PEMANFAATAN BASIS DATA GRAF PADA APLIKASI RUTE TRANSPORTASI UMUM (STUDI KASUS: BUS RAPID TRANSIT TRANS JOGJA)

Disusun Oleh:

N a m a : Muhammad Noer Ramadhan
NIM : 15523240

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2022

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PEMANFAATAN BASIS DATA GRAF PADA APLIKASI RUTE TRANSPORTASI UMUM (STUDI KASUS: BUS RAPID TRANSIT TRANS JOGJA)

TUGAS AKHIR

Disusun Oleh:

N a m a : Muhammad Noer Ramadhan

NIM : 15523240

Yogyakarta, 18 Januari 2023

Pembimbing,

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PEMANFAATAN BASIS DATA GRAF PADA APLIKASI RUTE TRANSPORTASI UMUM (STUDI KASUS: BUS RAPID TRANSIT TRANS JOGJA)

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 30 Desember 2022

Tim Penguji

Dr. Raden Teduh Dirgahayu, S.T., M.Sc. _____

Anggota 1

Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. _____

Anggota 2

Chanifah Indah Ratnasari, S.Kom., M.Kom., _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Muhammad Noer Ramadhan

NIM : 15523240

Tugas akhir dengan judul:

PEMANFAATAN BASIS DATA GRAF PADA APLIKASI RUTE TRANSPORTASI UMUM (STUDI KASUS: BUS RAPID TRANSIT TRANS JOGJA)

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apa pun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 5 Januari 2023

(Muhammad Noer Ramadhan)

HALAMAN PERSEMBAHAN

Kepada kehidupan yang belum berhasil penulis rampungkan,

Alhamdulillah, melihat daun yang masih mengikatkan diri pada batang dengan eratnya, penulis dapat bercanda dengan tuhan untuk kesekian kalinya dan menyelesaikan apa yang seperlunya penulis selesaikan.

Dengan demikian, bersama hilangnya alasan manusia lain untuk menggunakan penelitian tugas akhir sebagai alasan. Penulis menyambut dengan tangan terbuka kepada segala yang akan datang selanjutnya.

Terima Kasih.

HALAMAN MOTO

Untuk hidup menghujam pasak julangkan tiang.
Untuk terputus suram kala tekadmu menjelang,
Sayang.
(*Anonymous Alliance*, Semerkah Merah Rumah)

Ribuan harap menggantung di ujung malam
Langitkan niatan untuk senyum tunas harapan
Melabrak, bara pada sekam yang menahan
Jejak kaki menapak untuk hijau kesempatan
(*Anonymous Alliance*, Menyulam Azzam)

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh

Alhamdulillah, penulis panjatkan puja dan puji syukur ke hadirat Allah SWT yang telah memberikan rahmat, hidayah, dan karunia-Nya, sehingga laporan penelitian tugas akhir dapat penulis selesaikan. Tak lupa selawat dan salam penulis sampaikan kepada junjungan kita Nabi Muhammad SAW, yang telah membawa kita dari zaman jahiliah menuju zaman terang benderang.

Laporan penelitian tugas akhir ini dibuat sebagai salah satu syarat yang harus dipenuhi untuk memperoleh gelar sarjana di Program Studi Informatika Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia. Adapun penelitian tugas akhir mengenai "**PEMANFAATAN BASIS DATA GRAF PADA APLIKASI RUTE TRANSPORTASI UMUM (STUDI KASUS: BUS RAPID TRANSIT TRANS JOGJA)**".

Pelaksanaan penelitian tugas akhir ini merupakan salah satu mata kuliah wajib dari Program Studi Informatika Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia dan juga merupakan sarana bagi penulis untuk menambah wawasan serta pengalaman dalam menerapkan keilmuan, sesuai dengan yang diambil di bangku perkuliahan.

Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih kepada:

1. Allah SWT.
2. Bapak Fathul Wahid, S.T., M.Sc., Ph.D. selaku Rektor Universitas Islam Indonesia.
3. Bapak Prof. Dr. Ir. Hari Purnomo, M.T. selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Jurusan Informatika Fakultas Teknologi Industri Universitas Islam Indonesia dan Dosen Pembimbing Tugas Akhir.
5. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku Ketua Program Studi Informatika Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia dan Dosen Pembimbing Akademik.
6. Bapak, Ibu, Adik.
7. Lembaga Pers Mahasiswa Profesi Fakultas Teknologi Industri Universitas Islam Indonesia.
8. Asha Novianty S., Rizky Reldian R, dan Nasrul Haqqi.
9. Widi Setyo H, Kurniaji Gunawan, Agung Prabowo

10. Informatika 2015 Metamorf, Kelas E.
11. Semua pihak yang tidak dapat penulis sebutkan satu persatu, yang turut membantu penulis dalam pelaksanaan tugas akhir.

Penulis menyadari bahwa laporan ini belum sempurna, karena keterbatasan kemampuan dan pengalaman di lapangan. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun demi kesempurnaan laporan penelitian tugas akhir ini. Akhir kata, penulis berharap agar laporan ini dapat bermanfaat bagi semua pihak.

Wassalamu'alaikum Warahmatullahi Wabarakatuh

Yogyakarta, 5 Desember 2022

(Muhammad Noer Ramadhan)

SARI

Bus Rapid Transit (BRT) Trans Jogja merupakan salah satu penerapan sistem transportasi publik berbasis bus yang beroperasi di Daerah Istimewa Yogyakarta sejak tahun 2008. Sistem BRT dirancang dengan kapasitas dan keandalan yang lebih baik dibandingkan sistem bus konvensional. Sistem BRT menawarkan pelayanan dengan mobilitas cepat, nyaman, dan berbiaya rendah. Pelayanan tersebut dimungkinkan dengan adanya jalur khusus, pembayaran tarif di luar bus, prioritas persimpangan, dan lantai *boarding*. Dalam pelayanannya, Trans Jogja menyediakan 17 koridor, 267 *shelter*, dan 129 bus yang menjangkau beberapa wilayah di Yogyakarta, antara lain Kota Yogyakarta, Kabupaten Sleman, serta Kabupaten Bantul. Sebenarnya, Dishub DIY sudah menyediakan informasi terkait rute dan *shelter* ini dalam bentuk peta. Namun, penyajian informasi yang masih berbentuk peta tersebut belum interaktif karena masih diperlukan pencarian manual selumrahnya pembacaan peta untuk melihat rute yang ingin diambil.

Melihat permasalahan yang ada, sebuah aplikasi berbasis web yang memanfaatkan basis data graf dapat menjadi salah satu alternatif solusi. Salah satu basis data berjenis NoSQL yang menggunakan struktur data graf untuk *query* semantik dengan *Nodes*, *Edges*, dan *Properties* guna merepresentasikan dan menyimpan data. Hubungan dapat divisualisasikan secara intuitif menggunakan basis data graf, hal tersebut berguna untuk data yang saling terhubung. Aplikasi dikembangkan menggunakan metode antara lain kajian pustaka, analisis kebutuhan, pemodelan domain basis data, implementasi dan pengujian (*backend*), perancangan dan implementasi aplikasi (*frontend*), serta pengujian sistem secara menyeluruh.

Aplikasi yang dikembangkan bertujuan menjadi salah satu bentuk pemanfaatan basis data graf sebagai usaha menawarkan alternatif solusi untuk membantu menyelesaikan masalah dalam kehidupan sehari-hari. Dalam konteks penelitian ini, aplikasi yang dikembangkan memiliki tujuan untuk memberikan rekomendasi rute BRT Trans Jogja. Berdasarkan hasil pengujian yang telah dilakukan, berhasil dikembangkan aplikasi berbasis web yang mengimplementasikan basis data graf untuk menentukan rute transportasi BRT Trans Jogja. Melihat hasil pengujian tersebut, dicapai kesimpulan bahwa aplikasi tersebut dapat memenuhi kebutuhan dan dapat menjadi alternatif solusi dari masalah yang ada.

Kata kunci: Basis Data Graf, Neo4J, Bus Rapid Transit, BRT, Trans Jogja, Rute Transportasi Umum.

