pemilik *resource*
 20 berbagi kepercayaan mereka kepada *client*. Token dapat dikeluarkan dalam ruang lingkup
 21 terbatas, durasi yang terbatas, dan akan dicabut secara independen³.

22 **Twitter OAuth** yang diberikan memiliki fitur :

- 23 • *Secure*

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

- 26 • *Standard*

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

28 2.3.1 Application-only authentication

29 Twitter menawarkan aplikasi yang mampu mengeluarkan permintaan otentikasi atas nama
 30 aplikasi itu sendiri. Dengan menggunakan *application-only authentication*, perangkat lunak
 31 tidak mempunyai konteks dari otentikasi pengguna dan ini berarti setiap *request* API untuk
 32 endpoint akan membutuhkan konteks pengguna, seperti memposting *tweet* tidak akan be-
 33 kerja. *Application-only authentication* dapat melakukan berbagai macam aktivitas, seperti
 34 :

- 35 • melihat *timeline*,

- 36 • mengakses *following* dan *follower* dari suatu *akun*,

³Hueniverse Documentation , OAuth, <http://hueniverse.com/oauth/guide/intro/>, pada tanggal 20 Agustus 2014 pukul 12.58

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

3 Tetapi *application-only authentication* tidak dapat melakukan :

- 4 • Posting *tweet*
- 5 • Melakukan koneksi dengan *Streaming endpoint*
- 6 • Mencari akun seseorang
- 7 • Menggunakan *geo endpoint*
- 8 • Mengakses *Direct Message*

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

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

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

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

26 2.3.2 3-legged authorization

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

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

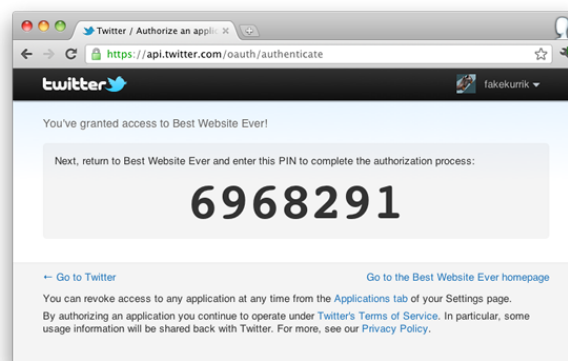
2.3.3 PIN-based authorization

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

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

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

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



Gambar 2.1: Contoh PIN-based authorization

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

2.4 KIRI API

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

- *Routing Web Service*
- *Search Place Web Service*

- *Nearest Transports Web Service*

2.4.1 *Routing Web Service*

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

Berikut ini adalah parameter *request* yang diperlukan:

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

Tabel 2.7: Parameter *Routing Web Service*

Listing 2.1: kode respon pencarian rute

```

6 {
7   "status": "ok" or "error"
8   "routingresults": [
9     {
10      "steps": [
11        [
12          "walk" or "none" or others ,
13          "walk" or vehicle_id or "none",
14          ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
15          "human readable description, dependant on locale",
16          URL for ticket booking or null (future)
17        ],
18        [
19          "walk" or "none" or others ,
20          "walk" or vehicle_id or "none",
21          ["lat_1,lon_1", "lan_2,lon_2", ... "lat_n,lon_n"],
22          "human readable description, dependant on locale",
23          URL for ticket booking or null (future)
24        ]
25      ],
26      "travelttime": any text string, null if and only if route is not found.
27    } ,
28    {
29      "steps": [ ... ],
30      "travelttime": "... "
31    } ,
32    {
33      "steps": [ ... ],
34      "travelttime": "... "
35    } ,
36    ...
37  ]
38 }

```

Listing 2.1 menunjukkan hasil yang akan diberikan dari pencarian rute. Ketika pencarian rute berhasil, maka status yang diberikan akan bernilai "ok" seperti pada baris 2. Kemudian server harus memberikan hasil dari rute yang berisi langkah-langkah yang disimpan di dalam *array*. Berikut ini adalah keterangan dari *array* tersebut:

- 1 • *Index* 0 berisikan "*walk*" atau "*none*" atau "*others*". "*Walk*" berarti jalan kaki, "*no-*"
2 *ne*" berarti rute jalan tidak ditemukan, dan "*others*" berarti menggunakan kendaraan.
- 3 • *Index* ke 1 merupakan *detail* dari *index* ke 0 yang memiliki arti:
 - 4 – Jika berisikan "*walk*" berarti *index* ini pun harus berisikan "*walk*",
 - 5 – Jika berisikan "*none*" maka *index* ini pun harus berisikan "*none*",
 - 6 – Selain itu, maka *field* ini berisikan id kendaraan yang dapat digunakan untuk
7 menampilkan gambar dari id kendaraan tersebut.
- 8 • *Index* ke 2 berisikan *array of string*, yang berisikan jalur dalam format "*lat,lon*". *Lat*
9 adalah *latitude*, dan *lon* adalah *longitude* yaitu titik awal dan titik akhir.
- 10 • *Index* ke 3 merupakan bentuk yang dapat dibaca oleh manusia lalu akan ditampilkan
11 kepada pengguna. Informasi tersebut dapat berupa:
 - 12 – %*fromicon* = sebuah *icon* penanda yang menunjukkan titik awal atau "*from*".
13 Biasanya digunakan untuk mode presentasi perangkat bergerak.
 - 14 – %*toicon* = sebuah *icon* penanda yang menunjukkan titik akhir atau "*to*". Bia-
15 sanya digunakan untuk mode presentasi perangkat bergerak.
- 16 • *Index* ke 4 berisi URL untuk pemesanan tiket untuk travel jika tersedia. Jika tidak
17 ada maka nilai dari *index* ini bernilai null.

18 2.4.2 Search Place Web Service

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

| | | |
|--------------------|--|---|
| <i>version</i> | 2 | Memberitahukan bahwa layanan yang dipakai adalah protokol versi 2 |
| <i>mode</i> | " <i>searchplace</i> " | menginstruksikan layanan untuk mencari tempat |
| <i>region</i> | " <i>cgk</i> " or " <i>bdo</i> " or " <i>sub</i> " | 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*

23 Berikut format kembalian dari KIRI API:

Listing 2.2: code *respond* pencarian lokasi

```

24 {
25   "status": "ok" or "error"
26   "searchresult": [
27     {
28       "placename": "place name"
29       "location": "lat,lon"
30     },
31     {
32       "placename": "place name"
33       "location": "lat,lon"
34     },
35     ...

```

```

1  |   ]
2  |   "attributions": [
3  |     "attribution_1", "attribution_2", ...
4  |   ]
5  | }

```

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

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

2.4.3 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 versi 2 |
| <i>mode</i> | "nearbytransports" | menginstruksikan 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

```

18 | {
19 |   "status": "ok" or "error"
20 |   "nearbytransports": [
21 |     [
22 |       "walk" or "none" or others ,
23 |       "walk" or vehicle_id or "none",
24 |       text string ,
25 |       decimal value
26 |     ],
27 |     [
28 |       "walk" or "none" or others ,
29 |       "walk" or vehicle_id or "none",
30 |       text string ,
31 |       decimal value
32 |     ],
33 |     ...
34 |   ]
35 | }

```

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

- *Index* ke 0 dapat berisi "walk" atau "none" atau "others". Artinya jika isi dari *array* tersebut "walk" berarti berjalan kaki, "none" jika rute tidak ditemukan dan "others" berarti menggunakan kendaraan.

- 1 • *Index* ke 1 merupakan detail dari *index* 0. Artinya jika *index* 0 "*walk*" berarti *index*
- 2 1 harus "*walk*", "*none*" berarti *index* 1 harus "*none*" dan selain itu menyatakan id
- 3 kendaraan yang mana bisa dipakai untuk ditampilkan gambarnya.
- 4 • *Index* ke 2 berisi nama kendaraan yang dapat dibaca oleh pengguna.
- 5 • *Index* ke 3 berisi jarak dalam satuan kilometer.

6 2.5 Twitter4J

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

- 10 • 100% menggunakan Bahasa Java.
- 11 • Tersedia untuk *Android platform* dan *Google App Engine*.
- 12 • Tidak adanya dependensi, tidak memerlukan *jar* tambahan.
- 13 • Mendukung sistem OAuth.
- 14 • Kompatibel dengan Twitter API 1.1

15 Dalam pembuatan aplikasi yang akan penulis buat, penulis membutuhkan beberapa *li-*
 16 *brary* yang telah diberikan oleh Twitter4J. Berikut adalah *library* yang diberikan Twitter4J
 17 :

18 2.5.1 Twitter

- 19 • *Constant*
 - 20 – public interface Twitter extends java.io.Serializable, OAuthSupport, OAuth2Support,
 - 21 TwitterBase, TimelinesResources, TweetsResources, SearchResource, DirectMes-
 - 22 sagesResources, FriendsFollowersResources, UsersResources, SuggestedUsersRe-
 - 23 sources, FavoritesResources, ListsResources, SavedSearchesResources, PlacesGe-
 - 24 oResources, TrendsResources, SpamReportingResource, HelpResources
- 25 • *Methods*
 - 26 – TimelinesResources timelines()
 - 27 – TweetsResources tweets()
 - 28 – SearchResource search()
 - 29 – DirectMessagesResources directMessages()
 - 30 – FriendsFollowersResources friendsFollowers()
 - 31 – UsersResources users()
 - 32 – SuggestedUsersResources suggestedUsers()
 - 33 – FavoritesResources favorites()

- 1 – ListsResources list()
- 2 – SavedSearchesResources savedSearches()
- 3 – PlacesGeoResources placesGeo()
- 4 – TrendsResources trends()
- 5 – SpamReportingResource spamReporting()
- 6 – HelpResources help()

7 *Tidak ada penjelasan yang diberikan oleh Twitter4J*

8 2.5.2 TwitterFactory

9 • *Constant*

- 10 – public final class TwitterFactory extends java.lang.Object implements java.io.Serializable
- 11 Sebuah *factory class* untuk Twitter

12 • *Constructor*

- 13 – TwitterFactory()
- 14 Membuat TwitterFactory dengan konfigurasi dari sumber.
- 15 – TwitterFactory(Configuration conf)
- 16 Membuat TwitterFactory dengan konfigurasi yang diberikan.
- 17 – TwitterFactory(java.lang.String configTreePath)
- 18 Membuat TwitterFactory yang berasal dari *config tree* yang spesifik.

19 • *Methods*

- 20 – public Twitter getInstance()
- 21 mengembalikan contoh yang terkait dengan konfigurasi.
- 22 – public Twitter getInstance(AccessToken accessToken)
- 23 mengembalikan OAuth yang sudah otentikasi.
- 24 – public Twitter getInstance(Authorization auth)
- 25 – public static Twitter getSingleton()
- 26 Mengembalikan *singleton* standar Twitter *instance*.

