LAPORAN PENELITIAN

INTEGRASI SITUS NAVIGASI DENGAN SITUS *CROWDSOURCING* RUTE ANGKOT

PASCAL ALFADIAN NUGROHO

NPM: 2003730013

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

RESEARCH REPORT

INTEGRATION OF *ANGKOT* NAVIGATION AND ROUTE CROWDSOURCING WEBSITE

PASCAL ALFADIAN NUGROHO

NPM: 2003730013

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

LEMBAR PENGESAHAN

INTEGRASI SITUS NAVIGASI DENGAN SITUS *CROWDSOURCING* RUTE ANGKOT

PASCAL ALFADIAN NUGROHO

NPM: 2003730013

Bandung, 19 Maret 2015

Menyetujui,

Pembimbing Tunggal

Lionov, M.Sc.

Ketua Tim Penguji	Anggota Tim Penguji
Thomas Anung Basuki, Ph.D.	Dr. rer. nat. Cecilia Esti Nugraheni

Mengetahui,

Ketua Program Studi

Thomas Anung Basuki, Ph.D.

PERNYATAAN

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

INTEGRASI SITUS NAVIGASI DENGAN SITUS *CROWDSOURCING* RUTE ANGKOT

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

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

Dinyatakan di Bandung,
Tanggal 19 Maret 2015

Meterai

Pascal Alfadian Nugroho
NPM: 2003730013

ABSTRAK

Penelitian ini mengintegrasikan situs navigasi rute angkot http://kiri.travel dengan situs *crowdsourcing* rute angkot https://angkot.web.id, di mana pengguna dapat berkontribusi memperbaiki rute angkot yang salah. Integrasi yang dimaksud adalah sinkronisasi data secara berkala dan otomatis, sehingga hasil navigasi yang diberikan mendekati ketepatan sesuai di lapangan.

Di akhir penelitian, integrasi berhasil diimplementasikan secara teknis, namun partisipasi pengguna dalam memperbaiki rute dirasa masih kurang. Sebagai catatan, hasil survei juga menyatakan rute yang dimiliki KIRI saat ini sudah baik, sehingga dapat menjelaskan alasan kurangnya partisipasi perbaikan data.

Kata-kata kunci: *crowdsourcing*, angkot, rute, navigasi, *REST*, *JSON*

ABSTRACT

This research integrates *angkot* (public bus) navigation website http://kiri.travel with *angkot* route crowdsourcing website, where public user can contribute by fixing erroneus *angkot* routes. The aforementioned integration is defined as automatic synchronization of both parties' data, such that the result accuracy of navigation is close to what found on the field.

At the end of this research, integration was implemented successfully, but users participation of route fixing is considerably minimum. As a note, result of survey showed that the routes provided by KIRI is already in good quality, hence can explain the reason behind lack of route fixing participation.

Keywords: crowdsourcing, route, navigation, REST, JSON

Dipersembahkan untuk seluruh warga Indonesia yang selalu setia menggunakan transportasi umum dan mahasiswa FTIS UNPAR yang akan menyelesaikan skripsinya.

KATA PENGANTAR

Proses bimbingan yang ideal menurut saya membahas hal-hal yang menarik seperti eksplorasi ide, analisis, eksperimen, dll. Kenyataannya, di tengah waktu yang terbatas, waktu bimbingan terbuang untuk hal-hal yang tidak terlalu penting, seperti tata cara penulisan, *troubleshooting* LaTeX, memperbaiki nalar yang salah, dll. Oleh karena itu, saya membuat contoh skripsi ini, yang terkait dengan template skripsi saya[1].

Contoh skripsi ini dibuat untuk memberi gambaran dan panduan bagi mahasiswa dalam pengambilan Skripsi 1 maupun 2. Kode sumber dari dokumen ini dapat diakses di *branch* "contoh". Jika ada kesalahan ataupun masukan, silahkan berkonsultasi terlebih dahulu pada daftar *issue* atau buat *issue* baru dengan label "contoh".

Peringatan: Walaupun segala usaha telah diupayakan untuk kesempurnaan contoh ini, tidak ada jaminan bahwa dokumen ini benar dan ideal di mata penguji.

Selamat bekerja!

Bandung, Maret 2015

Penulis

[1] https://github.com/pascalalfadian/Skripsi

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

KIRI (http://kiri.travel, gambar 1.1) adalah sebuah situs yang dikelola oleh PT. Kirana Sistem Transportasi, yang menyediakan layanan navigasi dari satu titik ke titik lain memanfaatkan transportasi publik. Layanan ini diberikan secara gratis kepada pengunjung situs / pengguna aplikasi bergerak mereka. Untuk mendukung layanan tersebut, tim KIRI melakukan kurasi rute angkot secara mandiri di Bandung dengan mencatat perjalanan setiap trayek menggunakan peralatan GPS (*Global Positioning System*). Pada perkembangannya, KIRI melakukan ekspansi ke beberapa kota lainnya termasuk Jakarta. Hanya saja, karena keterbatasan sumberdaya, data di Jakarta dimasukkan hanya berbekal data yang tersedia di internet tanpa validasi di lapangan. Di sisi lain, ada beberapa pihak yang tertarik akan layanan KIRI di Bandung dan berniat untuk mendapatkan layanan serupa di kota mereka.

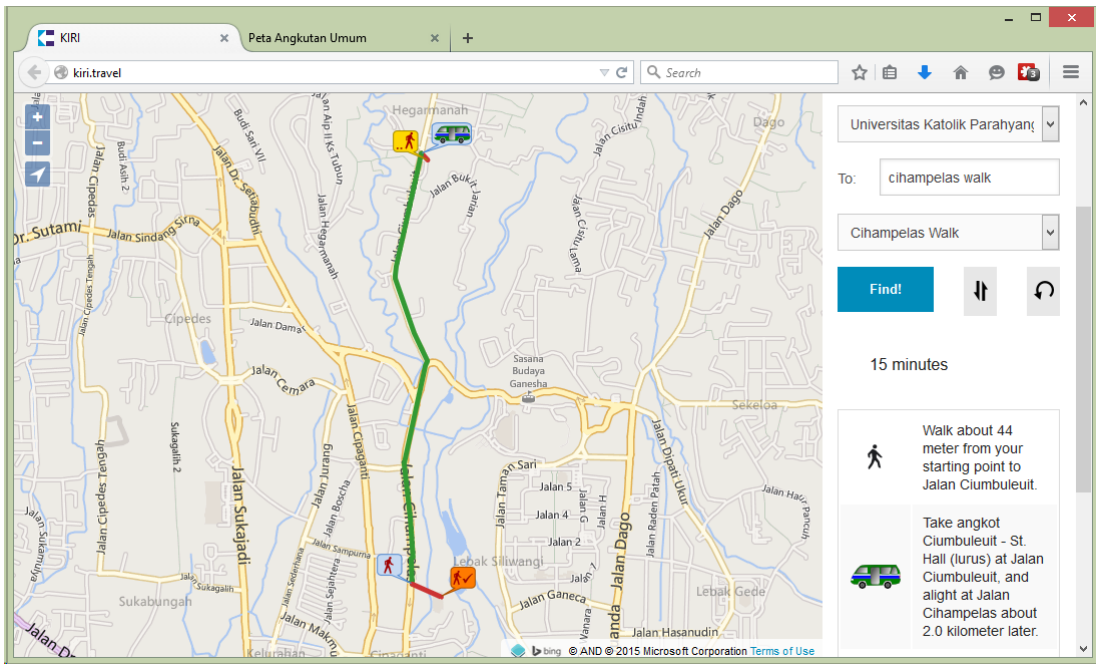
Situs Peta Angkutan Umum (https://angkot.web.id, gambar 1.2) merupakan sebuah situs yang dikembangkan oleh Fajran Iman Rusadi. Situs ini memungkinkan pengguna publik untuk melihat, memasukkan, atau memperbaiki data rute angkot di Indonesia (dengan kata lain, *crowdsourcing*). Seperti KIRI, layanan ini juga diberikan secara gratis.

Sebelum penelitian ini dilakukan, KIRI serta Peta Angkutan Umum merupakan dua buah situs yang terpisah dan bekerja secara independen.

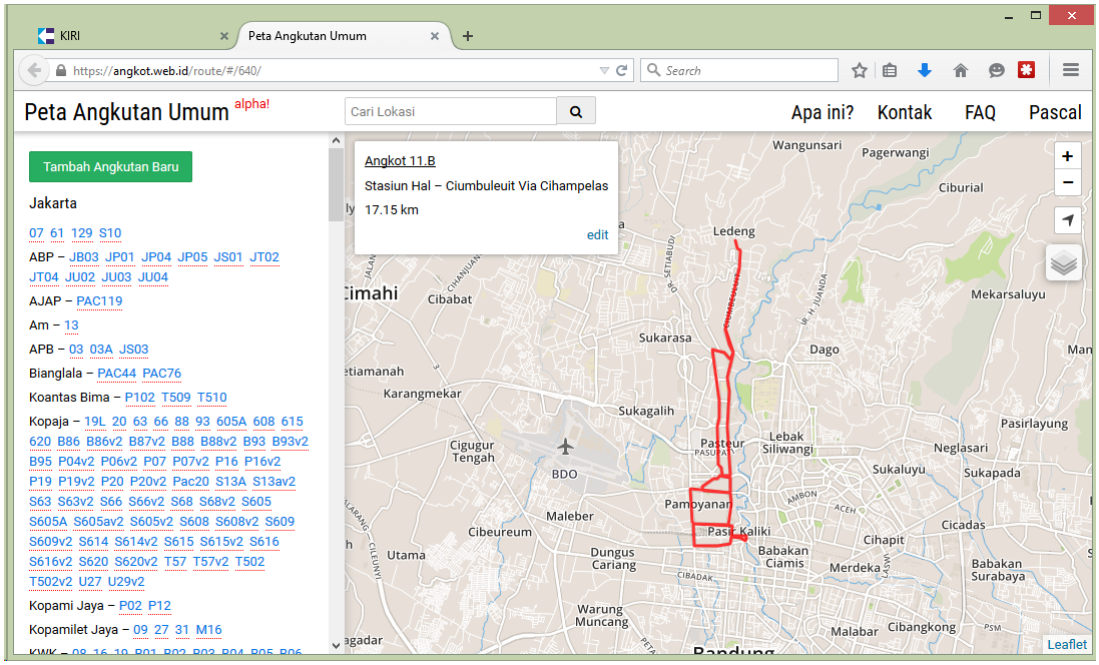
1.2 Rumusan Masalah

Dari latar belakang yang sudah dijelaskan, peneliti bermaksud untuk mengintegrasikan data yang dimiliki kedua situs web tersebut. Integrasi tersebut dirumuskan ke dalam masalah-masalah berikut:

1. Bagaimana mekanisme penarikan data oleh KIRI terhadap Peta Angkutan Umum secara otomatis?
2. Bagaimana memisahkan data yang dimiliki oleh KIRI sendiri dengan data yang ditarik dari Peta Angkutan Umum?
3. Bagaimana mengoptimasi protokol yang digunakan, sehingga kebutuhan *bandwidth* dapat dihemat?
4. Bagaimana respon pengguna KIRI terhadap fitur yang diimplementasikan?



Gambar 1.1: Situs Web KIRI



Gambar 1.2: Situs Web Peta Angkutan Umum

1.3 Tujuan

Berdasarkan rumusan masalah yang sudah dijabarkan, maka didefinisikan tujuan-tujuan berikut:

1. Mengimplementasikan mekanisme penarikan data otomatis oleh KIRI terhadap Peta Angkutan Umum.
2. Mengimplementasikan pemisahan data antara rute milik KIRI dengan data yang ditarik dari Peta Angkutan Umum.
3. Mengoptimasi protokol yang digunakan, sehingga kebutuhan *bandwidth* dapat dihemat.
4. Mempelajari respon pengguna KIRI terhadap fitur yang diimplementasikan.

1.4 Batasan Masalah

Penelitian ini memiliki batasan-batasan seperti berikut:

1. Penelitian dilakukan untuk rute angkot kota Bandung saja.
2. Dengan alasan kerahasiaan, penelitian ini tidak membahas keseluruhan bagian dari KIRI dan Peta Angkutan Umum, melainkan hanya bagian yang terkait penelitian saja.

1.5 Metode Penelitian

Dalam penelitian ini, akan dilakukan langkah-langkah berikut:

1. Melakukan studi terhadap mesin navigasi KIRI, protokol Peta Angkutan Umum, serta teori-teori lain yang mendukung kedua hal tersebut.
2. Melakukan analisis untuk menemukan hal yang dapat dilakukan untuk mengintegrasikan data kedua situs tersebut.
3. Melakukan perancangan untuk implementasi integrasi kedua sistem tersebut.
4. Melakukan implementasi dari rancangan yang sudah dilakukan.
5. Melakukan publikasi terhadap pengguna KIRI sehingga mereka dapat menguji hasil implementasi tersebut.
6. Menganalisa dan menarik kesimpulan atas hasil penelitian yang telah dilaksanakan.

1.6 Sistematika Penulisan

Berikut adalah sistematika penulisan dari dokumen ini:

- Bab 1 membahas latar belakang, rumusan masalah, tujuan penulisan, batasan-batasan, serta metode yang digunakan pada penelitian ini.

- Bab 2 membahas teori-teori yang digunakan dalam penelitian ini.
- Bab 3 membahas analisis yang dilakukan terhadap teori yang telah dijabarkan pada bab 2.
- Bab 4 membahas perancangan yang dilakukan sebelum mengimplementasikan integrasi yang dimaksud.
- Bab 5 membahas implementasi serta pengujian dari integrasi yang telah dilakukan.
- Bab 6 membahas kesimpulan dari keseluruhan penelitian ini, serta saran-saran yang dapat diberikan untuk penelitian berikutnya.

BAB 2

LANDASAN TEORI

2.1 JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data berbasis teks yang ringan dan tidak terikat pada bahasa tertentu[1]. Format JSON diturunkan dari bahasa pemrograman *ECMA-Script*[1].