GLOSARIUM

API	*Application Programming Interface*, koleksi perintah kode pemrograman yang menghubungkan komponen-komponen perangkat lunak.
Route	Bagian dari API yang berisi jalur untuk berkomunikasi.
Service	Bagian dari API yang berisi logika untuk memroses data sesuai dengan permintaan.
Model	Bagian dari API yang berisi perintah pengolahan data.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam Undang-Undang Nomor 22 Tahun 2009 Tentang Lalu Lintas dan Angkutan Jalan, pemerintah mendefinisikan transportasi menggunakan istilah angkutan. Pada Bab 1 Pasal 1 Ayat 3, angkutan didefinisikan sebagai perpindahan orang dan/atau barang dari satu tempat ke tempat lain dengan menggunakan kendaraan di ruang lalu lintas jalan. Dalam Kamus Besar Bahasa Indonesia, salah satu pengertian untuk Transportasi adalah pengangkutan barang oleh berbagai jenis kendaraan sesuai dengan kemajuan teknologi (Badan Pengembangan dan Pembinaan Bahasa Kemendikbud RI, 2016b). Sedangkan, Angkutan memiliki beberapa pengertian, yaitu: 1) Barang-barang (orang dan sebagainya yang diangkut; 2) Cara mengangkut; dan 3) Moda transportasi (Badan Pengembangan dan Pembinaan Bahasa Kemendikbud RI, 2016a). Transportasi umum sendiri adalah layanan angkutan penumpang oleh sistem perjalanan kelompok yang tersedia untuk digunakan oleh masyarakat umum, biasanya dikelola sesuai jadwal, dioperasikan pada rute yang ditetapkan, dan dikenakan biaya untuk setiap perjalanan.

Transportasi terbagi dalam beberapa jenis, yaitu: transportasi darat, transportasi udara, dan transportasi air. *Bus Rapid Transit* (BRT) termasuk dalam transportasi jenis darat. BRT sendiri merupakan sistem transit massal berbasis bus yang memberikan mobilitas cepat, nyaman, dan berbiaya rendah dalam pelayanannya sebagai angkutan dalam perkotaan. BRT menggunakan jalur khusus dan pelayanan prima terhadap pengguna yang pada dasarnya adalah mengadaptasi karakteristik kinerja dan keandalan pelayanan dari sistem transit modern berbasis rel, akan tetapi dalam biaya yang lebih rendah. Salah satu wilayah di mana BRT telah beroperasi adalah Provinsi Daerah Istimewa Yogyakarta dengan Trans Jogja.

Trans Jogja memiliki 129 armada yang tersebar di berbagai rute dengan jalur masing-masing, baik armada yang aktif maupun armada cadangan (Dinas Perhubungan Daerah Istimewa Yogyakarta, 2022). Menurut data dari Dinas Perhubungan (Dishub) Daerah Istimewa Yogyakarta (DIY), ada 17 rute dan 267 halte yang tersebar di beberapa titik wilayah Kota Yogyakarta, Kabupaten Sleman, dan Kabupaten Bantul (Dinas Perhubungan Daerah Istimewa Yogyakarta, 2021, 2022). Melihat banyaknya rute dan halte tersebut, mungkin sekali bagi calon

pengguna, terlebih pendatang, kesulitan untuk menentukan rute dan halte mana yang perlu diambil untuk sampai ke suatu tujuan ketika akan menggunakan Trans Jogja.