27 2.5.3 TwitterStream

28 • *Constant*

- 29 – public interface TwitterStream extends OAuthSupport, TwitterBase
- 30 Sebuah *factory class* untuk Twitter

31 • *Methods*

- 32 – void addConnectionLifeCycleListener(ConnectionLifeCycleListener listener)
- 33 Menambahkan *ConnectionLifeCycleListener*

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

31 2.5.4 TwitterStreamFactory

32 • *Constant*

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

37 • *Constructor*

- 1 – `TwitterStreamFactory()` Membuat `TwitterStreamFactory` dengan konfigurasi dari
- 2 sumber.
- 3 – `TwitterStreamFactory(Configuration conf)` Membuat `TwitterStreamFactory` de-
- 4 ngan konfigurasi yang diberikan.
- 5 – `TwitterStreamFactory(java.lang.String configTreePath)` Membuat `TwitterStrea-`
- 6 mFactory yang berasal dari *config tree* yang spesifik.

7 • *Methods*

- 8 – `public TwitterStream getInstance()`
- 9 Mengembalikan contoh yang terkait dengan konfigurasi.
- 10 – `public TwitterStream getInstance(AccessToken accessToken)`
- 11 Mengembalikan OAuth yang sudah diotentikasi.
- 12 – `public TwitterStream getInstance(Authorization auth)`
- 13 Mengembalikan *instance*.
- 14 – `private TwitterStream getInstance(Configuration conf, Authorization auth)`
- 15 Mengembalikan *instance* dengan konfigurasi dan otorisasi yang sesuai.
- 16 – `public static Twitter getSingleton()`
- 17 Mengembalikan *singleton* standar Twitter *instance*.

18 2.5.5 UserStreamListener

19 • *Constant*

- 20 – `public interface UserStreamListener extends StatusListener`

21 • *Methods*

- 22 – `void onDeleteNotice(long directMessageId, long userId)`
- 23 – `void onFriendList(long[] friendIds)`
- 24 – `void onFavorite(User source, User target, Status favoritedStatus)`
- 25 – `void onUnfavorite(User source, User target, Status unfavoritedStatus)`
- 26 – `void onFollow(User source, User followedUser)`
- 27 – `void onUnfollow(User source, User unfollowedUser)`
- 28 – `void onDirectMessage(DirectMessage directMessage)`
- 29 – `void onUserListMemberAddition(User addedMember, User listOwner, UserList`
- 30 `list)`
- 31 – `void onUserListMemberDeletion(User deletedMember, User listOwner, UserList`
- 32 `list)`
- 33 – `void onUserListSubscription(User subscriber, User listOwner, UserList list)`
- 34 – `void onUserListUnsubscription(User subscriber, User listOwner, UserList list)`
- 35 – `void onUserListCreation(User listOwner, UserList list)`

- 1 – void onUserListUpdate(User listOwner, UserList list)
- 2 – void onUserListDeletion(User listOwner, UserList list)
- 3 – void onUserProfileUpdate(User updatedUser)
- 4 – void onBlock(User source, User blockedUser)
- 5 – void onUnblock(User source, User unblockedUser)

6 *Tidak ada penjelasan yang diberikan oleh Twitter4J*

7 2.5.6 StatusListener

8 • *Constant*

- 9 – public interface StatusListener extends StreamListener

10 • *Methods*

- 11 – void onStatus(Status status)
- 12 – void onDeleteNotice(StatusDeletionNotice statusDeletionNotice)
- 13 Method untuk memberitahukan notifikasi deletionNotice.
- 14 – void onTrackLimitationNotice(int numberOfLimitedStatuses)
- 15 Method untuk memberitahukan bahwa predikat terlalu luas.
- 16 – void onScrubGeo(long userId, long upToStatusId)
- 17 Method untuk memberitahukan *location deletion*.
- 18 – void onStallWarning(StallWarning warning)
- 19 Method untuk memberitahukan pesan *warning*.
- 20 –

21 2.5.7 StatusUpdate

22 • *Constant*

- 23 – public final class StatusUpdate extends java.lang.Object implements java.io.Serializable

24 • *Field*

- 25 – private boolean displayCoordinates
- 26 – private long inReplyToStatusId
- 27 – private GeoLocation location
- 28 – private java.io.InputStream mediaBody
- 29 – private java.io.File mediaFile
- 30 – private long[] mediaIds
- 31 – private java.lang.String mediaName
- 32 – private java.lang.String placeId

```

1      – private boolean possiblySensitive
2      – private static long serialVersionUID
3      – private java.lang.String status
4
4      • Methods
5
5      – private void appendParameter(java.lang.String name, double value, java.util.List<HttpParameter>
6          params)
7      – private void appendParameter(java.lang.String name, long value, java.util.List<HttpParameter>
8          params)
9      – private void appendParameter(java.lang.String name, java.lang.String value, java.
10         util.List<HttpParameter> params)
11      – (package private) HttpParameter[] asHttpParameterArray()
12      – StatusUpdate displayCoordinates(boolean displayCoordinates)
13      – boolean equals(java.lang.Object o)
14      – long getInReplyToStatusId()
15      – GeoLocation getLocation()
16      – java.lang.String getPlaceId()
17      – java.lang.String getStatus()
18      – int hashCode()
19      – StatusUpdate inReplyToStatusId(long inReplyToStatusId)
20      – boolean isDisplayCoordinates()
21      – (package private) boolean isForUpdateWithMedia()
22      – boolean isPossiblySensitive()
23      – StatusUpdate location(GeoLocation location)
24      – StatusUpdate media(java.io.File file)
25      – StatusUpdate media(java.lang.String name, java.io.InputStream body)
26      – StatusUpdate placeId(java.lang.String placeId)
27      – StatusUpdate possiblySensitive(boolean possiblySensitive)
28      – void setDisplayCoordinates(boolean displayCoordinates)
29      – void setInReplyToStatusId(long inReplyToStatusId)
30      – void setLocation(GeoLocation location)
31      – void setMedia(java.io.File file)
32      – void setMedia(java.lang.String name, java.io.InputStream body)
33      – void setMediaIds(long[] mediaIds)
34      – void setPlaceId(java.lang.String placeId)
35      – void setPossiblySensitive(boolean possiblySensitive)
36      – java.lang.String toString()

```

37 *Tidak ada penjelasan yang diberikan oleh Twitter4J*

1 2.5.8 TweetsResources

2 • *Constant*

3 – public interface TweetsResources

4 • *Methods*

5 – ResponseList<Status> getRetweets(long statusId) throws TwitterException

6 Mengembalikan sampai dengan 100 retweet pertama yang diberikan.

7 – IDs getRetweeterIds(long statusId, long cursor) throws TwitterException

8 Mengembalikan sampai dengan 100 ID pengguna yang telah melakukan retweet
9 oleh parameter ID tertentu

10 – IDs getRetweeterIds(long statusId, int count, long cursor) throws TwitterExcep-
11 tion

12 Mengembalikan sampai dengan "*count*" ID pengguna yang telah melakukan re-
13 tweet oleh parameter ID tertentu

14 – Status showStatus(long id) throws TwitterException

15 Mengembalikan *single status* yang ditentukan oleh parameter ID yang telah di-
16 tentukan.

17 – Status destroyStatus(long statusId) throws TwitterException

18 Menghapus status yang ditentukan oleh parameter ID yang telah ditentukan.

19 – Status updateStatus(java.lang.String status) throws TwitterException

20 Melakukan update status oleh user yang telah diotentikasi

21 – Status updateStatus(StatusUpdate latestStatus) throws TwitterException

22 Melakukan update status oleh user yang telah diotentikasi.

23 – Status retweetStatus(long statusId) throws TwitterException

24 Melakukan retweet.

25 – OEmbed getOEmbed(OEmbedRequest req) throws TwitterException Mengem-
26 balikan informasi yang dapat merepresentasikan *third party* Tweet

27 – ResponseList<Status> lookup(long[] ids) throws TwitterException

28 Mengembalikan *fully-hydrated tweet objects* sampai dengan 100 tweet setiap *re-*
29 *questnya*.

30 – UploadedMedia uploadMedia(java.io.File mediaFile) throws TwitterException

31 Melakukan *upload* media gambar yang telah di dilampirkan via updateStatus(twitter4j.StatusUpdate)

32 2.5.9 OAuthSupport

33 • *Constant*

34 – public interface OAuthSupport

35 • *Methods*

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

32 2.5.10 RequestToken

33 • *Constant*

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

35 • *Constructor*

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

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

3 • *Methods*

4 – public java.lang.String getAuthorizationURL()

5 – public java.lang.String getAuthenticationURL()

6 **2.5.11 AccessToken**

7 • *Constant*

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

9 • *Constructor*

10 – AccessToken(HttpResponse res)

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

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

13 • *Methods*

14 – public java.lang.String getScreenName()

15 Mengembalikan *screen name*

16 – public long getUserId()

17 Mengembalikan *user id*

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

19 – public int hashCode()

20 – public java.lang.String toString()

21 **2.5.12 Status**

22 • *Constant*

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

25 • *Methods*

26 – java.util.Date getCreatedAts()

27 Mengembalikan *created_at*

28 – public long getUserId()

29 Mengembalikan *user id*

30 – java.lang.String getText()

31 Mengembalikan teks dari status

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

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

5 2.5.13 TweetsResources

6 • *Constant*

- 7 – public interface TweetsResources

8 • *Methods*

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

BAB 3

ANALISIS

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

3.1 Analisis Data

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

3.1.1 Analisis Twitter API

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

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

3.1.2 Analisis OAuth

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

3.1.3 Analisis KIRI API

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

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

Berikut adalah contoh pemanfaatan KIRI API :

- *Search Place Web Service*

Format *Search Place Web Service* yang dikirim melalui URL adalah [kiri.travel/handle.php?version=2&mode=searchplace®ion=cgk/bdo/sub&querystring=\"string\"&apikey=889C2C8FBB82C7E6](https://dev.kiri.travel/handle.php?version=2&mode=searchplace®ion=cgk/bdo/sub&querystring=\).

Parameter yang dikirimkan adalah :

1. version : 2

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

2. mode : "searchplace"

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

3. region : bdo

Region berfungsi sebagai parameter untuk memberitahukan kota yang akan menjadi bagian dalam pencarian lokasi. Parameter yang terdapat di region ada tiga

1 yaitu "cgk" untuk Kota Jakarta, "bdo" untuk Kota Bandung, dan "sub" untuk
2 Kota Surabaya.