Sebuah data JSON bisa berupa salah satu dari 5 elemen berikut:

1. **nilai literal**, yaitu salah satu dari "`false`", "`null`", atau "`true`" (tanpa tanda kutip).
2. **bilangan**, berupa bilangan bulat maupun desimal, dengan tanda titik ("`.`" sebagai pemisah antara bilangan bulat dengan pecahan.
3. ***string***, deretan karakter yang diapit dengan tanda kutip ganda ("`"`"). Di dalamnya bisa terdapat karakter karakter khusus seperti *line feed* ("`\n`"), tanda kutip ganda secara literal ("`\"`"), atau garis miring terbalik secara literal ("`\\`").
4. **larik** (*array*), nol atau lebih data JSON, yang dipisahkan koma ("`,`") dan diapit dengan kurung siku ("`[`" dan "`]`"). Data dalam larik tidak harus berjenis sama.
5. **objek**, nol atau lebih anggota berupa pasangan nama dan nilai, yaitu *string* teks dan data JSON yang dipisahkan dengan tanda titik dua ("`:`"). Untuk memisahkan pasangan satu dengan lainnya, digunakan tanda koma ("`,`").

Data JSON dapat mengandung karakter-karakter *whitespace* berupa spasi, *tab*, maupun *linefeed* dan akan diabaikan (kecuali di dalam *string*).

Berikut adalah contoh sebuah data JSON:

```
1 {
2     "lulus": false,
3     "nilaiUTS": 89.9,
4     "nilaiUAS": null,
5     "nilaiTugas": [75, null, 80.3, 100],
6     "nama": "Pascal"
7 }
```

Contoh di atas mendemonstrasikan sebuah objek (baris 1-7), nilai literal (baris 2 dan 4), bilangan (baris 3), larik (baris 5), dan *string* (baris 6).

[1]Sering juga dikenal dengan nama *JavaScript*

2.2 GeoJSON

GeoJSON adalah sebuah format berbasis JSON, untuk mengkodekan berbagai struktur data geografis[2]. Sebuah data GeoJSON selalu terdiri dari satu buah objek, yang memiliki aturan-aturan sebagai berikut:

1. Objek GeoJSON dapat memiliki berapapun jumlah anggota (pasangan nama / nilai).
2. Objek GeoJSON harus memiliki anggota, dengan nama "`type`", yang menunjukkan tipe dari objek GeoJSON tersebut.
3. Nilai dari `type` harus berupa salah satu dari "`Point`", "`MultiPoint`", "`LineString`", "`MultiLineString`", "`Polygon`", "`MultiPolygon`", "`GeometryCollection`", "`Feature`", atau "`FeatureCollection`" (*case sensitive*). Anggota ini menunjukkan tipe dari objek GeoJSON.
4. Objek GeoJSON dapat memiliki anggota dengan nama "`crs`" yang menunjukkan referensi sistem koordinat.
5. Objek GeoJSON dapat memiliki anggota dengan nama "`bbox`" yang menunjukkan *bounding box* (kotak yang melingkupi objek).

Subbab-subbab berikut menjelaskan bagian-bagian dari GeoJSON yang terkait dengan penelitian ini. Bagian lain yang tidak dijelaskan dapat dilihat pada [2].

2.2.1 *Geometry Object*

Geometry Object adalah objek-objek geometri, yaitu salah satu dari *Point*, *MultiPoint*, *LineString*, *MultiLineString*, *Polygon*, *MultiPolygon* atau *GeometryCollection*.

Sebuah *Geometry Object* harus memiliki sebuah anggota yang bernama "`coordinates`". Nilai dari *coordinates* adalah sebuah larik, yang isinya tergantung tipe objek tersebut.

2.2.2 Tipe *LineString*

Tipe ini digunakan untuk merepresentasikan deretan posisi, sehingga dalam GeoJSON objek ini direpresentasikan dengan sebuah larik dari posisi. Posisi sendiri adalah sebuah larik yang berisi dua atau lebih bilangan. Jika terdapat dua bilangan, maka kedua bilangan tersebut merepresentasikan koordinat dalam sumbu x dan y (bujur dan lintang). Jika ada bilangan ketiga, bilangan tersebut merepresentasikan koordinat dalam sumbu z (ketinggian). Bilangan keempat dan seterusnya tidak didefinisikan dalam standar ini.

Berikut adalah contoh sebuah objek *LineString*:

```
{
    "type": "LineString",
    "coordinates": [
        [107.60486, -6.88323],
        [107.60417, -6.88182],
        [107.60345, -6.87968],
        [107.60445, -6.87525]
    ]
}
```

Contoh di atas menunjukkan sebuah objek *LineString* dengan deretan 4 posisi yang menunjukkan sebagian dari Jalan Ciumbuleuit, Bandung.

2.2.3 Tipe *MultiLineString*

Tipe ini digunakan untuk merepresentasikan kumpulan dari *LineString*, direpresentasikan oleh larik dari *LineString* (dengan kata lain, larik dari larik dari posisi).

Berikut adalah contoh sebuah objek *MultiLineString*:

```
{
    "type": "MultiLineString",
    "coordinates": [
        [
            [107.60486, -6.88323],
            [107.60417, -6.88182],
            [107.60345, -6.87968],
            [107.60445, -6.87525]
        ],
        [
            [107.60485, -6.87369],
            [107.60563, -6.87381],
            [107.60660, -6.87411],
            [107.60683, -6.87407]
        ]
}
```

Contoh di atas menunjukkan sebuah objek *MultiLineString* dengan dua deretan, masing-masing 4 posisi yang menunjukkan sebagian dari Jalan Ciumbuleuit dan Jalan Menjangan, Bandung.

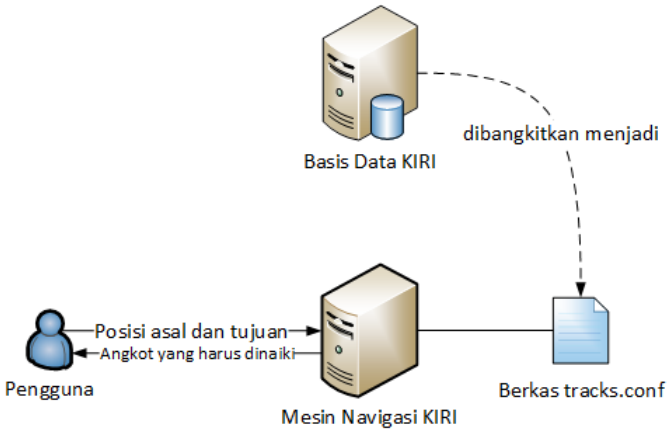
2.2.4 Tipe *Feature*

Tipe ini digunakan untuk merepresentasikan fitur dari sebuah objek geometri (kumpulan dari berbagai tipe lainnya). Aturan dari objek GeoJSON bertipe *Feature* adalah:

1. Objek ini harus memiliki anggota dengan nama "`geometry`", yang merupakan sebuah *geometry object* atau `null`.
2. Objek ini harus memiliki anggota dengan nama "`properties`", yang merupakan sebuah objek bebas atau `null`.
3. Jika objek ini diasosiasikan dengan sebuah *identifier* (penanda), penanda tersebut harus diikutsertakan sebagai anggota dari objek ini dengan nama "`id`".

Berikut adalah contoh dari objek bertipe *Feature*:

```
{
    "type": "Feature",
    "id": 1434610833,
    "properties": {
        "name": "Jalan Ciumbuleuit and Jalan Menjangan",
        "city": "Bandung"
    },
    "geometry": {
        "type": "MultiLineString",
        "coordinates": [
            [
                [107.60486, -6.88323],
                [107.60417, -6.88182],
                [107.60345, -6.87968],
                [107.60445, -6.87525]
            ],
            [
                [107.60485, -6.87369],
                [107.60563, -6.87381],
                [107.60660, -6.87411],
                [107.60683, -6.87407]
            ]
        ]
    }
}
```



Gambar 2.1: Arsitektur Saat Ini

Contoh di atas menunjukkan sebuah objek *Feature* yang mendeskripsikan Jalan Ciumbuleuit dan Jalan Menjangan, beserta atribut-atributnya.

2.3 Mesin Navigasi KIRI

KIRI memiliki mesin navigasi yang dibangun pada bahasa Java. Mesin ini bertugas untuk menerima masukan berupa koordinat titik asal dan tujuan, kemudian menemukan angkot-angkot yang harus dinaiki untuk menuju titik tujuan dari titik asal. Karena alasan kerahasiaan, pembahasan mengenai mesin navigasi KIRI tidak mengacu pada referensi publik, melainkan dari survei terhadap kode sumber internal KIRI.

Seperti dapat dilihat pada gambar 2.1, elemen arsitektur yang mendukung navigasi KIRI dibagi menjadi tiga, yaitu:

1. **Basis Data KIRI** menyimpan informasi rute 33 trayek angkot, yang masing-masing mencakup identifikasi trayek (`trackId` dan `trackTypeId`), nama trayek (`trackName`), daftar koordinat yang dilewati (`geodata`), informasi pulang-pergi (`pathloop`), catatan internal (`internalInfo`), prioritas untuk dipilih (`penalty`), informasi naik/turun penumpang (`transferNodes`), dan parameter ekstra untuk pembelian tiket (`extraParameters`).

2. **Berkas tracks.conf** adalah hasil ekstraksi dari basis data KIRI, yang menyimpan informasi penting saja yang dibutuhkan oleh algoritma mesin navigasi KIRI, yakni: `trackId`, `trackTypeId`, `penalty`, `geodata`, `pathloop`, dan `transferNodes`.

3. **Mesin Navigasi KIRI** adalah program yang bertugas mengolah data yang ada pada berkas tracks.conf, sehingga dapat menjawab pertanyaan navigasi dari titik asal ke titik tujuan. Karena alasan historis, mesin navigasi tidak membaca data langsung dari basis data, melainkan dari berkas tracks.conf.

Ketiga elemen tersebut dijelaskan pada subbab-subbab berikut.

Tabel 2.1: Struktur Tabel tracks

Nama kolom	Tipe	Keterangan
trackId	varchar(32)	Kode trayek angkot (misal: "`sthallciumbuleuitlurus`"). Menjadi PRIMARY KEY tabel bersama trackTypeId.
trackTypeId	varchar(32)	Kode tipe trayek (untuk angkot bandung, selalu berisi "`bdo_angkot`"). Menjadi PRIMARY KEY bersama trackId.
trackName	varchar(32)	Nama trayek yang dapat dibaca secara umum (misal: "`St. Hall - Ciumbuleuit (lurus)`").
internalInfo	varchar(1024)	Informasi internal yang dapat ditambahkan dan tidak akan ditampilkan pada hasil pencarian.
geodata	linestring	Daftar koordinat dari rute trayek ini.
pathloop	tinyint(1)	Menandakan apakah trayek ini adalah trayek pulang pergi atau satu arah.
penalty	decimal(4,2)	Bobot dari trayek ini. Semakin besar nilainya, semakin kecil kemungkinan akan muncul pada hasil navigasi.
transferNodes	varchar(1024)	Daftar indeks titik di mana penumpang dapat turun maupun naik, dipisahkan dengan koma. Untuk angkot Bandung, penumpang dapat turun dan naik di semua titik.
extraParameters	varchar(256)	Informasi yang ditambahkan saat melakukan pembelian tiket (tidak terkait dengan penelitian ini).

2.3.1 Basis Data KIRI

Basis data KIRI disimpan dalam sistem manajemen basis data MySQL. Salah satu dari tabel yang digunakan adalah tabel `tracks`, yang menyimpan informasi rute trayek. Struktur dari tabel ini dijelaskan pada tabel 2.1.

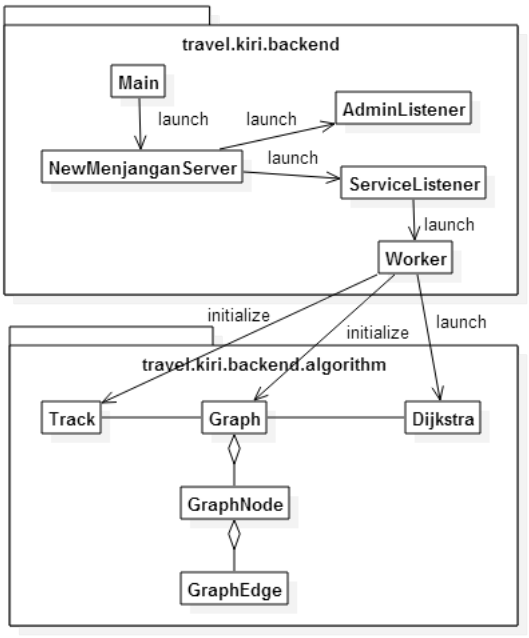
2.3.2 Berkas tracks.conf

Karena alasan historis[2], mesin navigasi KIRI tidak mengakses langsung ke basis data MySQL, melainkan membaca sebuah berkas yang bernama `tracks.conf`.

Berkas `tracks.conf` ini merupakan sebuah berkas teks yang menyimpan basis data trayek yang telah dibersihkan untuk dapat dibaca dengan mudah, satu *record* per baris. Setiap baris berisi 6 *field* yang dipisahkan dengan *tab* (*Unicode* U+0009[3]). Berikut adalah penjelasan dari keenam *field* tersebut:

1. **trackTypeId.trackId** Berisi kolom *trackTypeId* dan *trackId* dipisahkan dengan titik.
2. **penalty** Berisi nilai kolom *penalty*.
3. **numberOfNodes** Berisi jumlah *node* (titik) dari rute trayek ini.
4. **nodes** Berisi daftar koordinat *node* dari rute trayek ini dipisahkan dengan *tab*. Setiap koordinat node terdiri dari *latitude* dan *longitude* yang dipisahkan dengan spasi.
5. **pathLoop** Berisi nilai kolom *pathloop* yang menunjukkan apakah rute ini merupakan rute pulang pergi atau searah.