3 4. querystring

4 Merupakan kata kunci untuk lokasi.

5 5. apikey : 889C2C8FBB82C7E6

6 Merupakan *password* yang digunakan untuk mengakses KIRI API.

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

11 Berikut adalah hasil kembalian dari KIRI API:

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

```

12       {
13           "status": "ok",
14           "searchresult": [
15               {
16                  "placename": "J.Co Donuts & Coffee",
17                  "location": "-6.88929,107.59574"
18               },
19               {
20                  "placename": "Pepper Lunch Bandung (PVJ)",
21                  "location": "-6.88923,107.59615"
22               },
23               {
24                  "placename": "Domino's Pizza Pvj",
25                  "location": "-6.90348,107.61709"
26               },
27               {
28                  "placename": "Outlet Alleira Batik PVJ Bandung",
29                  "location": "-6.88875,107.59634"
30               },
31               {
32                  "placename": "Burger King Bandung PVJ Mall",
33                  "location": "-6.88894,107.59342"
34               },
35               {
36                  "placename": "Killiney Kopitiam PVJ",
37                  "location": "-6.88947,107.59654"
38               },
39               {
40                  "placename": "Adidas Pvj",
41                  "location": "-6.88909,107.59614"
42               },
43               {
44                  "placename": "Crocs - PVJ",
45                  "location": "-6.88894,107.59342"
46               },
47               {
48                  "placename": "Cross Pvj",
49                  "location": "-6.88906,107.59619"
50               },
51               {
52                  "placename": "Jonas Photo - PVJ",
53                  "location": "-6.88913,107.59643"
54               }
55           ],
56           "attributions": null
57       }

```

58 • *Routing Web Service*

59 Format *Routing Web Service* yang dikirim melalui URL adalah [kiri.travel/handle.php?version=2&mode=findroute&locale=en/id&start=lat,lng&finish=lat,lng&presentation=](http://kiri.travel/handle.php?version=2&mode=findroute&locale=en/id&start=lat,lng&finish=lat,lng&presentation=mobile/desktop&apikey=889C2C8FBB82C7E6)
60 [mobile/desktop&apikey=889C2C8FBB82C7E6](http://kiri.travel/handle.php?version=2&mode=findroute&locale=en/id&start=lat,lng&finish=lat,lng&presentation=mobile/desktop&apikey=889C2C8FBB82C7E6).
61 Parameter yang dikirimkan adalah :

62 Parameter yang dikirimkan adalah :

1. version : 2

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

2. mode : "findroute"

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

3. locale : id

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

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

7. apikey : 889C2C8FBB82C7E6

Merupakan password yang digunakan untuk mengakses KIRI API.

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

Berikut adalah hasil kembalian dari KIRI API:

Listing 3.2: hasil kembalian dari *Routing Web Service*

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

```

1      " -6.89466,107.59719"," -6.89598,107.59727"," -6.89598,107.59727",
2      " -6.89700,107.59731"," -6.89801,107.59735"," -6.89903,107.59740",
3      " -6.90005,107.59744"," -6.90005,107.59744"," -6.90113,107.59747",
4      " -6.90222,107.59751"," -6.90331,107.59754"," -6.90439,107.59757",
5      " -6.90439,107.59757"," -6.90540,107.59760"," -6.90641,107.59763",
6      " -6.90641,107.59763"," -6.90650,107.59781"," -6.90667,107.59887",
7      " -6.90684,107.59992"," -6.90684,107.59992"," -6.90690,107.60086",
8      " -6.90696,107.60179"," -6.90696,107.60179"," -6.90704,107.60306",
9      " -6.90711,107.60433"],
10     "Take angkot Kalapa - Karang Setra at Jalan Sukajadi, and alight at
11     Jalan Pajajaran about 2.6 kilometer later.",
12     null
13   ],
14   [
15     "angkot",
16     "ciroyomantapani",
17     [" -6.90713,107.60441"," -6.90713,107.60441"," -6.90679,107.60440",
18     " -6.90563,107.60438"," -6.90448,107.60435"," -6.90448,107.60435",
19     " -6.90429,107.60448"," -6.90422,107.60487"," -6.90403,107.60527",
20     " -6.90397,107.60564"," -6.90402,107.60608"," -6.90436,107.60671",
21     " -6.90488,107.60725"," -6.90522,107.60749"," -6.90588,107.60771",
22     " -6.90625,107.60772"," -6.90642,107.60783"," -6.90658,107.60806",
23     " -6.90678,107.60929"," -6.90678,107.60929"," -6.90685,107.60939",
24     " -6.90787,107.60939"," -6.90889,107.60939"," -6.90889,107.60939",
25     " -6.90913,107.60920"," -6.90918,107.60878"," -6.90924,107.60847",
26     " -6.90934,107.60843"," -6.91008,107.60880"," -6.91026,107.60890",
27     " -6.91030,107.60905"," -6.91029,107.60923"," -6.91020,107.60951",
28     " -6.90976,107.61056"," -6.90976,107.61056"," -6.90974,107.61091"],
29     "Take angkot Ciroyom - Antapani at Jalan Pajajaran, and alight at
30     Jalan Aceh about 1.7 kilometer later.",
31     null
32   ],
33   [
34     "walk",
35     "walk",
36     [" -6.90974,107.61091"," -6.90864,107.61108"],
37     "Walk about 124 meter from Jalan Aceh to your destination.",
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 :

- Twitter
- StatusListener
- StatusUpdate
- TwitterFactory
- TwitterStream

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. Penggunaan twitter4j.properties 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=false
3          oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIU1
4          oauth.consumerSecret=YUlgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7t313
5          oauth.accessToken=313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ
6          oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYeFMoEdYospYoOptcuR4i
7      }

```

2. via *ConfigurationBuilder*

Menggunakan *ConfigurationBuilder* class untuk melakukan konfigurasi Twitter4J.

Listing 3.4: isi dari twitter4j.properties

```

10     {
11         ConfigurationBuilder cb = new ConfigurationBuilder();
12         cb.setDebugEnabled(true)
13             .setOAuthConsumerKey("3iT8duMItTTrdaU1qTHxwDIU1")
14             .setOAuthConsumerSecret("YUlgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****")
15             .setOAuthAccessToken("313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ")
16             .setOAuthAccessTokenSecret("LVfDgtlfeht5yjBJGSgvSvtMYeFMoEdYospYoOptc****");
17         TwitterFactory tf = new TwitterFactory(cb.build());
18         Twitter twitter = tf.getInstance();
19     }

```

3. via *System Properties*

Menggunakan *System Properties* untuk melakukan konfigurasi Twitter4J.

Listing 3.5: isi dari twitter4j.properties

```

22     $ export twitter4j.debug=true
23     $ export twitter4j.oauth.consumerKey=3iT8duMItTTrdaU1qTHxwDIU1
24     $ export twitter4j.oauth.consumerSecret=
25         YUlgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7****
26     $ export twitter4j.oauth.accessToken=313287708-
27         NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBIZ
28     $ export twitter4j.oauth.accessTokenSecret=LVfDgtlfeht5yjBJGSgvSvtMYeFMoEdYospYoOptc
29         ****
30     $ java -cp twitter4j-core-4.0.2.jar:yourApp.jar yourpackage.Main

```

3.2 Analisis Perangkat Lunak

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

3.2.1 Spesifikasi Kebutuhan Fungsional

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

1. Dapat melakukan otentikasi untuk akun *Twitter bot* yang digunakan.
2. Dapat menerima dan membaca *tweet* yang di *mention* kepada akun *Twitter bot* @kviniink
3. Dapat melakukan proses pencarian koordinat suatu lokasi

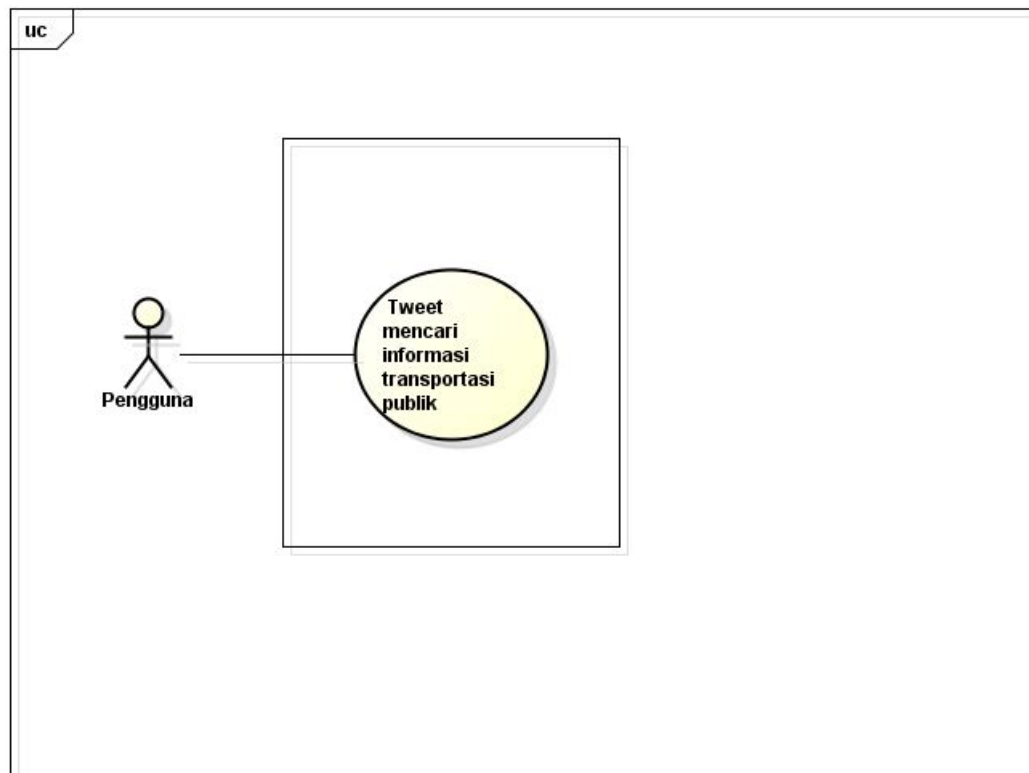
- 1 4. Dapat melakukan proses pencarian jalur transportasi publik dari lokasi awal menuju
- 2 lokasi tujuan
- 3 5. Dapat membalas *tweet* pencarian jalur transportasi publik yang diterima oleh *Twitter*
- 4 *bot* dengan melakukan *reply tweet* yang berisi hasil pencarian jalur transportasi publik
- 5 dengan format yang sudah ditentukan.

6 3.2.2 Use Case Diagram

7 Perangkat lunak yang dibangun akan memiliki satu figur utama, yaitu *tweet* mencari in-

8 formasi transportasi publik. Gambar 3.1 menunjukkan diagram *use case* dari perangkat

9 lunak.



Gambar 3.1: Use case Twitter Bot

10 **Skenario Use Case** Skenario ini hanya memiliki satu aktor yaitu pengguna. *Tweet*

11 mencari informasi transportasi publik pada skenario ini dilakukan dengan melakukan *tweet*

12 kepada akun *Twitter bot* @kviniink berisi format yang sesuai untuk pencarian rute trans-

13 portasi.

14 3.2.3 Class Diagram

15 Untuk membuat *class diagram* *Twitter bot* untuk mencari jalur transportasi publik, dibu-

16 tuhkan kebutuhan kelas dari skenario. Pada tabel skenario 3.1, masukan akan terjadi hal-hal

17 seperti dibawah ini:

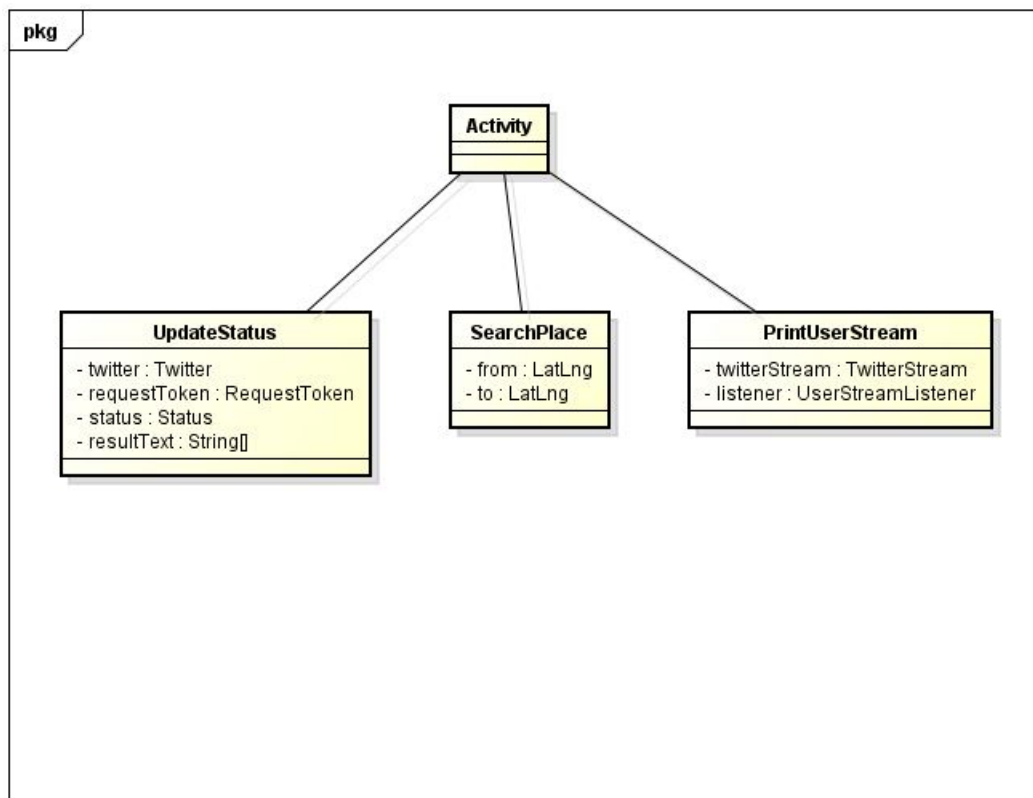
- 18 1. *Twitter bot* akan berjalan terus hingga *Twitter bot* di non-aktifkan.

| | |
|----------------|--|
| 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 <i>user</i> @kiriupdate dengan format yang sudah ditentukan |
| Eksepsi | Format penulisan salah |

Tabel 3.1: Skenario *Tweet* mencari informasi transportasi

- 1 2. Pengguna melakukan *tweet* mencari informasi transportasi dengan cara melakukan
- 2 *mention* kepada akun *Twitter bot* @kviniink dengan format yang sesuai dengan keten-
- 3 tuan.
- 4 3. *Twitter bot* menerima *mention* dari pengguna.
- 5 4. *Twitter bot* akan mencari jalur transportasi publik.
- 6 5. *Twitter bot* melakukan *reply* kepada pengguna yang berisi jalur transportasi publik
- 7 yang harus ditempuh.

Berikut adalah *class diagram* sederhana:

Gambar 3.2: *Class Diagram* Twitter Bot

BAB 4

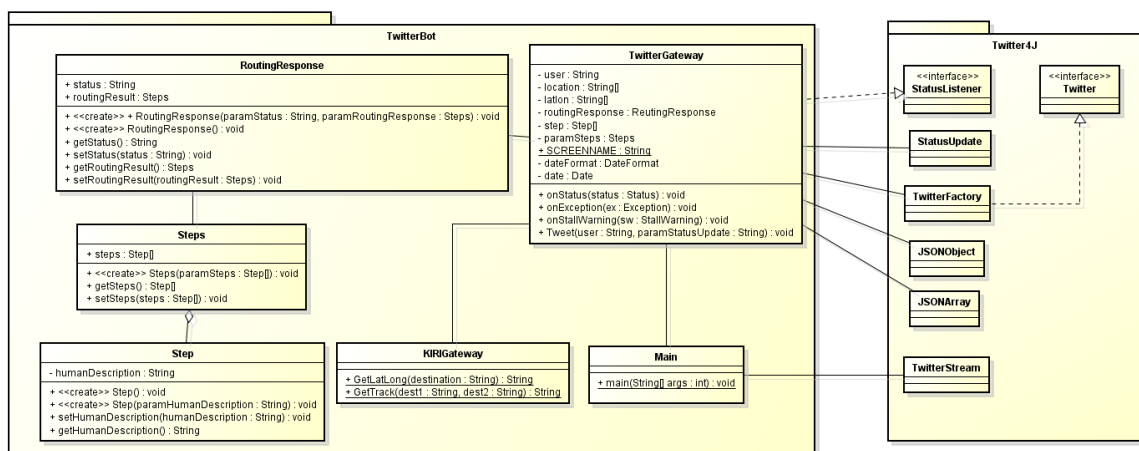
PERANCANGAN PERANGKAT LUNAK

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

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

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

- 1 * public void setHumanDescription(String humanDescription), merupakan *set-*
- 2 *ter* dari atribut humanDescription.
- 3 • Kelas Steps, merupakan kelas untuk menampung kumpulan step.
- 4 – Atribut
- 5 * Step[] steps, merupakan atribut yang berisi *array* step
- 6 – Method
- 7 * public Steps(Step[] paramSteps), merupakan konstruktor dari kelas Steps.
- 8 * public Step[] getSteps(), merupakan *getter* dari atribut steps.
- 9 * public void setSteps(Step[] steps), merupakan *setter* dari atribut steps.

10 4.1.2 Sequence Diagram

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

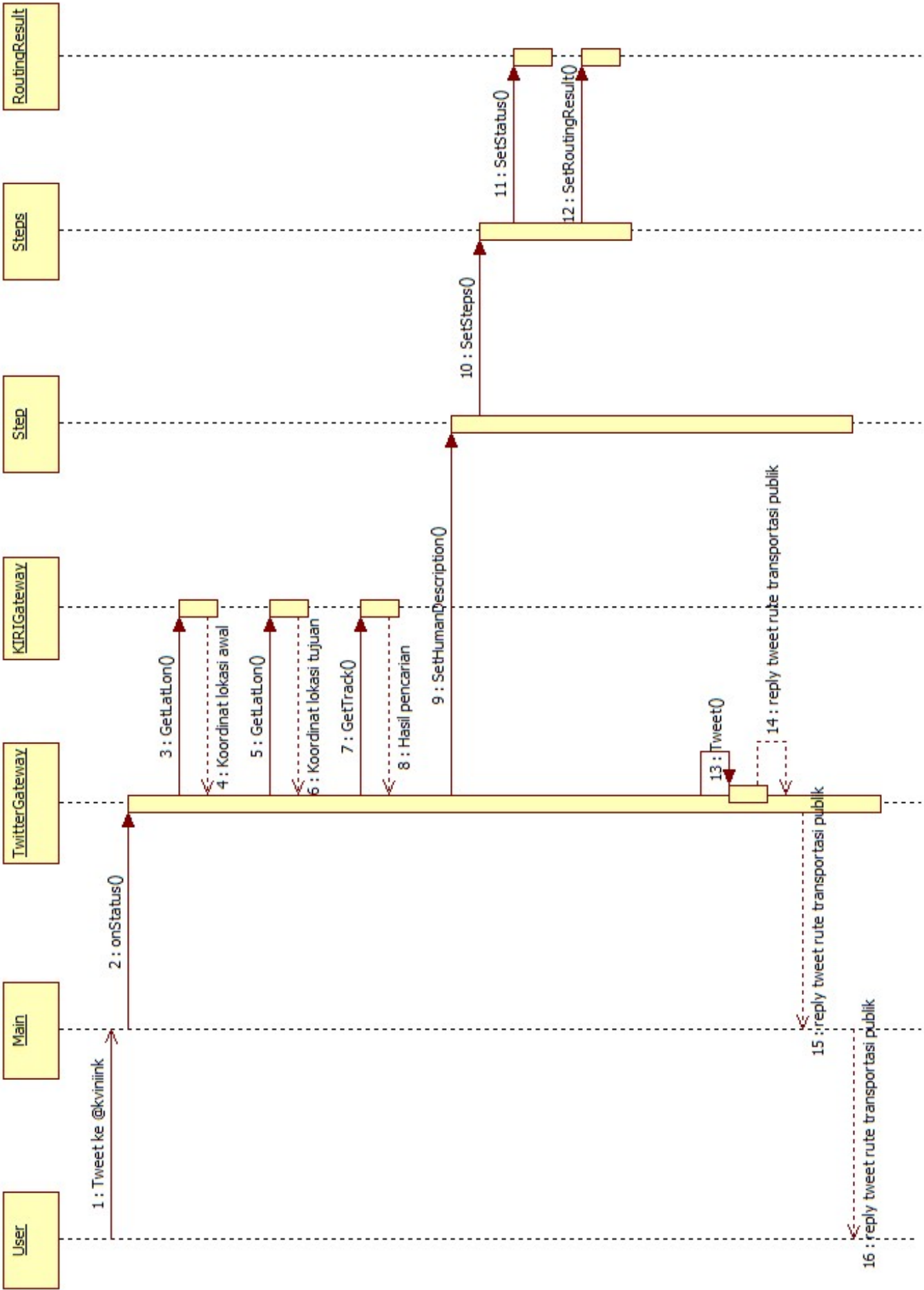
13 Pertama, perangkat lunak akan melakukan *streaming* pada saat kelas *main* dijalankan.
 14 Kelas *main* akan membuka gerbang untuk mengakses *Twitter API*, dengan menggunakan
 15 *Streaming API* perangkat lunak akan menangkap semua *tweet* yang melakukan *mention*
 16 kepada akun *Twitter bot @kviniink*.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

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

24 Kelas *KIRIGateway* akan memanggil *GetLatLon method* untuk mencari koordinat suatu
 25 lokasi. Setelah didapatkan koordinat lokasi awal dan lokasi tujuan, kelas *TwitterGateway*
 26 akan mengubah hasil dari *GetLatLon method* yang berupa *JSON* menjadi format *String*.
 27 Setelah didapatkan koordinat lokasi awal dan koordinat lokasi tujuan maka hasil dari masing-
 28 masing koordinat dikembalikan kepada kelas *KIRIGateway* untuk dicari jalur transportasi
 29 publik dari lokasi awal menuju lokasi tujuan menggunakan *GetTrack method*. Hasil dari
 30 *GetTrack method* akan disimpan pada atribut *step*, *steps*, dan *routingResult*.