[2]Pada awalnya mesin navigasi KIRI dibangun menggunakan bahasa C++, sehingga menyulitkan dalam mengakses basis data MySQL



Gambar 2.2: Diagram Kelas Sistem Kini

6. **transferNodes** Berisi nilai kolom *transferNodes* yang menunjukkan titik-titik di mana penumpang diperbolehkan untuk naik atau turun.

2.3.3 Mesin Navigasi KIRI

Mesin navigasi KIRI merupakan sebuah program yang mendengarkan dan menjawab permintaan navigasi dalam bentuk *HTTP request* [4] di *port* 8000. Program ini dibangun di atas bahasa Java, yang terdiri dari beberapa kelas yang ditunjukkan pada gambar 2.2.

Adapun penjelasan untuk setiap kelas dapat dilihat di bawah ini:

Main Kelas yang berfungsi sebagai antarmuka program, untuk dijalankan dari *console*.

NewMenjanganServer Kelas yang bertugas menjalankan program sebagai server, yakni menyiapkan servis-servis untuk dijalankan.

AdminListener Kelas yang berfungsi untuk mendengarkan dan merespon perintah administrasi, seperti *ping*, *shutdown*, dll.

ServiceListener Kelas yang berfungsi untuk mendengarkan dan merespon permintaan untuk navigasi. Servis ini menerima masukan berupa koordinat asal dan tujuan, dan mengembalikan rute angkot yang harus dinaiki.

Worker Kelas ini berfungsi untuk melakukan pekerjaan yang telah diterima oleh ServiceListener. Selain itu, di awal kelas ini juga menyiapkan bahan-bahan yang diperlukan untuk melakukan perhitungan rute, yakni mengkonversi data trayek ke dalam bentuk graf.

Track Kelas ini berfungsi untuk menyimpan informasi sebuah trayek angkot beserta rutenya.

Graph Kelas ini merepresentasikan sebuah graf.

GraphNode Kelas ini merepresentasikan setiap *node* yang dimiliki oleh graf.

GraphEdge Kelas ini merepresentasikan sebuah *edge* dari GraphNode.

Dijkstra Kelas ini merepresentasikan algoritma *Dijkstra's shortest path*[5], untuk digunakan dalam pencarian rute.

2.4 Protokol Peta Angkutan Umum

Berdasarkan kode sumber Peta Angkutan Umum [6], situs tersebut memanfaatkan *HTTP request* [4] untuk menerima perintah-perintah dan mengembalikan respon berupa data dalam sintaks JSON. Beberapa dari perintah tersebut dijelaskan pada subbab-subbab berikut.

2.4.1 Transportation List

Perintah ini digunakan untuk mendapatkan daftar semua daftar transportasi umum, dan dapat diakses pada dengan melakukan *HTTP request GET* pada path `/route/transportation-list.json`, tanpa parameter apapun. Adapun kembalian dari perintah ini adalah sebuah objek JSON yang terdiri dari pasangan nama/nilai berikut:

- `status`: Status kembalian dari perintah yang dikirimkan
- `provinces`: *Array* provinsi yang terdaftar, masing-masing elemen merupakan *array* lain yang berisi:
  - Kode provinsi
  - Nama provinsi
- `transportations`: *Array* provinsi yang terdaftar, masing-masing berisi:
  - `hasroute`: Menyatakan apakah trayek ini memiliki data rute
  - `company`: Perusahaan pengelola trayek ini
  - `id`: Indeks trayek ini dalam basis data
  - `origin`: Awal rute trayek
  - `destination`: Akhir rute trayek
  - `province`: Provinsi tempat trayek ini beroperasi
  - `city`: Kota tempat trayek ini beroperasi
  - `number`: Nomer trayek
  - `created`: *UNIX time* trayek ini dibuat
  - `updated`: *UNIX time* trayek ini terakhir diperbaharui

Contoh kembalian dari perintah transportation list (https://angkot.web.id/route/transportation-list.json) dapat dilihat pada kode di bawah ini:

```
{
    "status": "ok",
    "provinces": [
        [
            "ID-AC",
            "Aceh"
        ],
        ...
    ],
    "transportations": [
        {
            "hasRoute": true,
            "company": "KWK",
            "id": 21,
            "origin": null,
            "destination": null,
            "province": "ID-JK",
            "city": "Jakarta",
            "number": "S15A",
            "created": "1379952348",
            "updated": "1379952379"
        },
        ...
    ]
}
```

2.4.2 Transportation Detail

Perintah ini digunakan untuk mendapatkan detail dari sebuah trayek transportasi, dan diakses dengan melakukan *HTTP request GET* pada path `/route/transportation/nnn.json` (di mana *nnn* adalah nomer kode trayek dalam basis data). Adapun kembalian dari perintah ini adalah:

- `id`: Nomer kode trayek.
- `geojson`: Data rute dan atributnya dalam format GeoJSON, terdiri dari:
  - `type`: Tipe GeoJSON yang digunakan, selalu berisi "`Feature`"
  - `properties`: Properti-properti data GeoJSON, terdiri dari:
    - `number`: Nomer trayek
    - `accept`: *tidak diketahui*
    - `company`: Perusahaan pengelola trayek ini
    - `destination`: Akhir rute trayek
    - `province`: Provisi tempat trayek ini beroperasi
    - `origin`: Awal rute trayek
    - `city`: Kote tempat trayek ini beroperasi
  - `properties`: Data geometri rute trayek ini, mengikuti standar GeoJSON untuk tipe data `MultiLineString`.
- `created`: *UNIX time* trayek ini dibuat
- `status`: Status kembalian dari perintah yang dikirimkan
- `submission_id`: Kode submisi dari trayek yang dimaksud
- `updated`: *UNIX time* trayek ini terakhir diperbaharui

Contoh kembalian dari perintah transportation (https://angkot.web.id/route/transportation/348.json) dapat dilihat pada kode berikut:

```
{
    "id": 348,
    "geojson": {
        "type": "Feature",
        "properties": {
            "number": "M24v2",
            "accept": [],
            "company": "Mikrolet",
            "destination": "Joglo",
            "province": "ID-JK",
            "origin": "Terminal Grogol",
            "city": "Jakarta"
        },
        "geometry": {
            "type": "MultiLineString",
            "coordinates": [
                [
                    [106.79068207740782, -6.166144969433212],
                    [106.79347157478333, -6.166107635752627],
                    [106.79707646369934, -6.165899633769797],
                    ...
                ],
                [
                    [106.7961323261261, -6.196021736671819],
                    [106.7957353591919, -6.196037735916297],
                    [106.79550468921661, -6.195936407359816],
                    ...
                ]
            ]
        }
    },
    "created": "1402504737",
    "status": "ok",
    "submission_id": "CaZVRi",
    "updated": "1402506397"
}
```

BAB 3

ANALISIS

3.1 Mesin Navigasi KIRI

3.1.1 Mekanisme Penarikan

Untuk mendukung integrasi data antara Peta Angkutan Umum dan KIRI, diperlukan adanya penarikan secara berkala terhadap Peta Angkutan Umum oleh KIRI. Ada dua alternatif metode yang dipertimbangkan sebagai mekanisme penarikan data ini:

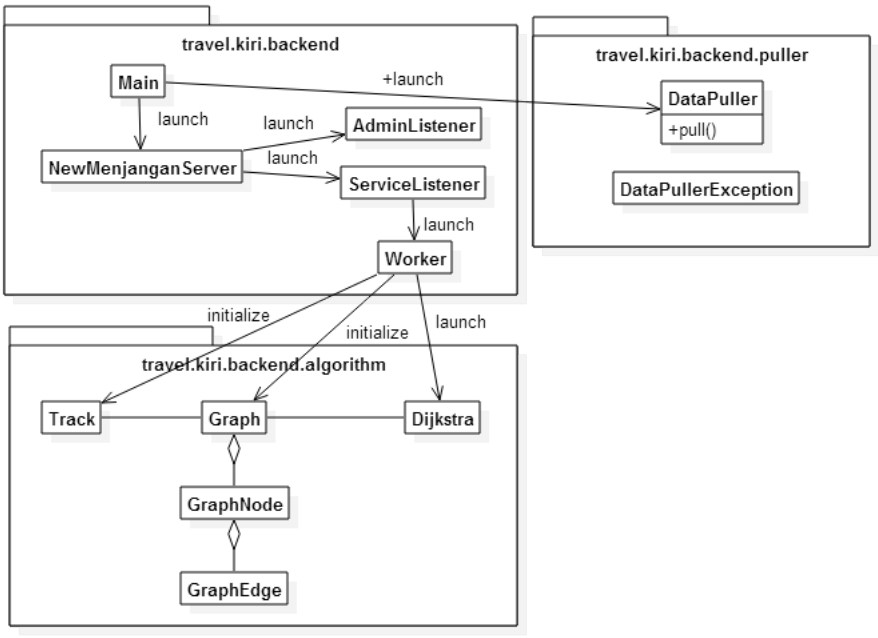
Metode *realtime* mendeteksi setiap perubahan yang terjadi pada data Peta Angkutan Umum, dan langsung memberi notifikasi kepada KIRI untuk mengambil ulang data yang berubah dari Peta Angkutan Umum

Metode *polling* di mana KIRI akan mengambil data secara berkala dari Peta Angkutan Umum, sehingga akan terdapat jeda antara perubahan data dan terbaharuinya data KIRI.

Dari kedua metode tersebut, metode *polling* dipilih dengan alasan-alasan sebagai berikut:

- **Lebih sedikit perubahan** Peta Angkutan Umum menggunakan protokol HTTP yang bersifat *conectionless*, sehingga secara alami akan menunggu perintah dari *client*, alih-alih secara aktif memberi notifikasi kepada *client*. Jika menggunakan metode *polling*, KIRI dapat memanfaatkan protokol *Transportation Detail* yang sudah dimiliki oleh Peta Angkutan Umum.
- **Mengurangi kebutuhan prosesor** Rute pada KIRI dimodelkan dalam bentuk graf, dan sebelum dapat digunakan untuk pencarian graf ini harus dikonstruksi terlebih dahulu. Akibat besarnya data yang digunakan, konstruksi ini akan memakan waktu (kurang lebih 30 detik). Dengan menggunakan metode *realtime*, setiap perubahan data pada kiri.web.id akan mengakibatkan graf ini harus direkonstruksi ulang. Dengan menggunakan metode *polling*, penarikan data dan rekonstruksi graf dapat dijadwalkan sedemikian sehingga dilakukan pada saat pengguna sedang tidak aktif.
- **Urgensi Perbaikan Data** Urgensi perbaikan data tidak terlalu tinggi untuk KIRI (jika dibandingkan dengan perubahan harga saham, *realtime GPS monitoring*, dll) sehingga penarikan tidak perlu dilakukan terlalu sering.

Ditetapkan penggunaan metode *polling* selama 24 jam sekali, dan dipilih waktu 0.30 (setengah jam setelah tengah malam) untuk melakukan *polling*. Penetapan waktu ini mempertimbangkan



Gambar 3.1: Diagram Kelas Sistem Usulan (Tahap Analisis)

sedikitnya penggunaan KIRI pada saat tengah malam. Jeda setengah jam dilakukan untuk memberikan toleransi terhadap perbedaan waktu komputer beberapa detik, yang mungkin memberikan masalah pada tanggal sistem.

3.1.2 Penyimpanan Data

Penyimpanan data diusahakan untuk meminimalisir perubahan serta menjaga kompatibilitas dengan versi sebelumnya. Seperti yang telah dibahas pada bab 2, mesin navigasi KIRI menggunakan berkas `tracks.conf` sebagai jembatan dari basis data menuju mesin navigasi. Berkas inilah yang akan dikonstruksi menjadi graf oleh kelas `Worker`. Peneliti memutuskan untuk tidak mengubah format dari berkas ini, melainkan melakukan modifikasi pada basis data.

Pada basis data, tetap diusahakan untuk meminimalisir perubahan. Dari struktur tabel `tracks` yang digunakan, diidentifikasi bahwa kolom `internalInfo` tidak digunakan dalam perhitungan (tidak berpengaruh pada konstruksi graf). Oleh sebab itu, kolom ini menjadi kandidat untuk disisipkan informasi terkait dengan penarikan data dari Peta Angkutan Umum. Adapun informasi yang harus disimpan adalah:

- Penanda bahwa rute ini ditarik dari Peta Angkutan Umum.
- Kode yang mengacu pada basis data Peta Angkutan Umum, dalam hal ini `id`.

3.1.3 Analisis Aplikasi

Pada penelitian ini, mesin navigasi KIRI dimodifikasi dengan menambahkan dua kelas baru, yaitu kelas `DataPuller` yang difungsikan sebagai penarik data dari Peta Angkutan Umum, serta kelas `DataPullerException` untuk mencatat kesalahan pada saat menarik data dari Peta Angkutan Umum. Kelas-kelas yang baru serta yang lama digambarkan pada diagram kelas di gambar 3.1.

3.2 Peta Angkutan Umum

Pada dasarnya Peta Angkutan Umum tidak diperlukan adanya perubahan, karena KIRI dapat memanfaatkan protokol *transportartion detail*. Namun, untuk keperluan optimasi, ada sedikit perubahan yang dijelaskan pada subbab berikutnya.

3.3 Optimasi

Dengan mekanisme yang sudah dijelaskan di subbab sebelumnya, data pada KIRI dapat tersinkronisasi dengan Peta Angkutan Umum. Namun, mekanisme tersebut dirasa tidak optimal. Setiap pukul 0.30, KIRI akan menarik ke-33 rute angkot yang dicatat pada Peta Angkutan Umum, terlepas dari apakah ada perubahan pada rute angkot atau tidak. Walaupun hanya menarik 33 rute dan dilakukan pada jam tidak sibuk, peneliti merasa optimasi tetap dibutuhkan sehingga solusi ini skalabel.

Mekanisme *Transportation List* maupun *Transportation Detail* pada Peta Angkutan Umum memberikan informasi *updated* yang menunjukkan waktu terakhir rute ini diperbaharui. Pada saat menarik data dari Peta Angkutan Umum, informasi ini dapat dicatat dan disimpan pada basis data, untuk dibandingkan pada kesempatan berikutnya. Jika informasi *updated* yang terdapat pada basis data sama atau lebih baru dibandingan dengan yang didapat dari Peta Angkutan Umum, maka tidak diperlukan adanya penarikan rute dari Peta Angkutan Umum. Adapun informasi yang dicatat pada basis data KIRI menjadi sebagai berikut (informasi baru hasil optimasi **dicetak tebal**):

- Penanda bahwa rute ini ditarik dari Peta Angkutan Umum
- Kode yang mengacu pada basis data Peta Angkutan Umum, dalam hal ini `id`.
- **Waktu terakhir rute ini berubah pada Peta Angkutan Umum**

Perlu dicatat pula bahwa tidak semua data trayek di Peta Angkutan Umum akan diintegrasikan dengan KIRI. Oleh karena itu, dibutuhkan sedikit perubahan pada Peta Angkutan Umum, sehingga KIRI dapat menarik informasi untuk rute-rute yang dibutuhkan saja. Detail dari perubahan ini akan dijelaskan pada bab berikutnya.

BAB 4

PERANCANGAN

4.1 Perancangan Aplikasi Mesin Navigasi KIRI

Seperti telah dibahas pada bab analisis, ada beberapa penambahan kelas pada mesin navigasi KIRI. Adapun kelas utama yang ditambahkan adalah kelas `DataPuller` yang bertanggung jawab untuk menarik data dari Peta Angkutan Umum. Kelas ini memiliki beberapa *method* antara lain:

- **public void (File sqlPropertiesFile, PrintStream output)**
  Berfungsi untuk memeriksa seluruh trayek di basis data. Untuk trayek yang memenuhi syarat (terintegrasi dengan Peta Angkutan Umum dan terdapat data yang lebih baru di Peta Angkutan Umum), melakukan penarikan.
  Parameter:
  - **sqlPropertiesFile** menunjukkan berkas yang menyimpan konfigurasi dari basis data yang akan digunakan.
  - **output** tempat di mana hasil dari penarikan akan ditulis (pada umumnya akan ditulis ke berkas tracks.conf).

  Kembalian: tidak ada.

- **private static LngLatAlt[] lineStringToLngLatArray(String wktText)**
  Method bantuan untuk mengubah teks dari format *Well-known text* [7] menjadi *array latitude* dan *longitude*. Hal ini diperlukan karena MySQL mengembalikan data rute dalam format *Well-known text* tersebut, sedangkan pemrosesan dalam bahasa Java lebih mudah jika datanya memiliki struktur.
  Parameter:
  - **wktText** Teks yang berisi data rute dalam format *Well-known text*

  Kembalian: rute dalam *array latitude* dan *longitude*

- **private static double computeDistance(LngLatAlt p1, LngLatAlt p2)**
  Method bantuan untuk menghitung jarak dari dua buah titik dalam format *latitude* dan *longitude*. Perhitungannya menggunakan metode *haversine*[1].
  Parameter:

[1] http://www.movable-type.co.uk/scripts/latlong.html

  - **p1** titik pertama
  - **p2** titik kedua

  Kembalian: jarak kedua titik tersebut dalam kilometer

- **private RouteResult formatTrack(String trackTypeId, String trackId, LngLatAlt[] geodata, boolean isPathLoop, String penalty, String transferNodesStr, int lastUpdate))**
  Mengkonversikan sebuah data trayek dalam format yang digunakan pada berkas tracks.conf. **Parameter:**

  - **trackTypeId** kode tipe trayek
  - **trackId** kode trayek angkot
  - **geodata** titik kedua
  - **isPathLoop** titik kedua
  - **penalty** titik kedua
  - **transferNodeStr** titik kedua
  - **lastUpdate** titik kedua

  Kembalian: informasi rute dalam format berkas tracks.conf

- **private RouteResult formatTrackFromAngkotWebId(String angkotId, String trackTypeId, String trackId)**
  Mengkonversikan data dari situs Peta Angkutan Umum menjadi format yang digunakan pada berkas tracks.conf. **Parameter:**

  - **angkotId** kode angkot pada Peta Angkutan Umum
  - **trackTypeId** kode tipe trayek
  - **trackId** kode trayek angkot

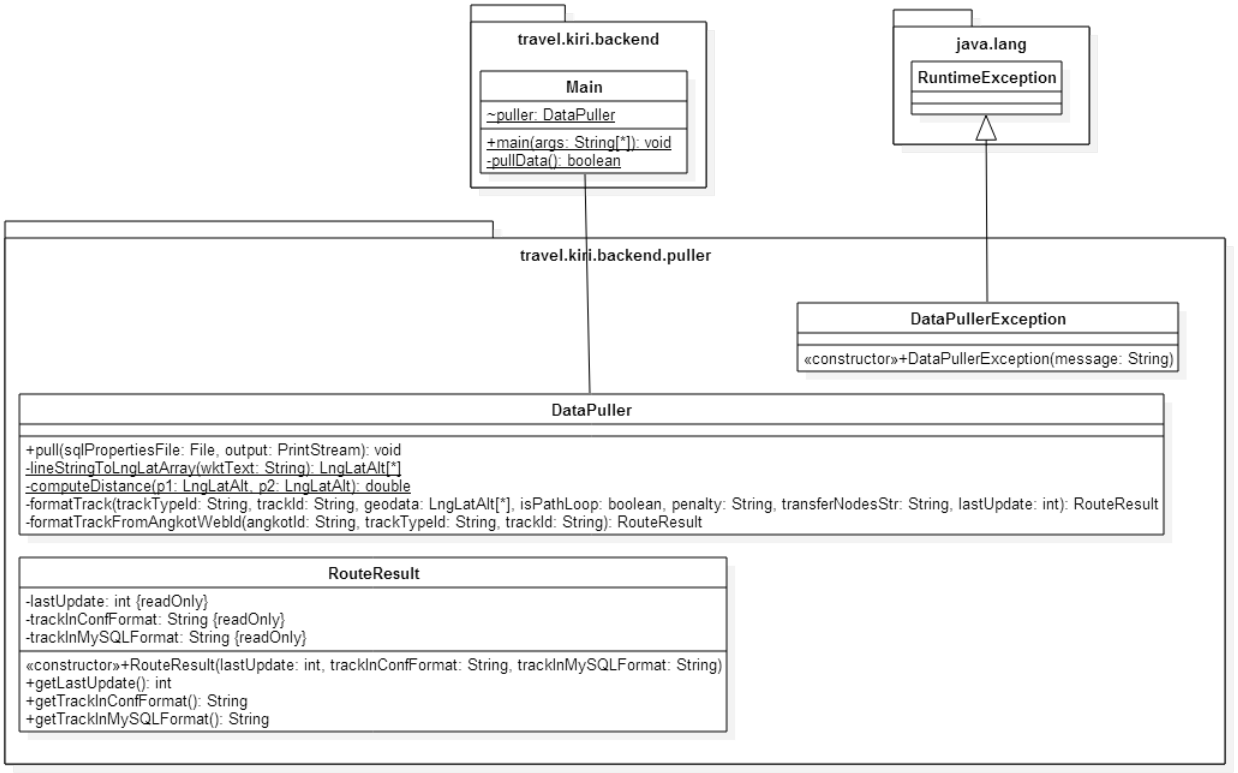
  Kembalian: informasi rute dalam format berkas tracks.conf

Selain itu, peneliti juga membuat kelas lain `RouteResult` yang merupakan *inner class* dari `DataPuller`, untuk menampung hasil dari pembuatan rute serta pembacaan rute dari Peta Angkutan Umum. Kelas ini memiliki beberapa atribut antara lain:

- int **lastUpdate**
  Menyimpan tanggal terakhir rute ini diperbaharui di Peta Angkutan Umum dalam format UNIX time.

- String **trackInConfFormat**
  Menyimpan representasi rute ini dalam format berkas tracks.conf.

- String **trackInMySQLFormat**
  Menyimpan representasi rute ini dalam format query MySQL.



Gambar 4.1: Diagram Kelas (Tahap Desain)

Kelas `RouteResult` tidak memiliki method kecuali konstruktor dan *getter*.

Terakhir, ditambahkan pula kelas `DataPullerException` untuk mencatat segala eksepsi pada saat menarik data dari Peta Angkutan Umum.

Detail seluruh kelas yang ditambahkan dapat dilihat pada kelas diagram pada gambar 4.1.

4.2 Perancangan Protokol

Protokol untuk melakukan sinkronisasi dibuat di atas protokol *Transportation List* dan *Transportation Detail* milik situs Peta Angkutan Umum. Di awal sinkronisasi, mesin navigasi KIRI mencatat trayek apa saja yang harus disinkronkan dengan Peta Angkutan Umum. Kemudian, mesin navigasi KIRI akan mengirimkan permintaan *Transportartion List* dengan menyertakan parameter id/kode angkot Peta Angkutan Umum, yang dipisahkan dengan *pipe* ("|"). Tambahan parameter ini merupakan hasil dari optimasi protokol yang sudah ada, sehingga jawaban yang dikirimkan hanya mencakup angkot yang diperlukan oleh KIRI saja.

Contohnya, permintaan *Transportation List* yang meminta status dari angkot dengan kode 1 dan 2 saja menggunakan perintah `GET /route/transportation-list.json?id=1|2`. Hasil dari permintaan tersebut akan menghasilkan kembalian kurang lebih seperti berikut:

```
{
   "transportations":[
      {
         "city":"Jakarta",
         "id":1,
         "updated":"1381097188",
         "number":"M17",
         "province":"ID-JK",
         "company":"Mikrolet",
```

```
10            "created":"1376493332",
11            "destination":"Pasar Lenteng Agung",
12            "origin":"Pasar Minggu",
13            "hasRoute":true
14        },
15        {
16            "city":"Jakarta",
17            "id":2,
18            "updated":"1379737743",
19            "number":"S616",
20            "province":"ID-JK",
21            "company":"Kopaja",
22            "created":"1376494541",
23            "destination":"Cipedak",
24            "origin":"Blok M",
25            "hasRoute":true
26        }
27    ],
28    "status":"ok",
29    "provinces":[
30        [
31            "ID-AC",
32            "Aceh"
33        ],
34        ...
35     ]
36 }
```

Dari kembalian di atas, mesin navigasi KIRI dapat mengetahui kapan rute di Peta Angkutan Umum terakhir diperbaharui, untuk trayek-trayek yang diminta. Untuk setiap rute yang telah berubah, KIRI mengirimkan lagi perintah berikutnya, yaitu *Transportation Detail*, yang memberikan rute penuh untuk trayek yang diminta. Sebagai contoh, jika rute dengan kode 1 ditemukan telah berubah, maka akan dikirimkan perintah *Transportation Detail* `GET /route/transportation/1.json`. Dari situ, rute lengkap akan disimpan pada berkas `tracks.conf`.

4.3 Perancangan Basis Data

Pada bab 3, telah disimpulkan bahwa dalam kolom `internalInfo` pada basis data perlu menampung informasi-informasi sebagai berikut:

- Penanda bahwa rute ini ditarik dari Peta Angkutan Umum
- Kode yang mengacu pada basis data Peta Angkutan Umum, dalam hal ini `id`.
- Waktu terakhir rute ini berubah pada Peta Angkutan Umum

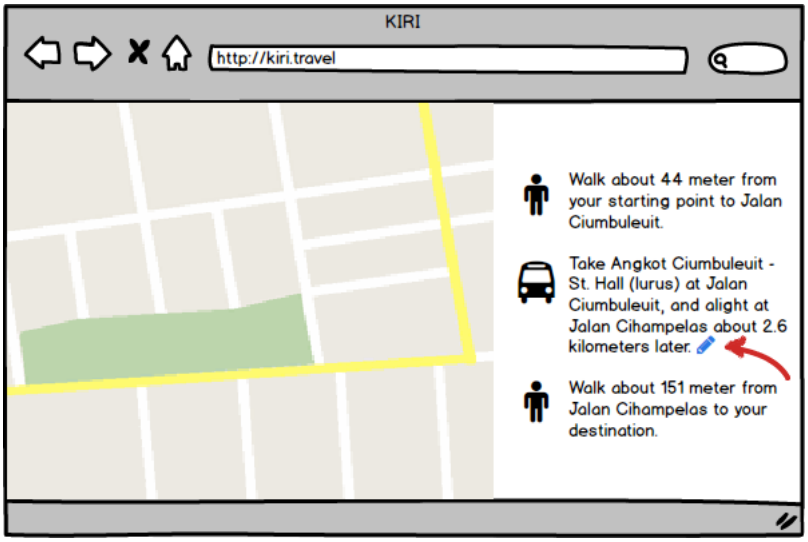
Untuk itu, ditetapkan bahwa sebuah rute angkot yang terintegrasi dengan data pada Peta Angkutan Umum akan memiliki kolom `internalInfo` yang berisi:

angkotwebid:*id*:*updated*

Dengan kata lain, teks yang berisi tiga *token* terpisah tanda titik dua (“:”) yang terdiri dari:

- *String* “angkotwebid” sebagai penanda bahwa rute ini ditarik dari Peta Angkutan Umum
- *id* berupa informasi *id* transportasi yang didapat dari Peta Angkutan Umum.
- *updated* berupa informasi waktu terakhir rute ini diperbaharui, seperti didapat dari informasi *updated* dari Peta Angkutan Umum.

Dengan spesifikasi tersebut, mesin navigasi KIRI dapat menentukan apakah perlu menarik data dari Peta Angkutan Umum.