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

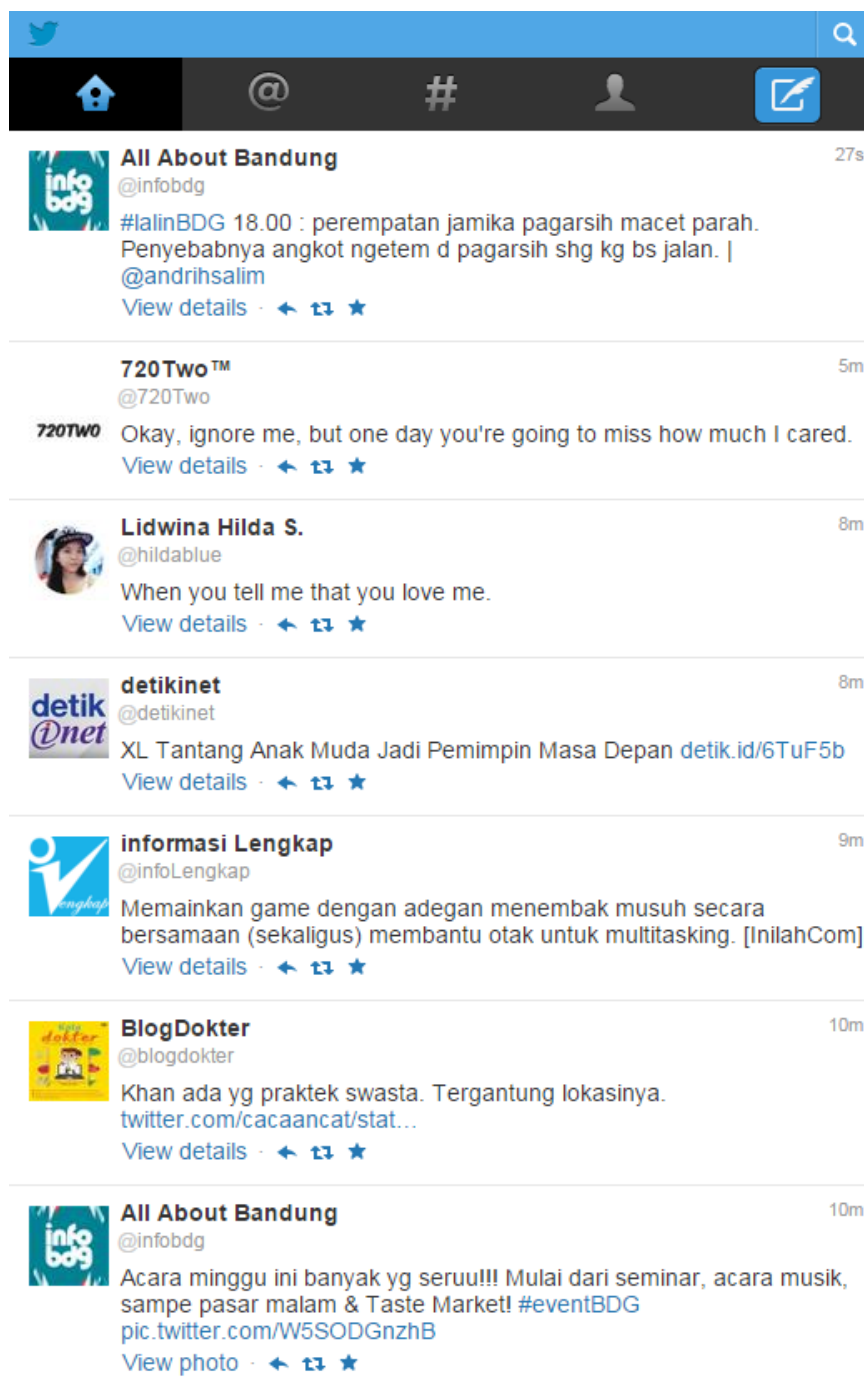
Diagram Final.jpg



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

4.1.3 Perancangan Antar Muka

- Perangkat lunak yang akan dibangun memiliki antar muka berbasis teks, tetapi tampilan antar muka untuk pengguna berbeda-beda. Interaksi dengan pengguna dapat dilakukan melalui *website* atau aplikasi Twitter. Gambar 4.3 adalah tampilan antar muka dari Twitter yang diakses melalui *website mobile* Twitter <https://mobile.twitter.com/>. Antar muka *Twitter bot* akan menampilkan *tweet* yang diterima oleh perangkat lunak, dan menampilkan hasil *tweet* yang di-reply *Twitter bot* kepada pengguna.



Mobile Twitter.PNG

Gambar 4.3: Homepage Twitter yang Diakses Melalui Mobile Twitter

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 : YUIgJTbQT3i5tYA5RE0L38dPT9HaDhuBTifvVmKDYeOgJ7t313
 - AccessToken : 313287708-NO5SPbreQvoOxtXUD5EcKlubIfCBNfCb6aRqYBlZ
 - AccessTokenSecret : LVfDgtlfeht5yjBJGSgvSvtMYcFMoEdYOspYoOptcuR4i
- Akun Twitter penguji : kviniinktest123

5.2 Hasil Tampilan Antarmuka

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

```

run:
[Tue May 19 19:01:11 ICT 2016]Establishing connection.
[Tue May 19 19:01:30 ICT 2016]Connection established.
[Tue May 19 19:01:30 ICT 2016]Receiving status stream.
@kviniinktest123 - @kviniink bip to ip
Lokasi 1 : bip
Lokasi 2 : ip
@kviniinktest123 Jalan dari lokasi mulai Anda ke Jalan Merdeka sejauh kurang lebih 58 meter.
@kviniinktest123 Naik angkot Dago - St. Hall di Jalan Merdeka, dan turun di Jalan Riau kurang lebih setelah 3,9 kilometer.
@kviniinktest123 Jalan dari Jalan Riau ke tujuan akhir Anda sejauh kurang lebih 92 meter.
@kviniinktest123 Untuk lebih lengkap silahkan lihat di http://kiri.travel?start=%20bip&finish=ip&region=bdo
@kviniinktest123 - @kviniink badung to unpar
Lokasi 1 : badung
Lokasi 2 : unpar
@kviniinktest123 badung tidak ditemukan

```

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

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

- Antar muka Twitter yang diakses melalui aplikasi Twitter di Android dapat dilihat pada gambar 5.2



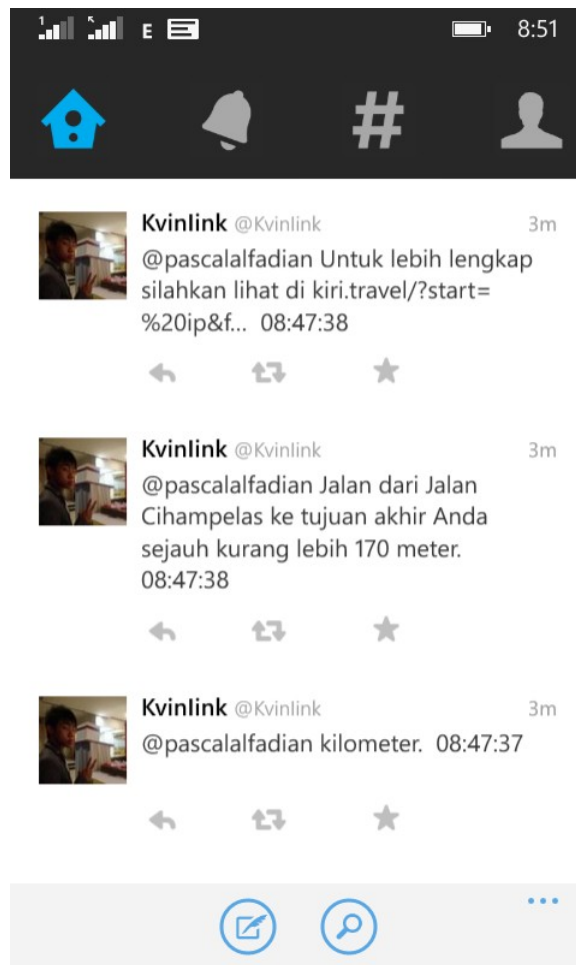
Gambar 5.2: Antar muka Twitter yang diakses melalui aplikasi Twitter di Android

- Antar muka Twitter yang diakses melalui aplikasi Twitter di iOS dapat dilihat pada gambar 5.3



Gambar 5.3: Antar muka Twitter yang diakses melalui aplikasi Twitter di iOS

- 1 • Antar muka Twitter yang diakses melalui aplikasi Twitter di Windows Phone dapat
- 2 dilihat pada gambar 5.4



Gambar 5.4: Antar muka Twitter yang diakses melalui aplikasi Twitter di Windows Phone

- 3 • Antar muka Twitter yang diakses melalui aplikasi Twitter di Website Twitter dapat
- 4 dilihat pada gambar 5.5



Gambar 5.5: Antar muka Twitter yang diakses melalui aplikasi Twitter di Website Twitter

5.3 Pengujian

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

5.3.1 Pengujian Fungsional

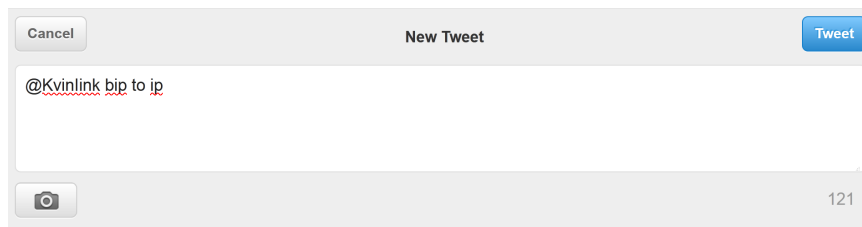
Pengujian fungsional dilakukan pada fungsionalitas yang tersedia pada aplikasi yang dibangun. Pengujian ini dilakukan untuk mengetahui kesesuaian reaksi nyata dengan reaksi yang diharapkan dari aplikasi yang dibangun. Hasil pengujian ditunjukkan pada tabel 5.1.

5.3.2 Pengujian Eksperimental

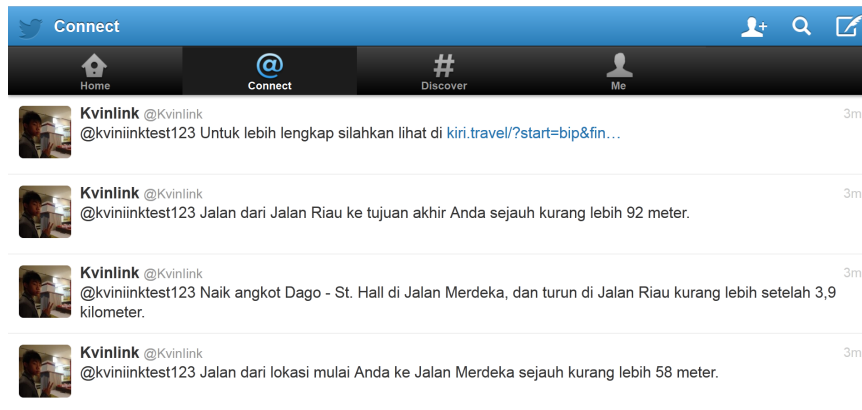
Pada sub bab ini akan dilakukan pengujian terhadap *Twitter bot* untuk mencari jalur transportasi publik. Peneliti meminta kepada beberapa orang untuk melakukan pencarian jalur transportasi publik kepada *Twitter bot* untuk mencari jalur transportasi publik. Selain itu juga peneliti mencoba melakukan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

1. Pengujian 1

Pada pengujian 1, peneliti mencoba untuk mencari jalur transportasi publik untuk lokasi yang umum dikunjungi yaitu *mall*. Pencarian dilakukan dengan lokasi awal yaitu BIP (Bandung Indah Plaza) menuju lokasi tujuan yaitu IP (Istana Plaza). Akun penguji @kviniinktest123 melakukan *mention* kepada akun *Twitter bot* @kviniink123 yang dapat dilihat pada gambar 5.6.



Gambar 5.6: Tweet dari BIP menuju IP



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

Setelah proses *tweet* dilakukan, *Twitter bot* akan menangkap *tweet* tersebut dan memprosesnya. Setelah proses pencarian selesai dilakukan, akun *Twitter bot* @kvinlink melakukan *reply* kepada akun @kvinlinktest123 yang dapat dilihat pada gambar 5.7.

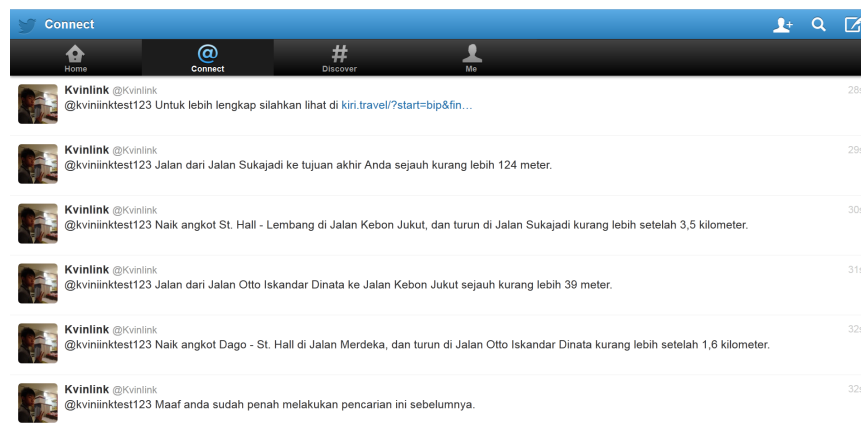
Pencarian kedua dilakukan dengan lokasi awal yaitu BIP (Bandung Indah Plaza) dan lokasi tujuan yaitu PVJ (Paris van Java). Dapat dilihat pada gambar 5.8, akun @kvinlinktest123 melakukan *tweet* pencarian jalur transportasi publik yang di-mention kepada akun *Twitter bot* @kvinlink dengan lokasi awal yaitu BIP dan lokasi tujuan yaitu PVJ.

Setelah itu *tweet* tersebut diproses oleh aplikasi untuk dicari jalur transportasi publiknya, lalu akun *Twitter bot* @kvinlink melakukan *reply* kepada akun @kvinlinktest123. *Reply tweet* tersebut merupakan jalur transportasi publik yang harus ditempuh, *reply tweet* tersebut dapat dilihat pada gambar 5.9.

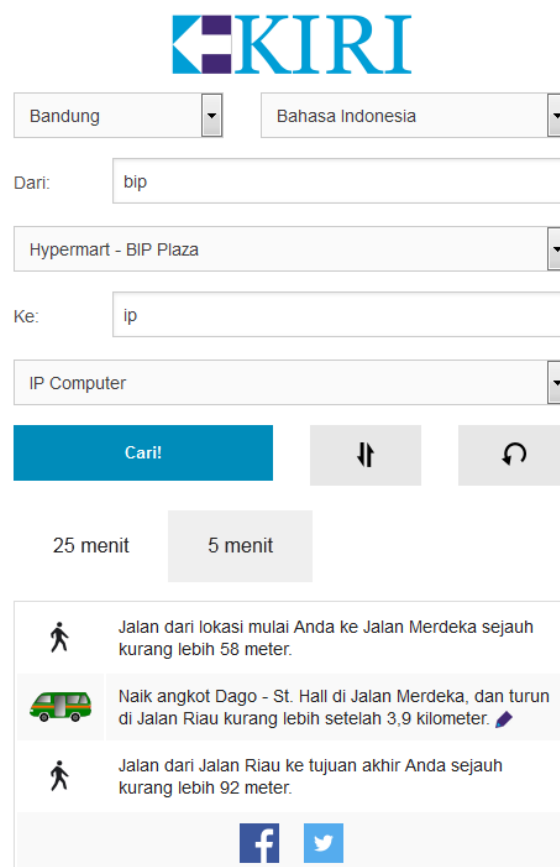
Pada pencarian kedua dapat dilihat pada *tweet* pertama terjadi ketidak sesuaian hasil dari KIRI API dengan hasil *tweet*. Peneliti lalu melakukan pencarian melalui *website* KIRI yaitu <http://kiri.travel>. Pencarian pertama pada *website* KIRI dilakukan dengan lokasi awal yaitu BIP dan lokasi tujuan yaitu IP. Hasil pencarian pada *website* KIRI dapat dilihat pada gambar 5.10.



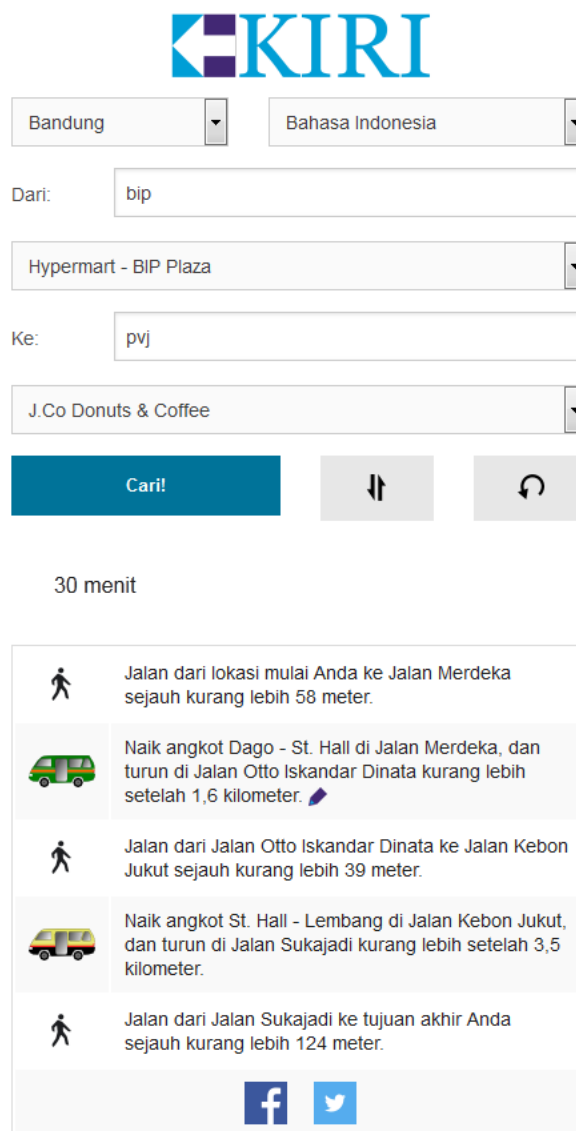
Gambar 5.8: Tweet dari BIP menuju PVJ



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



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



KIRI

Bandung Bahasa Indonesia

Dari: bip






Hypermart - BIP Plaza



Ke: pvj

J.Co Donuts & Coffee

Cari!

30 menit

| | |
|---|---|
|  | Jalan dari lokasi mulai Anda ke Jalan Merdeka sejauh kurang lebih 58 meter. |
|  | Naik angkot Dago - St. Hall di Jalan Merdeka, dan turun di Jalan Otto Iskandar Dinata kurang lebih setelah 1,6 kilometer. |
|  | Jalan dari Jalan Otto Iskandar Dinata ke Jalan Kebon Jukut sejauh kurang lebih 39 meter. |
|  | Naik angkot St. Hall - Lembang di Jalan Kebon Jukut, dan turun di Jalan Sukajadi kurang lebih setelah 3,5 kilometer. |
|  | Jalan dari Jalan Sukajadi ke tujuan akhir Anda sejauh kurang lebih 124 meter. |

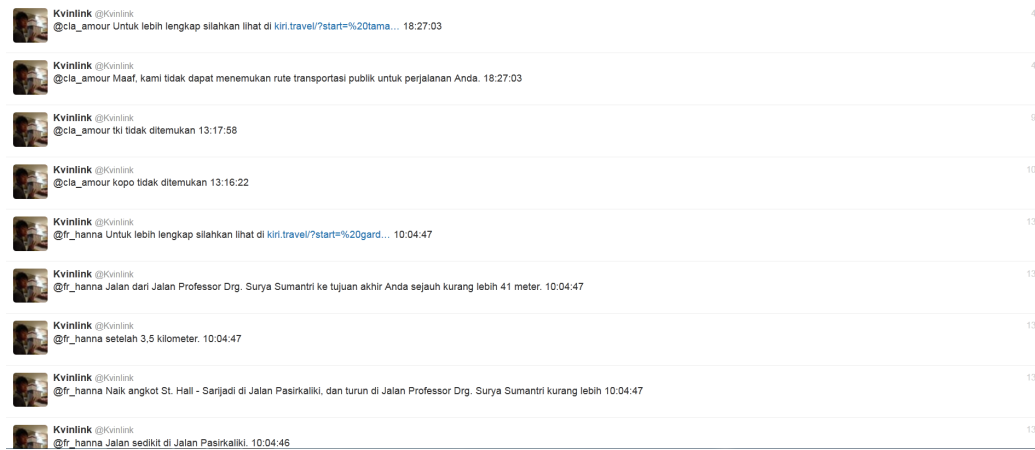
 

Gambar 5.11: Hasil Pencarian Jalur Transportasi Publik dari BIP menuju PVJ Melalui Website KIRI