Gambar 4.2: Tombol ubah di situs web KIRI

4.4 Perancangan Antarmuka

4.4.1 Antarmuka Mesin Navigasi KIRI

Mesin navigasi KIRI adalah program yang dijalankan sebagai server, sehingga hanya memiliki antarmuka minimal berbasis teks yang menampilkan aksi-aksi yang dilakukan oleh server. Setelah ditambahkan fitur menarik data dari Peta Angkutan Umum, maka akan ada tambahan penampilan aksi menarik rute seperti contoh berikut (tambahan ada pada baris 1-8):

```
1  May 20, 2015 1:56:02 PM travel.kiri.backend.puller.DataPuller pull
2  INFO: Fetching https://angkot.web.id/route/transportation-list.json?id=157|247|636...
3  May 20, 2015 1:56:11 PM travel.kiri.backend.puller.DataPuller formatTrackFromAngkotWebId
4  INFO: Fetching bdo_angkot.cicaheumciroyom from https://angkot.web.id/route/transportation/157.json...
5  May 20, 2015 1:56:17 PM travel.kiri.backend.puller.DataPuller formatTrackFromAngkotWebId
6  INFO: Fetching bdo_angkot.cicaheumledeng from https://angkot.web.id/route/transportation/247.json...
7  May 20, 2015 1:56:19 PM travel.kiri.backend.puller.DataPuller formatTrackFromAngkotWebId
8  INFO: Fetching bdo_angkot.ciroyomantapani from https://angkot.web.id/route/transportation/636.json...
9  May 20, 2015 1:56:37 PM travel.kiri.backend.Worker <init>
10 INFO: Configuration were read successfully
11 May 20, 2015 1:56:41 PM travel.kiri.backend.Worker <init>
12 INFO: Tracks were read successfully
13 May 20, 2015 1:57:04 PM travel.kiri.backend.Worker <init>
14 INFO: Tracks were linked successfully
15 2015-05-20 13:57:07.356:INFO::main: Logging initialized @65987ms
16 2015-05-20 13:57:10.173:INFO:oejs.Server:main: jetty-9.2.3.v20140905
17 2015-05-20 13:57:11.483:INFO:oejs.ServerConnector:main: Started ServerConnector@2996771e{HTTP
       /1.1}{0.0.0.0:8000}
18 2015-05-20 13:57:11.485:INFO:oejs.Server:main: Started @70398ms"
```

4.4.2 Antarmuka Situs Web KIRI

Pada situs web KIRI, jika ditemukan rute yang dihasilkan terintegrasi dengan Peta Angkutan Umum, maka situs akan menambahkan sebuah tombol kecil berbentuk pensil, yang jika diklik akan membawa pengguna ke situs Peta Angkutan Umum untuk memodifikasi rute terkait. Tampilan tombol tersebut dapat dilihat di gambar 4.2 (diberi panah merah).

BAB 5

IMPLEMENTASI DAN PENGUJIAN

5.1 Implementasi

5.1.1 Lingkungan Implementasi

Implementasi dilakukan di dua buah mesin. Implementasi pertama dilakukan pada komputer peneliti untuk keperluan pengujian fungsional. Komputer tersebut memiliki spesifikasi seperti berikut:

- Prosesor: 1.80 GHz
- RAM: 4.00 GB
- Sistem Operasi: Windows 8.1 Pro (64-bit)
- Versi Java: 1.7.0_51 (Oracle)

Implementasi kedua dilakukan pada sebuah mesin virtual yang dibuat pada *platform Windows Azure* untuk keperluan pengujian eksperimental. Mesin virtual ini terhubung ke internet serta beroperasi 24 jam sehingga dapat melayani permintaan tanpa henti. Adapun spesifikasi dari mesin virtual ini adalah sebagai berikut:

- Prosesor: 1 core
- RAM: 1.75 GB
- Hard disk: 28.8 GB
- Sistem Operasi: Ubuntu 14.04.2 LTS (64-bit)
- Versi Java: 1.7.0_79 (OpenJDK)

5.1.2 Basis Data

Aplikasi yang dibuat membutuhkan adanya implementasi tabel sesuai dengan yang dijabarkan di tabel 2.1. Dalam penelitian ini, digunakan basis data MySQL[8]. Dalam pengujian eksperimental, aplikasi menggunakan tabel yang sudah ada di basis data KIRI (bersifat rahasia). Untuk pengujian fungsional, tabel baru akan dibuat dengan beberapa data sampel untuk keperluan pengujian saja. Berikut adalah skrip SQL yang digunakan untuk membentuk dan mengisi tabel tersebut:

Tabel 5.1: Hasil Pengujian Fungsional

Nama & deskripsi pengujian	Hasil
testEnsurePullNoException Memastikan tidak ada eksepsi yang terjadi saat operasi pull dijalankan.	berhasil
testSQLUpdatedFromNoTimestamp Memastikan pull berhasil memperbaharui SQL untuk data dengan internalInfo berisi 'angkotwebid:nnn' (tanpa timestamp)	berhasil
testTracksConfUpdatedFromNoTimestamp Memastikan pull berhasil memperbaharui tracks.conf untuk data dengan internalInfo berisi 'angkotwebid:nnn' (tanpa timestamp).	berhasil
testSQLUpdatedFromLowerTimestamp Memastikan pull berhasil memperbaharui SQL untuk data dengan internalInfo berisi 'angkotwebid:nnn:mmm' (dengan timestamp tetapi lebih rendah).	berhasil
testTracksConfUpdatedFromLowerTimestamp Memastikan pull berhasil memperbaharui tracks.conf untuk data dengan internalInfo berisi 'angkotwebid:nnn:mmm' (dengan timestamp tetapi lebih rendah).	berhasil
testSQLNotUpdatedFromHigherTimestamp Memastikan pull berhasil memperbaharui SQL untuk data dengan internalInfo berisi 'angkotwebid:nnn:mmm' (dengan timestamp tetapi lebih tinggi).	berhasil
testTracksConfNotUpdatedFromHigherTimestamp Memastikan pull berhasil memperbaharui tracks.conf untuk data dengan internalInfo berisi 'angkotwebid:nnn:mmm' (dengan timestamp tetapi lebih tinggi).	berhasil
testSQLNotUpdatedFromNotAngkotWebid Memastikan pull tidak memperbaharui SQL untuk data dengan internalInfo berisi 'Not an angkotwebid track' (bukan track yang terintegrasi).	berhasil
testTracksConfUpdatedFromHigherTimestamp Memastikan pull tidak memperbaharui tracks.conf untuk data dengan internalInfo berisi 'Not an angkotwebid track' (bukan track yang terintegrasi).	berhasil

```
1  DROP TABLE IF EXISTS `tracks`;
2  CREATE TABLE IF NOT EXISTS `tracks` (
3      `trackId` varchar(32) NOT NULL,
4      `trackTypeId` varchar(32) NOT NULL DEFAULT 'bdo_angkot',
5      `trackName` varchar(64) NOT NULL,
6      `internalInfo` varchar(1024) NOT NULL,
7      `geodata` linestring DEFAULT NULL,
8      `pathloop` tinyint(1) NOT NULL DEFAULT '0',
9      `penalty` decimal(4,2) NOT NULL DEFAULT '1.00',
10     `transferNodes` varchar(1024) DEFAULT NULL,
11     `extraParameters` varchar(256) DEFAULT NULL,
12     `officialTrackNo` varchar(32) DEFAULT NULL,
13     `officialTrackName` varchar(256) DEFAULT NULL
14 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

5.2 Hasil Pengujian

5.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk memastikan fitur integrasi berjalan sesuai dengan spesifikasi yang telah ditetapkan. Untuk melakukan pengujian fungsional, digunakan *framework* JUnit[9]. Ada 9 tes kasus yang diujikan, dan detail serta hasilnya dapat dilihat di tabel 5.1.

5.2.2 Pengujian Eksperimental

Pengujian eksperimental dilakukan untuk melihat respon pengguna setelah fitur integrasikan diimplementasikan. Pengujian eksperimental dibagi menjadi 3 tahap:

1. Survei *online* terhadap kepuasan kualitas pencarian pengguna, dan potensi keinginan untuk berkontribusi selama satu bulan.
2. Publikasi fitur integrasi perbaikan rute.
3. Survei *online* terhadap kepuasan kualitas pencarian serta keterlibatan pengguna dalam perbaikan rute.

Ketiga tahap tersebut dijelaskan pada subbab-subbab berikut.

Tahap 1: Survei Kepuasan Kualitas Pencarian dan Potensi Keinginan Berkontribusi

Pada tahap ini, peneliti membuat survei *online* memanfaatkan aplikasi Google Docs[1]. Ada 6 pertanyaan yang ditanyakan, yaitu:

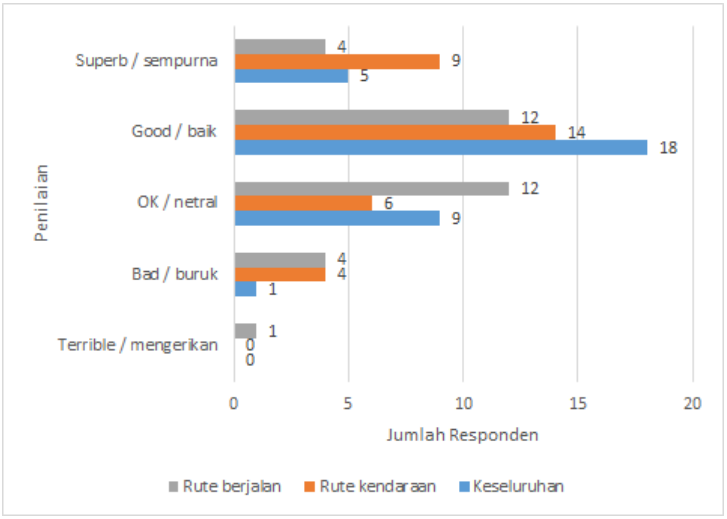
- **Seberapa baik rute kami secara keseluruhan?** (mengerikan / buruk / netral / baik / sempurna)
- **Seberapa baik rute kami pada bagian rute kendaraan?** (mengerikan / buruk / netral / baik / sempurna)
- **Seberapa baik rute kami pada bagian rute berjalan?** (mengerikan / buruk / netral / baik / sempurna)
- **Di kota apa Anda menggunakan KIRI?** (Bandung / Jakarta / Malang / Surabaya)
- **Di mana Anda tinggal?** (Bandung / Jakarta / Malang / Surabaya)
- **Apakah Anda tertarik untuk berkontribusi data rute?** (Ya, dengan imbalan / Ya, tanpa syarat / Tidak / Tidak tahu)

Survei ini disediakan *online* pada alamat `http://goo.gl/forms/WkeU8oK4Ek`, dan ditunjukkan pada situs web KIRI setelah hasil pencarian ditampilkan (gambar 5.1) selama bulan April 2015.

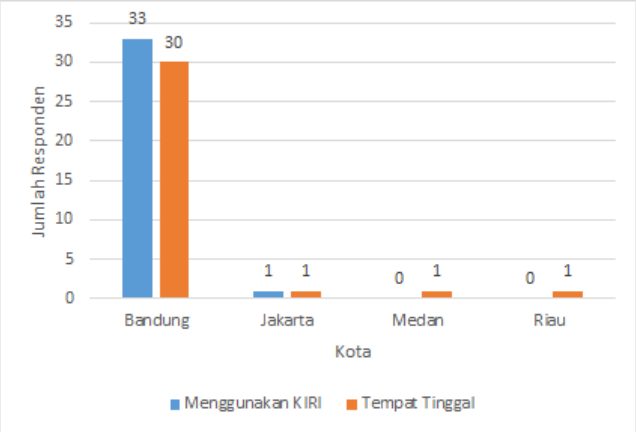
Setelah masa survei berakhir, didapatkan 33 responden survei. Dari segi kualitas pencarian (Gambar 5.2), secara umum pengguna puas dengan hasil kualitas pencarian, ditandai dengan sebagian besar pengguna memilih "baik" pada ketiga jenis pertanyaan. Pada bagian kualitas rute kendaraan, relatif lebih banyak pengguna yang memilih "sempurna" dibandingkan dengan rute berjalan. Diduga hal ini dikarenakan oleh dua faktor:

1. Pencarian rute di KIRI relatif lebih baik dibandingkan dengan pada situs web sejenis[2], karena lebih presisi hingga titik koordinat.
2. Karena keterbatasan data, hasil rute berjalan di KIRI menghasilkan garis lurus dan tidak mengikuti arah jalan yang sesungguhnya.

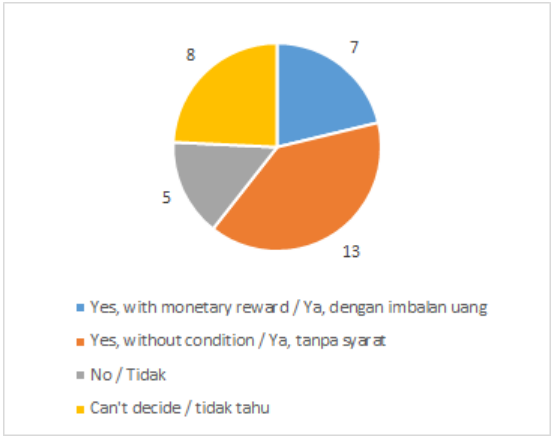
Gambar 5.1: Tautan ke Survei Tahap 1



Gambar 5.2: Hasil Survei Tahap 1 (Kualitas Pencarian)



Gambar 5.3: Hasil Survei Tahap 1 (Demografi Pengguna)



Gambar 5.4: Hasil Survei Tahap 1 (Kesediaan Berkontribusi)

Dari segi demografi pengguna (Gambar 5.3), sebagian besar pengguna menggunakan KIRI di Bandung dan tinggal di Bandung. Hal ini memberikan dua keuntungan kepada peneliti:

1. Sesuai dengan batasan masalah, yaitu penelitian dilakukan di kota Bandung.
2. Pengguna relatif mengenal rute transportasi publik di Bandung, karena tinggal di kota Bandung.

Terakhir, survei mengenai potensi keinginan berkontribusi (Gambar 5.4) memberikan hasil yang bervariasi. Sebagian besar responden bersedia memperbaiki rute yang salah, namun ada juga yang berharap imbalan maupun tidak berminat. Hasil tersebut cukup meyakinkan peneliti bahwa pada tahap berikutnya akan ada sebagian dari pengguna yang bersedia memperbaiki.

Tahap 2: Publikasi fitur integrasi perbaikan rute

Setelah fitur integrasi diimplementasikan, KIRI perlu mendorong penggunanya untuk berpartisipasi dalam perbaikan rute. Di situs KIRI sendiri, sebuah tautan akan diberikan setelah hasil pencarian diberikan. Jika tautan tersebut diklik, maka akan membuka video[3] di situs *Youtube* yang menun-

[1] https://google.com/docs, diakses 1 Juni 2015
[2] http://angkot.tibandung.com/, diakses 1 Juni 2015
[3] https://www.youtube.com/watch?v=jDFePujA8Kk, diakses 10 Juni 2015

Gambar 5.5: Tautan ke Petunjuk Perbaikan

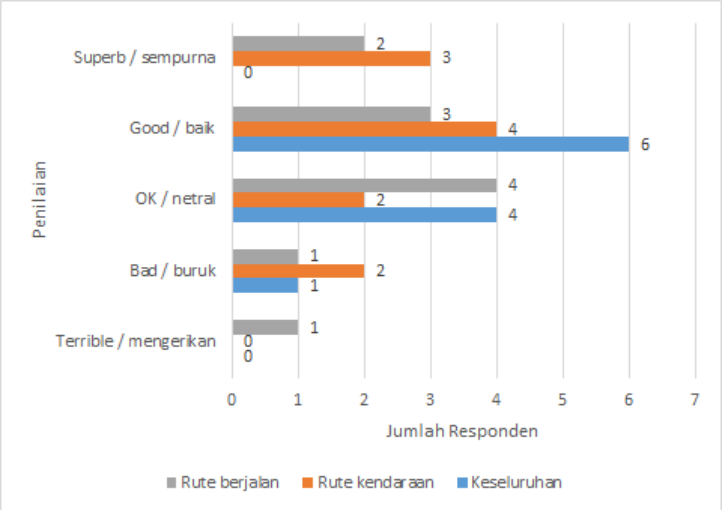
jukkan langkah-langkah dalam memperbaiki rute yang salah di KIRI. Tautan ini terletak di tempat yang sama seperti pada saat survei di tahap 1, dan dapat dilihat pada gambar 5.5.

Selain itu, dilakukan juga kampanye berupa iklan di situs media sosial *Facebook*[4] selama satu bulan. Parameter-parameter yang digunakan untuk iklan ini adalah sebagai berikut:

- Budget, Schedule & Optimization:
  - Budget: Rp 1,500,000 lifetime
  - Schedule: 06/01/2015 - 06/30/2015
  - Duration: 29 days
  - Bidding: Bid for website clicks
  - Pricing: Your bid will be optimized to get more clicks to your website. You'll be charged each time your ad is served.
- Targeting & Placement:
  - Location - Living In: Indonesia / Bandung (+25 mi) West Java
  - Behaviors: Technology early adopters
  - Age: 18 - 65+
  - Language: Indonesian
  - Mobile Placement: Third-party Apps or News Feed
  - Desktop: News Feed or Right Column
- Estimated Daily Reach: 2,800 - 7,400 peoples

Berdasarkan laporan dari *Facebook*, iklan tersebut menghasilkan 2.518 klik dari 31.718 pengguna *Facebook* yang terjangkau (melihat iklan tersebut).

[4] https://www.facebook.com/, diakses 10 Juni 2015



Gambar 5.6: Hasil Survei Tahap 2 (Kualitas Pencarian)

Tahap 3: Survei Kepuasan Kualitas Pencarian dan Keterlibatan Pengguna

Di tahap ini, peneliti membuat survei *online* kedua untuk mengetahui respon pengguna, setelah dilakukan kampanye selama satu bulan. Ada 6 pertanyaan yang ditanyakan, yaitu:

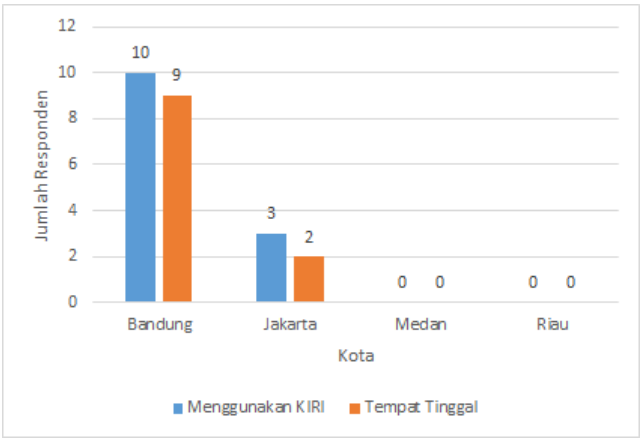
- **Apakah Anda tahu tentang fitur crowdsourcing kami? (di** `https://angkot.web.id`**)** (ya, dan berkontribusi / ya, tidak berkontribusi / tidak tahu)
- **Seberapa baik rute kami sekarang secara keseluruhan?** (mengerikan / buruk / netral / baik / sempurna)
- **Seberapa baik rute kami sekarang pada bagian rute kendaraan?** (mengerikan / buruk / netral / baik / sempurna)
- **Seberapa baik rute kami sekarang pada bagian rute berjalan?** (mengerikan / buruk / netral / baik / sempurna)
- **Di kota apa Anda menggunakan KIRI?** (Bandung / Jakarta / Malang / Surabaya)
- **Di mana Anda tinggal?** (Bandung / Jakarta / Malang / Surabaya)

Survei ini juga dilakukan secara *online* seperti pada tahap 1, dan ditunjukkan pada situs web KIRI setelah hasil pencarian dilakukan. Survei ini dilaksanakan selamat bulan Juli 2015.

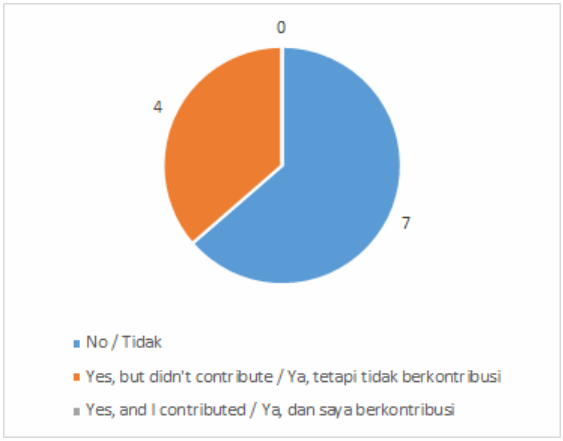
Jumlah responden survei menurun menjadi 11 responden. Dari segi kualitas pencarian (Gambar 5.6), pilihan "baik" masih mendominasi jawaban yang diberikan responden. Pengguna juga relatif memberikan skor yang lebih tinggi untuk rute kendaraan dibanding rute berjalan. Hal ini, sayangnya, menunjukkan bahwa integrasi KIRI dengan Peta Angkutan Umum belum meningkatkan kualitas pencarian secara signifikan.

Demografi pengguna saat ini (Gambar 5.7) sedikit lebih bervariasi jika dibandingkan dengan survei sebelumnya, di mana sekitar 30% pengguna menggunakan KIRI di kota Jakarta.

Dari segi partisipasi (Gambar 5.8), sebagian besar pengguna sudah mengetahui adanya fitur integrasi, sehingga publikasi melalui *Facebook* dan tautan video dirasa sudah cukup efektif dalam



Gambar 5.7: Hasil Survei Tahap 2 (Demografi Pengguna)



Gambar 5.8: Hasil Survei Tahap 2 (Partisipasi)

mengedukasi pengguna. Walau begitu, tidak ada pengguna yang berpartisipasi dalam perbaikan data. Hal ini bisa disebabkan oleh dua hal:

1. Rute yang ada sudah cukup baik.
2. Proses memperbaiki terlalu sulit.

Dalam kasus ini, penyebab pertama (rute cukup baik) dirasa lebih masuk akal, mengingat data rute angkot kota Bandung di KIRI sebagian besar didapatkan dari survei lapangan.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dari penelitian yang telah dilakukan, didapatkanlah kesimpulan-kesimpulan sebagai berikut:

1. Telah berhasil diimplementasikan pengunduhan data otomatis oleh KIRI terhadap Peta Angkutan Umum secara berkala, setiap hari pukul 0.30. Pengunduhan data dilakukan dengan menggunakan *HTTP Request* dan format data *JSON* dan *GeoJSON*.

2. Telah berhasil diimplementasikan pemisahan data antara rute milik KIRI dengan data yang ditarik dari Peta Angkutan Umum. Dalam basis data KIRI, hal ini dicatat dalam kolom "internalInfo" dengan format "angkotwebid:*id-angkutan-umum*:*waktu-terakhir-berubah*".

3. Protokol yang digunakan telah berhasil dioptimasi, dengan mencatat waktu terakhir sebuah rute diunduh. Dengan begitu, hanya rute yang berubah sejak pengunduhan terakhir saja yang diunduh kembali.

4. Pengguna KIRI cukup puas dengan kualitas navigasi dari KIRI (mayoritas menjawab baik), dengan tingkat kepuasan rute kendaraan lebih tinggi dibanding rute berjalan. Pengguna KIRI masih belum berpartisipasi aktif dalam perbaikan rute.

6.2 Saran

Dari hasil penelitian termasuk kesimpulan yang didapat, berikut adalah beberapa saran untuk pengembangan:

1. Pada penelitian ini pengguna kurang berpartisipasi dalam perbaikan rute, sehingga sulit untuk menarik kesimpulan dari fitur integrasi. Untuk pengembangan, disarankan untuk mengulang penelitian di kota yang lebih cocok: rute yang belum akurat, namun penduduknya memiliki semangat untuk berkontribusi.

2. Penelitian ini memanfaatkan kolom "internalInfo" yang awalnya tidak didesain untuk menampung informasi penanda rute dari Peta Angkutan Umum. Dalam pengembangan perangkat lunak yang baik, ada baiknya penanda tersebut dipisahkan sehingga tidak mengganggu fungsi awal dari kolom "internalInfo" itu sendiri, yaitu menyimpan informasi yang dapat dimanfaatkan oleh administrator.

DAFTAR REFERENSI

[1] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format." RFC 7159 (Proposed Standard), Mar. 2014.

[2] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and C. Schmidt, "The GeoJSON Format Specification." http://geojson.org/geojson-spec.html, 2008. [Online; diakses 31-Maret-2015].

[3] Unicode, Inc., "The Unicode Standard, Version 8.0: C0 Controls and Basic Latin." http://www.unicode.org/charts/PDF/U0000.pdf, 2015. [Online; diakses 24-Juni-2015].

[4] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content." RFC 7231 (Proposed Standard), June 2014.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (Second ed.)*. MIT Press and McGraw-Hill, 2001.

[6] F. I. Rusadi, "Peta Angkutan Umum." https://github.com/angkot/angkot, 2015. [Online; diakses 30-Maret-2015].

[7] J. R. Herring, *OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. Open Geospatial Consortium Inc., May 2011.

[8] Oracle Corporation, "MySQL :: The world's most popular open source database." https://www.mysql.com/, 2015. [Online; diakses 4-Juni-2015].

[9] D. Saff, K. Cooney, S. Birkner, and M. Phillipp, "JUnit." http://junit.org/, 2014. [Online; diakses 9-Juni-2015].

LAMPIRAN A

KODE PROGRAM DATA PULLER

Listing A.1: DataPuller.java

```
1  package travel.kiri.backend.puller;
2
3  import java.io.File;
4  import java.io.FileReader;
5  import java.io.IOException;
6  import java.io.PrintStream;
7  import java.net.URL;
8  import java.sql.Connection;
9  import java.sql.DriverManager;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.sql.Statement;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.Properties;
16 import java.util.SortedMap;
17 import java.util.TreeMap;
18
19 import org.geojson.Feature;
20 import org.geojson.LngLatAlt;
21 import org.geojson.MultiLineString;
22
23 import travel.kiri.backend.Main;
24
25 import com.fasterxml.jackson.core.JsonFactory;
26 import com.fasterxml.jackson.core.JsonParser;
27 import com.fasterxml.jackson.core.JsonToken;
28 import com.fasterxml.jackson.databind.JsonNode;
29 import com.fasterxml.jackson.databind.ObjectMapper;
30
31 public class DataPuller {
32     public static final double EARTH_RADIUS = 6371.0;
33     public static final Double MAX_DISTANCE = 0.1;
34     public static final String ANGKOTWEBID_URL = "https://angkot.web.id/route/transportation/%s.json";
35     public static final String ANGKOTWEBID_ROUTELIST_PREFIX = "https://angkot.web.id/route/transportation-
           list.json?id=";
36     public static final int ANGKOTWEBID_MAX_ROUTELIST = 200;
37     public static final String DEFAULT_PENALTY = "0.05";
38     public static final double MAX_LINK_DISTANCE = 0.5;
39
40     public void pull(File sqlPropertiesFile, PrintStream output)
41             throws IOException, SQLException {
42         Properties sqlProperties = new Properties();
43         sqlProperties.load(new FileReader(sqlPropertiesFile));
44         Connection connection = null;
45         connection = DriverManager.getConnection(String.format(
46                 "jdbc:mysql://%s/%s?user=%s&password=%s",
47                 sqlProperties.get("host"), sqlProperties.get("database"),
48                 sqlProperties.get("user"), sqlProperties.get("password")));
49
50         // Look for angkot.web.id refreshes
51         Statement statement = connection.createStatement();
52         ResultSet result = statement.executeQuery("SELECT trackTypeId, trackId, AsText(geodata),
               internalInfo FROM tracks");
53         SortedMap<Integer, AngkotWebIdCacheInfo> obsoleteRoutesMap = new TreeMap<Integer,
               AngkotWebIdCacheInfo>();
54         while (result.next()) {
55             if (result.getString(4).startsWith("angkotwebid:")) {
56                 String[] fields = result.getString(4).split(":");
57                 int id = Integer.parseInt(fields[1]);
58                 long lastUpdate = fields.length > 2 ? Long.parseLong(fields[2]) : 0;
59                 obsoleteRoutesMap.put(id, new AngkotWebIdCacheInfo(result.getString(1), result.getString
                       (2), lastUpdate, id, result.getString(3) != null));
60             }
61         }
62         List<Integer> obsoleteRoutesList = new ArrayList<Integer>(obsoleteRoutesMap.keySet());
63         for (int i = 0; i < (obsoleteRoutesList.size() + ANGKOTWEBID_MAX_ROUTELIST - 1) /
               ANGKOTWEBID_MAX_ROUTELIST; i++) {
64             StringBuilder urlBuilder = new StringBuilder(ANGKOTWEBID_ROUTELIST_PREFIX);
65             for (int j = i * 250; j < Math.min((i + 1) * 250, obsoleteRoutesList.size()); j++) {
66                 urlBuilder.append(j > i * 250 ? "|" : "");
67                 urlBuilder.append(obsoleteRoutesList.get(j));
```