- 1 Lalu pencarian kedua pada *website* KIRI dilakukan dengan lokasi awal yaitu BIP dan
- 2 lokasi tujuan yaitu PVJ. Hasil pencarian KIRI dari BIP menuju PVJ dapat dilihat pa-
- 3 da gambar 5.11. Setelah dilihat dari hasil keduanya, *Twitter bot* melakukan *duplicate*
- 4 *tweet* pada *tweet* pertama dalam pencarian ke dua yang dilakukan oleh akun @kvini-
- 5 ink123. *Duplicate tweet* adalah *tweet* yang dinyatakan dinyatakan identik oleh Twitter
- 6 dalam jangka waktu tertentu. *Duplicate tweet* tidak diperbolehkan oleh Twitter. Oleh
- 7 karena itu, untuk menghindari adanya *duplicate tweet*, penulis menambahkan waktu
- 8 untuk jam, menit, dan detik di setiap *tweet* yang dilakukan oleh *Twitter bot* agar
- 9 membuat setiap *tweet* tersebut bersifat unik.

| 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 @kviniink | Setiap <i>tweet</i> yang <i>dimention</i> kepada akun <i>Twitter bot</i> @kviniink 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 <i>Twitter bot</i> @kviniink melakukan <i>reply</i> kepada akun penguji @kviniinktest123, <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



Gambar 5.12: Hasil Reply Twitter Bot

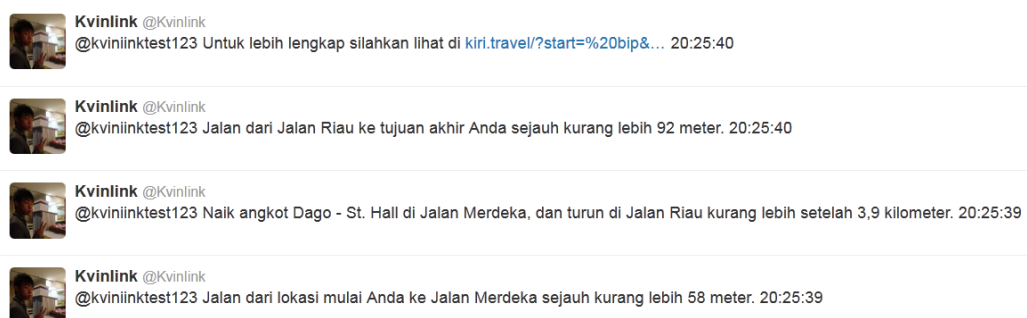


Gambar 5.13: Akun Twitter Bot Mendapat Banyak Mention Dalam Satu Tweet










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 untuk jam, menit, dan detik pada setiap *tweet*, aplikasi *Twitter bot* berjalan dengan baik. *Twitter bot* dapat memberitahu bahwa suatu lokasi pencarian tidak ditemukan yang dapat dilihat pada gambar 5.12, 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 5.12. *Twitter bot* tidak akan mendapatkan *error* ketika akun Twitter Bot @kvinlink mendapat banyak *mention* dalam satu *tweet* seperti pada gambar 5.13, jika format penulisan benar maka *Twitter bot* akan tetap mencari jalur transportasi publiknya yang dapat dilihat pada gambar 5.14.










Gambar 5.12 , gambar 5.15 , dan gambar 5.16 merupakan beberapa hasil *reply* dari



Gambar 5.14: Hasil Reply Twitter Bot

| | | |
|---|--|-----|
|  | Kvinlink @Kvinlink @renairinn cibaduyut tidak ditemukan 23:03:14 | 24h |
|  | Kvinlink @Kvinlink @lip_lip_lip pasirkaliki tidak ditemukan 22:21:54 | 1d |
|  | Kvinlink @Kvinlink @lip_lip_lip batununggal tidak ditemukan 22:21:07 | 1d |
|  | Kvinlink @Kvinlink @lip_lip_lip pasir kaliki tidak ditemukan 22:04:20 | 1d |
|  | Kvinlink @Kvinlink @chensansan Untuk lebih lengkap silahkan lihat di kiri.travel/?start=%20jela... 20:59:34 | 1d |
|  | Kvinlink @Kvinlink @chensansan Naik angkot Ciumbuleuit ke tujuan akhir Anda sejauh kurang lebih 39 meter. 20:59:34 | 1d |
|  | Kvinlink @Kvinlink @chensansan 6,8 kilometer. 20:59:32 | 1d |
|  | Kvinlink @Kvinlink @chensansan Naik angkot Ciumbuleuit - St. Hall (lurus) di Jalan Kebon Jukut, dan turun di Jalan Ciumbuleuit kurang lebih setelah 20:59:32 | 1d |
|  | Kvinlink @Kvinlink @chensansan Jalan dari Jalan Stasiun Timur ke Jalan Kebon Jukut sejauh kurang lebih 27 meter. 20:59:32 | 1d |

Gambar 5.15: Hasil Reply Twitter Bot

| | | |
|---|---|----|
|  | Kvinlink @Kvinlink @kvinlinktest123 badung tidak ditemukan 20:26:33 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ Untuk lebih lengkap silahkan lihat di kiri.travel/?start=%20soek... 20:25:57 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ Jalan dari Jalan Ciumbuleuit ke tujuan akhir Anda sejauh kurang lebih 39 meter. 20:25:57 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ setelah 5,0 kilometer. 20:25:56 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ Naik angkot Ciumbuleuit - St. Hall (lurus) di Jalan Pajajaran, dan turun di Jalan Ciumbuleuit kurang lebih 20:25:56 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ Jalan sedikit di Jalan Pajajaran. 20:25:55 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ kilometer. 20:25:54 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ Naik angkot Dago - Caringin di Jalan Soekarno Hatta, dan turun di Jalan Pajajaran kurang lebih setelah 6,3 20:25:54 | 1d |
|  | Kvinlink @Kvinlink @katherineangel_ Jalan dari lokasi mulai Anda ke Jalan Soekarno Hatta sejauh kurang lebih 38 meter. 20:25:53 | 1d |

Gambar 5.16: 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 penelitian 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 akun @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 nama jalan harus lengkap. Sebagai contoh adalah jalan mekar wangi, jalan kopo. Jika penulisan hanya mekar wangi atau kopo saja, maka terjadi kemungkinan pencarian lokasi tidak akan ditemukan.
2. Membuat akun *Twitter bot* menjadi premium agar tidak mengalami adanya keterbatasan *tweet* per harinya.
3. Pada biodata informasi akun *Twitter bot*, sebaiknya diberitahu cara penggunaan *Twitter bot* untuk mencari jalur transportasi publik agar pengguna bisa langsung mencoba.
4. Pengguna tidak harus melakukan *mention* kepada akun *Twitter Bot*, pengguna dapat memanfaatkan fungsi *hashtag* yang telah diberikan Twitter.
5. Mengatasi format *tweet* agar tidak terlihat seperti *spam*.

DAFTAR REFERENSI

- [1] "Twitter Documentation." <https://dev.twitter.com/overview/documentation>, 2014. Accessed: 2014-8-20
- [2] Tim O'Reilly, *The Twitter Book*. O'Reilly Media, Inc, 2009
- [3] "KIRI API v2 Documentation." https://bitbucket.org/projectkiri/kiri_api/wiki/KIRI%20API%20v2%20Documentation, 2014. Accessed: 2014-8-20
- [4] "Twitter4J Documentation." <http://twitter4j.org/javadoc/index.html>, 2007. Accessed: 2014-8-20

1

LAMPIRAN A

2

KODE PROGRAM KELAS MAIN

Listing A.1: Main.java

```
3  {
4      import twitter4j.FilterQuery;
5      import twitter4j.TwitterStream;
6      import twitter4j.TwitterStreamFactory;
7
8      /**
9       *
10      * @author Kevin
11      */
12      public class Main {
13          public static void main(String[] args){
14              TwitterStream twitterStream = new TwitterStreamFactory().getInstance();
15              TwitterGateway twittergateway = new TwitterGateway();
16
17              FilterQuery fq = new FilterQuery();
18              String keywords[] = {twittergateway.screenName};
19
20              fq.track(keywords);
21
22              twitterStream.addListener(twittergateway);
23              twitterStream.filter(fq);
24          }
25      }
26
27 }
```