```
68              }
69              Main.globalLogger.info("Fetching " + urlBuilder + "...");
70              ObjectMapper mapper = new ObjectMapper();
71              JsonNode parent = mapper.readTree(new URL(urlBuilder.toString()));
72              JsonNode transportations = parent.get("transportations");
73              for (int j = 0; j < transportations.size(); j++) {
74                  JsonNode transportation = transportations.get(j);
75                  int updated = transportation.get("updated").asInt();
76                  int id = transportation.get("id").asInt();
77                  if (obsoleteRoutesMap.get(id).pathAvailable && obsoleteRoutesMap.get(id).lastUpdate >=
                        updated) {
78                      obsoleteRoutesMap.remove(id);
79                  }
80              }
81          }
82
83          statement = connection.createStatement();
84          result = statement
85                  .executeQuery("SELECT trackTypeId, trackId, AsText(geodata), pathloop, penalty,
                        transferNodes, internalInfo FROM tracks ORDER BY trackTypeId, trackId");
86
87          while (result.next()) {
88              RouteResult routeResult;
89              if (result.getString(7) != null
90                      && result.getString(7).startsWith("angkotwebid:")) {
91                  String[] fields = result.getString(7).split(":");
92                  if (obsoleteRoutesMap.containsKey(Integer.parseInt(fields[1]))) {
93                      routeResult = formatTrackFromAngkotWebId(
94                              fields[1], result.getString(1), result.getString(2));
95                      if (routeResult != null) {
96                          Statement updateStatement = connection.createStatement();
97                          String sql = String
98                                  .format("UPDATE tracks SET internalInfo='%s', geodata=%s WHERE trackTypeId
                                        ='%s' AND trackId='%s'",
99                                          fields[0] + ':' + fields[1]
100                                                 + ':' + routeResult.lastUpdate,
101                                         routeResult.getTrackInMySQLFormat(),
102                                         result.getString(1),
103                                         result.getString(2));
104                         updateStatement
105                                 .execute(sql);
106                     }
107                 } else {
108                     routeResult = formatTrack(result.getString(1), result
109                             .getString(2), lineStringToLngLatArray(result
110                             .getString(3)), result.getString(4).equals("1") ? true
111                             : false, result.getString(5), result.getString(6), 0);
112                 }
113                 output.println(routeResult.getTrackInConfFormat());
114             } else if (result.getString(3) != null) {
115                 routeResult = formatTrack(result.getString(1), result
116                         .getString(2), lineStringToLngLatArray(result
117                         .getString(3)), result.getString(4).equals("1") ? true
118                         : false, result.getString(5), result.getString(6), 0);
119                 output.println(routeResult.getTrackInConfFormat());
120             } else {
121                 throw new DataPullerException("Route not found everywhere for "
122                         + result.getString(1) + "." + result.getString(2));
123             }
124         }
125
126         result.close();
127         statement.close();
128         connection.close();
129     }
130
131     private static LngLatAlt[] lineStringToLngLatArray(String wktText) {
132         wktText = wktText.replace("LINESTRING(", "").replace(")", "");
133         String[] textCoordinates = wktText.split(",");
134         LngLatAlt[] coordinates = new LngLatAlt[textCoordinates.length];
135         for (int i = 0; i < textCoordinates.length; i++) {
136             String[] textLonlat = textCoordinates[i].split(" ");
137             LngLatAlt coordinate = new LngLatAlt(
138                     Double.parseDouble(textLonlat[0]),
139                     Double.parseDouble(textLonlat[1]));
140             coordinates[i] = coordinate;
141         }
142         return coordinates;
143     }
144
145     private static double computeDistance(LngLatAlt p1, LngLatAlt p2) {
146         double lat1 = p1.getLatitude(), lon1 = p1.getLongitude();
147         double lat2 = p2.getLatitude(), lon2 = p2.getLongitude();
148         double dLat = Math.toRadians(lat2 - lat1);
149         double dLon = Math.toRadians(lon2 - lon1);
150         lat1 = Math.toRadians(lat1);
151         lat2 = Math.toRadians(lat2);
152         double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) + Math.sin(dLon / 2)
153                 * Math.sin(dLon / 2) * Math.cos(lat1) * Math.cos(lat2);
154         double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
155         return EARTH_RADIUS * c;
156     }
157
158     private RouteResult formatTrack(String trackTypeId, String trackId,
159             LngLatAlt[] geodata, boolean isPathLoop, String penalty,
160             String transferNodesStr, int lastUpdate) {
161
162         // Setup track info
163         LngLatAlt[] tracks = geodata;
```

```
164         int[][] transitNodes;
165         if (transferNodesStr == null || transferNodesStr.length() == 0) {
166             transitNodes = new int[][] { { 0, tracks.length - 1 } };
167         } else {
168             String[] transitNodesString = transferNodesStr.split(",");
169             transitNodes = new int[transitNodesString.length][2];
170             for (int i = 0; i < transitNodes.length; i++) {
171                 String[] numbers = transitNodesString[i].split("-");
172                 transitNodes[i][0] = Integer.parseInt(numbers[0]);
173                 transitNodes[i][1] = Integer
174                         .parseInt(numbers.length > 1 ? numbers[1] : numbers[0]);
175             }
176         }
177 
178         List<LngLatAlt> trackString = new ArrayList<LngLatAlt>();
179 
180         int insertedNodes = 0;
181         int[] transferNodesOffset = new int[tracks.length];
182         LngLatAlt previousPoint = null;
183 
184         // Print tracks
185         for (int i = 0; i < tracks.length; i++) {
186             LngLatAlt currentPoint = tracks[i];
187             if (i > 0) {
188                 boolean inTransitNode = false;
189                 for (int j = 0; j < transitNodes.length; j++) {
190                     if (i >= transitNodes[j][0] && i <= transitNodes[j][1]) {
191                         inTransitNode = true;
192                     }
193                 }
194                 // then, check if we have to add virtual nodes
195                 double distance;
196                 if (MAX_DISTANCE != null
197                         && (distance = computeDistance(currentPoint,
198                                 previousPoint)) > MAX_DISTANCE && inTransitNode) {
199                     int extraNodes = (int) Math.ceil(distance / MAX_DISTANCE) - 1;
200                     for (int j = 1; j <= extraNodes; j++) {
201                         double lat = previousPoint.getLatitude()
202                                 + j
203                                 * (currentPoint.getLatitude() - previousPoint
204                                         .getLatitude()) / extraNodes;
205                         double lng = previousPoint.getLongitude()
206                                 + j
207                                 * (currentPoint.getLongitude() - previousPoint
208                                         .getLongitude()) / extraNodes;
209                         LngLatAlt extraPoint = new LngLatAlt(lng, lat);
210                         trackString.add(extraPoint);
211                     }
212                     insertedNodes += extraNodes;
213                 }
214             }
215             transferNodesOffset[i] = insertedNodes;
216             trackString.add(currentPoint);
217             previousPoint = currentPoint;
218         }
219 
220         for (int i = 0; i < transitNodes.length; i++) {
221             // Adjust with offset
222             for (int j = 0; j < 2; j++) {
223                 transitNodes[i][j] += transferNodesOffset[transitNodes[i][j]];
224             }
225         }
226         StringBuilder finalTextConf = new StringBuilder();
227         StringBuilder finalTextMySQL = new StringBuilder("GeomFromText('LineString(");
228         finalTextConf.append(trackTypeId + "." + trackId + "\t");
229         finalTextConf.append(penalty + "\t");
230         finalTextConf.append(trackString.size() + "\t");
231         for (int i = 0; i < trackString.size(); i++) {
232             if (i > 0) {
233                 finalTextConf.append("␣");
234                 finalTextMySQL.append(",");
235             }
236             finalTextConf.append(String.format("%.6f␣%.6f", trackString.get(i)
237                     .getLatitude(), trackString.get(i).getLongitude()));
238             finalTextMySQL.append(String.format("%.6f␣%.6f", trackString.get(i)
239                     .getLongitude(), trackString.get(i).getLatitude()));
240         }
241         finalTextConf.append("\t");
242         finalTextConf.append((isPathLoop ? "1" : "0") + "\t");
243         for (int i = 0; i < transitNodes.length; i++) {
244             if (i > 0) {
245                 finalTextConf.append(",");
246             }
247             if (transitNodes[i][0] == transitNodes[i][1]) {
248                 finalTextConf.append(transitNodes[i][0]);
249             } else {
250                 finalTextConf.append(String.format("%d-%d", transitNodes[i][0],
251                         transitNodes[i][1]));
252             }
253         }
254         finalTextMySQL.append(")')");
255         return new RouteResult(lastUpdate, finalTextConf.toString(), finalTextMySQL.toString());
256     }
257 
258     private RouteResult formatTrackFromAngkotWebId(String angkotId,
259             String trackTypeId, String trackId) throws IOException {
260         URL url = new URL(String.format(ANGKOTWEBID_URL, angkotId));
261         Main.globalLogger.info("Fetching␣" + trackTypeId + "." + trackId
262                 + "␣from␣" + url + "...");
```

```
263         JsonFactory factory = new JsonFactory();
264         JsonParser parser = factory.createParser(url);
265 
266         List<LngLatAlt> finalCoordinates = null;
267         Boolean isPathLoop = null;
268         int lastUpdate = -1;
269         while (!parser.isClosed()) {
270             JsonToken token = parser.nextToken();
271             if (token == null) {
272                 break;
273             }
274             if (JsonToken.FIELD_NAME.equals(token)
275                     && "updated".equals(parser.getCurrentName())) {
276                 parser.nextValue();
277                 lastUpdate = parser.getValueAsInt();
278             } else if (JsonToken.FIELD_NAME.equals(token)
279                     && "geojson".equals(parser.getCurrentName())) {
280                 parser.nextBooleanValue();
281                 Feature feature = new ObjectMapper().readValue(parser,
282                         Feature.class);
283                 List<List<LngLatAlt>> coordinates = ((MultiLineString) feature
284                         .getGeometry()).getCoordinates();
285                 if (coordinates.size() == 0) {
286                     Main.globalLogger.warning(String.format(
287                             "%s.%s/%s has zero routes, will be ignored.",
288                             trackTypeId, trackId, angkotId));
289                     return null;
290                 }
291                 if (coordinates.size() == 1) {
292                     finalCoordinates = coordinates.get(0);
293                     isPathLoop = computeDistance(finalCoordinates.get(0),
294                             finalCoordinates.get(finalCoordinates.size() - 1)) < MAX_LINK_DISTANCE;
295                 } else if (coordinates.size() == 2) {
296                     List<LngLatAlt> c1 = coordinates.get(0);
297                     List<LngLatAlt> c2 = coordinates.get(1);
298                     if (computeDistance(c1.get(c1.size() - 1), c2.get(0)) < MAX_LINK_DISTANCE
299                             && computeDistance(c1.get(0), c2.get(c2.size() - 1)) < MAX_LINK_DISTANCE) {
300                         finalCoordinates = c1;
301                         finalCoordinates.addAll(c2);
302                         isPathLoop = true;
303                     } else if (computeDistance(c1.get(0), c2.get(0)) < MAX_LINK_DISTANCE
304                             && computeDistance(c1.get(c1.size() - 1),
305                                     c2.get(c2.size() - 1)) < MAX_LINK_DISTANCE) {
306                         finalCoordinates = c1;
307                         for (int j = c2.size() - 1; j >= 0; j--) {
308                             finalCoordinates.add(c2.get(j));
309                         }
310                         isPathLoop = true;
311                     } else {
312                         throw new DataPullerException(
313                                 String.format(
314                                         "Does not support linking tracks that far away: %s.%s/%s ",
315                                         trackTypeId, trackId, angkotId));
316                     }
317                 } else {
318                     Main.globalLogger
319                             .warning(String
320                                     .format("Does not support tracks with %d routes: %s.%s/%s ",
321                                             coordinates.size(), trackTypeId,
322                                             trackId, angkotId));
323                     return null;
324                 }
325             }
326         }
327         if (finalCoordinates != null) {
328             RouteResult result = formatTrack(trackTypeId, trackId,
329                     finalCoordinates.toArray(new LngLatAlt[0]), isPathLoop,
330                     DEFAULT_PENALTY, null, lastUpdate);
331             return result;
332         } else {
333             Main.globalLogger.warning("Doesn't have GeoJSON info: " + angkotId);
334             return null;
335         }
336     }
337 
338     public static class RouteResult {
339         private final int lastUpdate;
340         private final String trackInConfFormat;
341         private final String trackInMySQLFormat;
342 
343         public RouteResult(int lastUpdate, String trackInConfFormat,
344                 String trackInMySQLFormat) {
345             super();
346             this.lastUpdate = lastUpdate;
347             this.trackInConfFormat = trackInConfFormat;
348             this.trackInMySQLFormat = trackInMySQLFormat;
349         }
350 
351         public int getLastUpdate() {
352             return lastUpdate;
353         }
354 
355         public String getTrackInConfFormat() {
356             return trackInConfFormat;
357         }
358 
359         public String getTrackInMySQLFormat() {
360             return trackInMySQLFormat;
361         }
```

```
362
363     }
364
365     private static class AngkotWebIdCacheInfo {
366         public final long lastUpdate;
367         public final boolean pathAvailable;
368
369         public AngkotWebIdCacheInfo(String trackTypeId, String trackId,
370                 long lastUpdate, int id, boolean pathAvailable) {
371             super();
372             this.lastUpdate = lastUpdate;
373             this.pathAvailable = pathAvailable;
374         }
375
376
377     }
378 }
```

Listing A.2: DataPullerException.java

```
1  package travel.kiri.backend.puller;
2
3  public class DataPullerException extends RuntimeException {
4
5      private static final long serialVersionUID = -5734202145456381699L;
6
7      public DataPullerException(String message) {
8          super(message);
9      }
10 }
```

LAMPIRAN B

KODE PROGRAM PENGUJIAN FUNGSIONAL

Listing B.1: DataPullerTest.java

```
1  package travel.kiri.backend.test;
2
3
4  import java.io.BufferedReader;
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.io.FileOutputStream;
8  import java.io.FileReader;
9  import java.io.IOException;
10 import java.io.PrintStream;
11 import java.sql.Connection;
12 import java.sql.DriverManager;
13 import java.sql.ResultSet;
14 import java.sql.SQLException;
15 import java.sql.Statement;
16 import java.util.Properties;
17
18 import org.junit.AfterClass;
19 import org.junit.Assert;
20 import org.junit.BeforeClass;
21 import org.junit.Test;
22
23 import travel.kiri.backend.puller.DataPuller;
24
25 public class DataPullerTest {
26
27     public static String SQL_PROPERTIES_FILE = "etc/mysql.properties";
28     public static String TRACKS_CONF_FILE = "etc/tracks.conf";
29
30     private static Connection connection;
31     private static Statement statement;
32     private static DataPuller puller;
33     private static Exception pullException;
34
35     @BeforeClass
36     public static void setUp() throws FileNotFoundException, IOException, SQLException {
37         // Setup database
38         Properties sqlProperties = new Properties();
39         sqlProperties.load(new FileReader(SQL_PROPERTIES_FILE));
40         connection = DriverManager.getConnection(String.format(
41                 "jdbc:mysql://%s/%s?user=%s&password=%s",
42                 sqlProperties.get("host"), sqlProperties.get("database"),
43                 sqlProperties.get("user"), sqlProperties.get("password")));
44
45         statement = connection.createStatement();
46         statement.execute("DROP␣TABLE␣IF␣EXISTS␣`tracks`;");
47         statement.execute("CREATE␣TABLE␣IF␣NOT␣EXISTS␣`tracks`␣(`trackId`␣varchar(32)␣NOT␣NULL,␣`
               trackTypeId`␣varchar(32)␣NOT␣NULL␣DEFAULT␣'angkot',␣`trackName`␣varchar(64)␣NOT␣NULL,␣`
               internalInfo`␣varchar(1024)␣NOT␣NULL,␣`geodata`␣linestring␣DEFAULT␣NULL,␣`pathloop`␣tinyint(1)
               ␣NOT␣NULL␣DEFAULT␣'0',␣`penalty`␣decimal(4,2)␣NOT␣NULL␣DEFAULT␣'1.00',␣`transferNodes`␣varchar
               (1024)␣DEFAULT␣NULL,␣`extraParameters`␣varchar(256)␣DEFAULT␣NULL,␣`officialTrackNo`␣varchar
               (32)␣DEFAULT␣NULL,␣`officialTrackName`␣varchar(256)␣DEFAULT␣NULL)␣ENGINE=InnoDB␣DEFAULT␣
               CHARSET=latin1;");
48         statement.execute("INSERT␣INTO␣`tracks`␣VALUES␣('testnotimestamp',␣'bdo_angkot',␣'Test␣for␣
               timestamp␣is␣not␣yet␣specified',␣'angkotwebid:642',␣GeomFromText(NULL),␣'0',␣'1.00',␣NULL,␣
               NULL,␣'1A',␣'Abdul␣Muis␣-␣Cicaheum␣(via␣Binong)');");
49         statement.execute("INSERT␣INTO␣`tracks`␣VALUES␣('testlowertimestamp',␣'bdo_angkot',␣'Test␣for␣
               timestamp␣is␣lower␣than␣today',␣'angkotwebid:641:808628400',␣GeomFromText('POINT(107.60380␣
               -6.91082)'),␣'0',␣'1.00',␣NULL,␣NULL,␣'1B',␣'Abdul␣Muis␣-␣Cicaheum␣(via␣Aceh)');");
50         statement.execute("INSERT␣INTO␣`tracks`␣VALUES␣('testhighertimestamp',␣'bdo_angkot',␣'Test␣for␣
               timestamp␣is␣higher␣than␣today',␣'angkotwebid:109:2386551600',␣GeomFromText('LINESTRING
               (107.6038␣-6.91082)'),␣'0',␣'1.00',␣NULL,␣NULL,␣'1B',␣'Abdul␣Muis␣-␣Cicaheum␣(via␣Aceh)');");
51         statement.execute("INSERT␣INTO␣`tracks`␣VALUES␣('testnokeyword',␣'bdo_angkot',␣'Test␣for␣no␣
               keyword␣is␣used',␣'Not␣an␣angkotwebid␣track',␣GeomFromText('LINESTRING(107.6038␣-6.91082)'),␣
               '0',␣'1.00',␣NULL,␣NULL,␣NULL,␣NULL);");
52
53         // Remove tracks.conf file (silent on error)
54         new File(TRACKS_CONF_FILE).delete();
55
56         // Perform pull, catching any errors thrown
57         pullException = null;
58         try {
59             puller = new DataPuller();
```

```
60              puller.pull(new File(SQL_PROPERTIES_FILE), new PrintStream(new FileOutputStream(
                    TRACKS_CONF_FILE)));
61          } catch (Exception e) {
62              pullException = e;
63          }
64      }
65
66      @AfterClass
67      public static void tearDown() throws SQLException {
68          statement.close();
69          connection.close();
70      }
71
72      /**
73       * Utility class untuk mencari baris pada berkas tracks.conf dengan trackId tertentu.
74       * @param trackId trackId yang dicari
75       * @return array of string kolom, dari baris terpisah tab atau null jika tidak ditemukan.
76       * @throws IOException
77       */
78      private static String[] getColumnsFromTracksConf(String trackId) throws IOException {
79          BufferedReader reader = new BufferedReader(new FileReader(TRACKS_CONF_FILE));
80          String line;
81          while ((line = reader.readLine()) != null) {
82              String[] columns = line.trim().split("\\t");
83              if (columns[0].equals("bdo_angkot." + trackId)) {
84                  reader.close();
85                  return columns;
86              }
87          }
88          reader.close();
89          return null;
90      }
91
92      /**
93       * Memastikan tidak ada eksepsi yang terjadi saat operasi pull dijalankan.
94       */
95      @Test
96      public void testEnsurePullNoException() {
97          if (pullException != null) {
98              pullException.printStackTrace();
99              Assert.fail("Exception was found during pull: " + pullException);
100         }
101     }
102
103     /**
104      * Memastikan pull berhasil memperbaharui SQL untuk data dengan internalInfo berisi 'angkotwebid:nnn'
             (tanpa timestamp)
105      * @throws SQLException
106      */
107     @Test
108     public void testSQLUpdatedFromNoTimestamp() throws SQLException {
109         ResultSet result = statement.executeQuery("SELECT internalInfo, AsText(geodata) FROM tracks WHERE
                trackId='testnotimestamp';");
110         Assert.assertTrue("Data uji tidak ditemukan!", result.next());
111         Assert.assertTrue("internalInfo tidak terupdate dengan baik!", result.getString(1).matches("
                angkotwebid:642:[0-9]+"));
112         Assert.assertTrue("geodata tidak terupdate dengan baik!", result.getString(2).matches("LINESTRING
                (.+)"));
113     }
114
115     /**
116      * Memastikan pull berhasil memperbaharui tracks.conf untuk data dengan internalInfo berisi '
             angkotwebid:nnn' (tanpa timestamp)
117      * @throws IOException
118      */
119     @Test
120     public void testTracksConfUpdatedFromNoTimestamp() throws IOException {
121         String[] columns = getColumnsFromTracksConf("testnotimestamp");
122         Assert.assertNotNull("Tidak ditemukan data uji di tracks.conf", columns);
123         int numOfCoordinates = Integer.parseInt(columns[2]);
124         Assert.assertTrue("Jumlah titik harus lebih dari 0!", numOfCoordinates > 0);
125         String[] coordinateNumbers = columns[3].split(" ");
126         Assert.assertEquals("Jumlah titik harus sesuai jumlah koordinat", numOfCoordinates * 2,
                coordinateNumbers.length);
127         Assert.assertEquals("transferNodes tidak sesuai di tracks.conf!", "0-" + (numOfCoordinates - 1),
                columns[5]);
128     }
129
130     /**
131      * Memastikan pull berhasil memperbaharui SQL untuk data dengan internalInfo berisi 'angkotwebid:nnn:
             mmm' (dengan timestamp tetapi lebih rendah)
132      * @throws SQLException
133      */
134     @Test
135     public void testSQLUpdatedFromLowerTimestamp() throws SQLException {
136         ResultSet result = statement.executeQuery("SELECT internalInfo, AsText(geodata) FROM tracks WHERE
                trackId='testlowertimestamp';");
137         Assert.assertTrue("Data uji tidak ditemukan!", result.next());
138         Assert.assertTrue("internalInfo tidak terupdate dengan baik: " + result.getString(1), result.
                getString(1).matches("angkotwebid:641:[0-9]+"));
139         String[] string1 = result.getString(1).split(":");
140         long timestamp = Long.parseLong(string1[2]);
141         Assert.assertTrue("timestamp tidak terupdate dengan baik!", timestamp > 808628400);
142         Assert.assertTrue("geodata tidak terupdate dengan baik!", result.getString(2).matches("LINESTRING
                (.+)"));
143     }
144
145     /**
```

```
146         * Memastikan pull berhasil memperbaharui tracks.conf untuk data dengan internalInfo berisi '
                angkotwebid:nnn:mmm' (dengan timestamp tetapi lebih rendah)
147         * @throws IOException
148         */
149        @Test
150        public void testTracksConfUpdatedFromLowerTimestamp() throws IOException {
151            String[] columns = getColumnsFromTracksConf("testlowertimestamp");
152            Assert.assertNotNull("Tidak ditemukan data uji di tracks.conf", columns);
153            int numOfCoordinates = Integer.parseInt(columns[2]);
154            Assert.assertTrue("Jumlah titik harus lebih dari 0!", numOfCoordinates > 0);
155            String[] coordinateNumbers = columns[3].split(" ");
156            Assert.assertEquals("Jumlah titik harus sesuai jumlah koordinat", numOfCoordinates * 2,
                    coordinateNumbers.length);
157            Assert.assertEquals("transferNodes tidak sesuai di tracks.conf!", "0-" + (numOfCoordinates - 1),
                    columns[5]);
158        }
159
160        /**
161         * Memastikan pull berhasil memperbaharui SQL untuk data dengan internalInfo berisi 'angkotwebid:nnn:
                mmm' (dengan timestamp tetapi lebih tinggi)
162         * @throws SQLException
163         */
164        @Test
165        public void testSQLNotUpdatedFromHigherTimestamp() throws SQLException {
166            ResultSet result = statement.executeQuery("SELECT internalInfo, AsText(geodata) FROM tracks WHERE
                    trackId='testhighertimestamp';");
167            Assert.assertTrue("Data uji tidak ditemukan!", result.next());
168            Assert.assertEquals("internalInfo tidak boleh terupdate!", "angkotwebid:109:2386551600", result.
                    getString(1));
169            Assert.assertEquals("geodata tidak boleh terupdate!", "LINESTRING(107.6038 -6.91082)", result.
                    getString(2));
170        }
171
172        /**
173         * Memastikan pull berhasil memperbaharui tracks.conf untuk data dengan internalInfo berisi '
                angkotwebid:nnn:mmm' (dengan timestamp tetapi lebih tinggi)
174         * @throws IOException
175         */
176        @Test
177        public void testTracksConfNotUpdatedFromHigherTimestamp() throws IOException {
178            String[] columns = getColumnsFromTracksConf("testhighertimestamp");
179            Assert.assertNotNull("Tidak ditemukan data uji di tracks.conf", columns);
180            int numOfCoordinates = Integer.parseInt(columns[2]);
181            Assert.assertEquals("Jumlah titik harus tetap 1!", 1, numOfCoordinates);
182            Assert.assertEquals("transferNodes tidak boleh berubah!", "0", columns[5]);
183        }
184
185        /**
186         * Memastikan pull tidak memperbaharui SQL untuk data dengan internalInfo berisi 'Not an angkotwebid
                track' (bukan track yang terintegrasi)
187         * @throws SQLException
188         */
189        @Test
190        public void testSQLNotUpdatedFromNotAngkotWebid() throws SQLException {
191            ResultSet result = statement.executeQuery("SELECT internalInfo, AsText(geodata) FROM tracks WHERE
                    trackId='testnokeyword';");
192            Assert.assertTrue("Data uji tidak ditemukan!", result.next());
193            Assert.assertEquals("internalInfo tidak boleh terupdate!", "Not an angkotwebid track", result.
                    getString(1));
194            Assert.assertEquals("geodata tidak boleh terupdate!", "LINESTRING(107.6038 -6.91082)", result.
                    getString(2));
195        }
196
197        /**
198         * Memastikan pull tidak memperbaharui tracks.conf untuk data dengan internalInfo berisi 'Not an
                angkotwebid track' (bukan track yang terintegrasi)
199         * @throws IOException
200         */
201        @Test
202        public void testTracksConfUpdatedFromHigherTimestamp() throws IOException {
203            String[] columns = getColumnsFromTracksConf("testnokeyword");
204            Assert.assertNotNull("Tidak ditemukan data uji di tracks.conf", columns);
205            int numOfCoordinates = Integer.parseInt(columns[2]);
206            Assert.assertEquals("Jumlah titik harus tetap 1!", 1, numOfCoordinates);
207            Assert.assertEquals("transferNodes tidak boleh berubah!", "0", columns[5]);
208        }
209
210    }
```

Listing B.2: Main.java

```
1  package travel.kiri.backend;
2
3  import java.util.logging.Logger;
4
5  public class Main {
6      public static final Logger globalLogger = Logger.getGlobal();
7  }
```