LAMPIRAN B

KODE PROGRAM KELAS KIRIGATEWAY

Listing B.1: KIRIGateway.java

```
3 {
4     import java.text.DateFormat;
5     import java.text.SimpleDateFormat;
6     import twitter4j.*;
7
8     import java.util.ArrayList;
9     import java.util.Arrays;
10    import java.util.Date;
11    import java.util.logging.Level;
12
13    public final class TwitterGateway implements StatusListener{
14        public static final String screenName = "@Kvinlink";
15        private String user;
16        private String location [];
17        private String latlon [] = new String[2];
18        private RoutingResponse routingResponse;
19        private Step [] step;
20        private Steps steps;
21        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
22        Date date = new Date();
23
24        @Override
25        public void onStatus(Status status) {
26            user = status.getUser().getScreenName();
27            String mentionStatus = status.getText();
28            System.out.println("@ " + user + " - " + mentionStatus);
29            String paramScreenName = screenName.toLowerCase();
30            mentionStatus = mentionStatus.toLowerCase().replace(paramScreenName, "");
31            location = mentionStatus.split(" to ");
32            boolean statusLocation1 = false;
33            boolean statusLocation2 = false;
34
35
36            if(location.length == 2){
37                try {
38                    System.out.println("Lokasi 1 : "+location[0]);
39                    System.out.println("Lokasi 2 : "+location[1]);
40                    //string destination menampung hasil dari JSONObject hasil
41                    //pencarian apa saja yang ditemukan dari KIRIGateway.
42                    getLatLong
43                    String destination1 = KIRIGateway.GetLatLong(location[0]);
44                    String destination2 = KIRIGateway.GetLatLong(location[1]);
45
46                    //dimasukan ke JSONObject
47                    JSONObject objDest1 = new JSONObject(destination1);
48                    JSONObject objDest2 = new JSONObject(destination2);
49
50                    //memasukan hasil pencarian pertama dari JSONObject ke atribut routingResponse
51                    JSONObject res1 = objDest1.getJSONArray("searchresult").
52                        getJSONObject(0);
53                    String hasilDest1 = res1.getString("placename");
54                    latlon[0] = res1.getString("location");
55                    if(hasilDest1 != null)
56                    {
57                        statusLocation1 = true;
58                    }
59
60                    JSONObject res2 = objDest2.getJSONArray("searchresult").
61                        getJSONObject(0);
62                    String hasilDest2 = res2.getString("placename");
63                    latlon[1] = res2.getString("location");
64                    if(hasilDest2 != null)
65                    {
66                        statusLocation2 = true;
```

```

1      }
2
3      //Mendapatkan hasil pencarian lalu dimasukkan ke JSONArray
4      paramSteps untuk dipisah-pisah lalu dimasukkan ke
5      RoutingResponse
6      String hasilPencarian = KIRIGateway.GetTrack(latlon[0],
7      latlon[1]);
8      JSONObject objTrack = new JSONObject(hasilPencarian);
9      JSONObject routingresults = objTrack.getJSONArray("
10      routingresults").getJSONObject(0);
11      JSONArray paramSteps = routingresults.getJSONArray("steps
12      ");
13      //buat variable step, steps, dan routing response
14      step = new Step[paramSteps.length()];
15      for (int i = 0; i < step.length; i++) {
16          step[i] = new Step(paramSteps.getJSONArray(i).
17          getString(3) + "");
18      }
19      steps = new Steps(step);
20
21      routingResponse = new RoutingResponse(objTrack.getString("
22      status"), steps);
23      if(routingResponse.getStatus().equals("ok")){
24          for (int i = 0; i < routingResponse.
25          getRoutingResult().getSteps().length ; i++) {
26              date = new Date();
27              Tweet(user, routingResponse.
28              getRoutingResult().getSteps()[i].
29              getHumanDescription());
30          }
31          Tweet(user, "Untuk lebih lengkap silahkan lihat di
32          http://kiri.travel?start="+ location[0].
33          replace(" ", "%20") + "&finish="+ location[1].
34          replace(" ", "%20") + "&region=bdo");
35
36          for (int i = 0; i < routingResponse.
37          getRoutingResult().getSteps().length ; i++) {
38              System.out.println("@"+user + " " +
39              routingResponse.getRoutingResult().
40              getSteps()[i].getHumanDescription());
41          }
42          System.out.println("@"+user + " Untuk lebih
43          lengkap silahkan lihat di http://kiri.travel?
44          start="+ location[0].replace(" ", "%20") + "&
45          finish="+ location[1].replace(" ", "%20") + "&
46          region=bdo");
47      }else{
48          System.out.println("status error");
49      }
50      } catch (Exception ex) {
51          try {
52              if(!statusLocation1)
53              {
54                  date = new Date();
55                  Tweet(user,location[0] + " tidak ditemukan
56                  ");
57                  System.out.println("@"+user+" "+location
58                  [0] + " tidak ditemukan");
59              }
60              else if(!statusLocation2)
61              {
62                  date = new Date();
63                  Tweet(user,location[1] + " tidak ditemukan
64                  ");
65                  System.out.println("@"+user+" "+location
66                  [1] + " tidak ditemukan");
67              }
68              else
69              {
70                  Tweet(user,"Pencarian tidak ditemukan");
71                  System.out.println("@"+user+" Pencarian
72                  tidak ditemukan");
73              }
74              //java.util.logging.Logger.getLogger(
75              TwitterGateway.class.getName()).log(Level.
76              SEVERE, null, ex);
77          } catch (TwitterException ex1) {
78              java.util.logging.Logger.getLogger(TwitterGateway.
79              class.getName()).log(Level.SEVERE, null, ex1);
80          }
81      }
82  }
83
84  }
85

```

```

1      @Override
2      public void onDeleteNotice(StatusDeletionNotice statusDeletionNotice) {
3          System.out.println("Got a status deletion notice id:" +
4              statusDeletionNotice.getStatusId());
5      }
6
7      @Override
8      public void onTrackLimitationNotice(int numberOfLimitedStatuses) {
9          System.out.println("Got track limitation notice:" +
10              numberOfLimitedStatuses);
11      }
12
13      @Override
14      public void onScrubGeo(long userId, long upToStatusId) {
15          System.out.println("Got scrub_geo event userId:" + userId + " upToStatusId
16              : " + upToStatusId);
17      }
18
19      @Override
20      public void onException(Exception ex) {
21          System.out.println("onException");
22          ex.printStackTrace();
23      }
24
25      @Override
26      public void onStallWarning(StallWarning sw) {
27          System.out.println("onStallWarning");
28          throw new UnsupportedOperationException("Not supported yet."); //To change
29              body of generated methods, choose Tools | Templates.
30      }
31
32      public void Tweet(String user, String paramStatusUpdate) throws TwitterException
33      {
34          Twitter twitter = new TwitterFactory().getInstance();
35          int userLength = user.length();
36          int maxTweet = 140 - (userLength + 2 + 9); // ditambah 2 untuk @ dan " ",
37              ditambah 9 untuk waktu " HH:mm:ss";
38
39          String[] tampung = paramStatusUpdate.split(" ", 0); //misahin semua kata
40              jadi array of kata
41
42          StatusUpdate[] statusUpdate = new StatusUpdate[(paramStatusUpdate.length()
43              / maxTweet) + 1]; //panjang status dibagi batas maksimal huruf yang
44              diperbolehkan, ditambah 1 karena 130/140 = 0
45          String[] tampungStatusUpdate = new String[statusUpdate.length]; //bikin
46              banyaknya array sepanjang tweet yang mau di tweet
47
48          int increment = 0;
49          int incTampung = 0;
50          for (int i = 0; i < tampungStatusUpdate.length; i++) {
51              tampungStatusUpdate[i] = "@" + user + " "; //setiap tweet harus
52              dimasukin nama user yang dituju
53          }
54
55          while(increment < statusUpdate.length){
56
57              while(incTampung < tampung.length){
58                  tampungStatusUpdate[increment] += tampung[incTampung]; //
59                      nambahin kata
60                  if(tampungStatusUpdate[increment].length() >= 131){ //
61                      melakukan cek apakah tweet sudah over 140 kata atau
62                      belum, 131 soalnya 140 dikurang 9 untuk waktu " HH:mm:
63                      ss";
64
65                      tampungStatusUpdate[increment] =
66                          tampungStatusUpdate[increment].substring(0,
67                              tampungStatusUpdate[increment].length() -
68                              tampung[incTampung].length()); //ngebuang kata
69                          terakhir yang ditambahin
70                      tampungStatusUpdate[increment] += " "+ dateFormat.
71                          format(date);
72                      increment++;
73
74                      break;
75                  }else{
76                      tampungStatusUpdate[increment] += " "; //Jika
77                          belum 140 kata maka ditambahkan spasi
78                  }
79                  incTampung++;
80                  if(incTampung >= tampung.length){
81                      tampungStatusUpdate[increment] += " "+ dateFormat.
82                          format(date);
83                      increment++; //kalo kata-kata sudah habis,
84                          keluarin dari statement
85                  }

```

```
1         }
2     }
3
4     for (int i = 0; i < statusUpdate.length; i++) {
5         statusUpdate[i] = new StatusUpdate(tampungStatusUpdate[i]);
6     }
7     try {
8         for (int i = 0; i < statusUpdate.length; i++) {
9             twitter.updateStatus(statusUpdate[i]);
10        }
11    } catch (TwitterException ex) {
12        twitter.updateStatus("@ " + user + " Maaf anda sudah pernah
13            melakukan pencarian ini sebelumnya. " + dateFormat.format(date
14            ) );
15        System.out.println("Error : " + ex);
16    }
17 }
18 }
19
20 }
```

LAMPIRAN C

KODE PROGRAM KELAS KIRIGATEWAY

Listing C.1: KIRIGateway.java

```
3 {
4     import java.io.BufferedReader;
5     import java.io.InputStreamReader;
6     import java.net.HttpURLConnection;
7     import java.net.URL;
8     import java.net.URLEncoder;
9
10    /**
11     *
12     * @author Kevin
13     */
14    public class KIRIGateway {
15        public static String GetLatLong(String destination) throws Exception {
16            String url = "http://kiri.travel/handle.php?version=2&mode=
17                searchplace&region=bdo&querystring=" + URLEncoder.encode(
18                destination.trim(), "UTF-8")+"&apikey=889C2C8FBB82C7E6";
19
20            URL obj = new URL(url);
21            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
22
23            con.setRequestMethod("GET");
24
25            int responseCode = con.getResponseCode();
26
27            BufferedReader in = new BufferedReader(
28                new InputStreamReader(con.getInputStream()));
29            String inputLine;
30            StringBuffer response = new StringBuffer();
31
32            while ((inputLine = in.readLine()) != null) {
33                response.append(inputLine);
34            }
35            in.close();
36
37            return response.toString();
38        }
39
40        public static String GetTrack(String dest1, String dest2) throws Exception
41        {
42            //presentation == desktop biar tidak ada %toicon and %fromicon
43            String url = "http://kiri.travel/handle.php?version=2&mode=
44                findroute&locale=id&start="+dest1+"&finish="+dest2+"&
45                presentation=desktop&apikey=889C2C8FBB82C7E6";
46            URL obj = new URL(url);
47            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
48
49            con.setRequestMethod("GET");
50
51            int responseCode = con.getResponseCode();
52
53            BufferedReader in = new BufferedReader(new InputStreamReader(con.
54                getInputStream()));
55            String inputLine;
56            StringBuffer response = new StringBuffer();
57
58            while ((inputLine = in.readLine()) != null) {
59                response.append(inputLine);
60            }
61            in.close();
62
63            return response.toString();
64        }
65    }
66 }
```

1 | }

1

LAMPIRAN D

2

KODE PROGRAM KELAS ROUTINGRESPONSE

Listing D.1: RoutingResponse.java

```
3  {
4      /**
5       *
6       * @author Kevin
7       */
8      public class RoutingResponse {
9          public String status;
10         public Steps routingResult;
11
12         public RoutingResponse(String paramStatus, Steps paramRoutingResult){
13             this.status = paramStatus;
14             this.routingResult = paramRoutingResult;
15         }
16
17         public RoutingResponse() {
18             this.status = "";
19             this.routingResult = null;
20         }
21         public String getStatus() {
22             return status;
23         }
24
25         public void setStatus(String status) {
26             this.status = status;
27         }
28
29         public Steps getRoutingResult() {
30             return routingResult;
31         }
32
33         public void setRoutingResult(Steps routingResult) {
34             this.routingResult = routingResult;
35         }
36     }
37 }
```


1

LAMPIRAN E

2

KODE PROGRAM KELAS STEP

Listing E.1: Step.java

```
3  {
4      /**
5       *
6       * @author Kevin
7       */
8      class Step {
9          public String humanDescription;
10
11          public Step(String paramHumanDescription){
12              this.humanDescription = paramHumanDescription;
13          }
14
15          public Step() {
16              this.humanDescription = "";
17          }
18          public String getHumanDescription() {
19              return humanDescription;
20          }
21
22          public void setHumanDescription(String humanDescription) {
23              this.humanDescription = humanDescription;
24          }
25      }
26
27
28 }
```


1

LAMPIRAN F

2

KODE PROGRAM KELAS STEPS

Listing F.1: Steps.java

```
3  {
4      /**
5       *
6       * @author Kevin
7       */
8      class Steps {
9          public Step [] steps;
10
11          public Steps(Step [] paramSteps){
12              this.steps = paramSteps;
13          }
14          public Step [] getSteps() {
15              return steps;
16          }
17
18          public void setSteps(Step[] steps) {
19              this.steps = steps;
20          }
21      }
22 }
```