Sebenarnya, Dishub DIY sudah menyediakan informasi terkait rute dan halte ini dalam bentuk peta yang dapat diakses di situs baik Dishub DIY maupun WikiMedia dan WikiPedia yang disediakan oleh Dishub DIY dan Forum Diskusi Transportasi Yogyakarta. Namun, penyajian informasi yang masih berbentuk peta tersebut belum interaktif karena masih diperlukan pencarian manual selumrahnya pembacaan peta untuk melihat rute yang ingin diambil. Penyajian informasi terkait halte juga dapat diakses di situs Sistem Informasi Lalu Lintas Angkutan Jalan (SI LLAJ) Dishub DIY. Penyajian informasi di situs tersebut lebih interaktif karena sudah menggunakan Google Maps, dibandingkan dengan informasi terkait rute yang hanya berupa peta saja. Selain letak halte, Informasi yang tersedia pun sudah cukup informatif dengan adanya keterangan berupa kategori halte, jenis halte, sumber dana, sumber listrik, dan kondisi halte. Akan tetapi, penyajian informasi terkait halte tersebut belum terlalu interaktif bagi calon pengguna yang ingin menentukan rute karena hanya menampilkan data halte saja. Calon pengguna belum mendapatkan kemudahan menyaring informasi berupa kebebasan penentuan tujuan dan rute yang dapat diambil untuk sampai ke suatu tujuan. Selain itu, informasi yang disediakan hanya salah satu layanan BRT yang ada di Yogyakarta. Belum ada integrasi informasi rute bagi layanan Trans Jogja Istimewa dan Teman Bus. Memang sudah tersedia aplikasi-aplikasi yang dapat memenuhi kebutuhan calon pengguna untuk menentukan rute BRT Trans Jogja. Namun, untuk mengakses informasi BRT Trans Jogja, pengguna perlu memasang aplikasi tersebut pada perangkat yang digunakan.

Salah satu alternatif solusi yang dapat digunakan untuk mengatasi masalah tersebut adalah aplikasi yang memanfaatkan basis data graf. Salah satu basis data berjenis NoSQL yang menggunakan struktur data graf untuk *query* semantik dengan *Nodes, Edges,* dan *Properties* guna merepresentasikan dan menyimpan data. Basis data ini menjadi pilihan yang cocok melihat data dalam studi kasus ini seperti misalnya antara rute dan halte. Hal tersebut dikarenakan graf menghubungkan item data dengan kumpulan *nodes* dan *edge*, setiap *edge* merepresentasikan hubungan antar *nodes*. Hubungan tersebut memungkinkan data untuk bertautan langsung dan, dalam banyak kasus, diambil dalam satu operasi. Basis data graf dapat melakukan *query* hubungan dengan cepat karena hubungan antar data secara terus menerus disimpan di dalam basis data. Hubungan dapat divisualisasikan secara intuitif menggunakan basis data graf, hal tersebut berguna untuk data yang saling terhubung. Memanfaatkan salah satu basis data graf yaitu Neo4J, aplikasi dapat menampilkan informasi terkait rute yang dapat

digunakan secara terperinci seperti jalan mana saja yang dilalui dan halte mana yang dapat digunakan sesuai dengan tujuan yang telah ditentukan oleh calon pengguna Trans Jogja. Perkembangan teknologi dewasa kini juga dapat memberikan kemudahan untuk mendapatkan informasi yang informatif dan interaktif dengan membuat aplikasi tersebut berbasis web sehingga calon pengguna dapat mengakses informasi tersebut kapan saja dan di mana saja tanpa perlu memasang aplikasi di perangkan yang digunakan.

Berdasarkan uraian sebelumnya, adanya aplikasi berbasis web yang memanfaatkan basis data graf diharapkan dapat membuat informasi terkait rute BRT Trans Jogja menjadi lebih informatif dan interaktif sesuai dengan kebutuhan calon pengguna.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang ada, berikut adalah rumusan masalah yang diperoleh, antara lain sebagai berikut:

a. Bagaimana memanfaatkan basis data graf untuk menentukan rute transportasi *Bus Rapid Transit* Trans Jogja?
b. Bagaimana memodelkan rute transportasi *Bus Rapid Transit* Trans Jogja dalam bentuk graf?
c. Bagaimana mengimplementasikan model graf rute transportasi *Bus Rapid Transit* Trans Jogja dalam bentuk aplikasi berbasis web yang memanfaatkan basis data graf?
d. Bagaimana membangun aplikasi berbasis web dengan memanfaatkan basis data graf untuk menentukan rute transportasi *Bus Rapid Transit* Trans Jogja?

1.3 Batasan Masalah

Berikut adalah batasan masalah yang digunakan dalam penelitian ini:

a. Aplikasi dibangun menggunakan basis data graf Neo4J.
b. Aplikasi dikembangkan untuk *Bus Rapid Transit* Trans Jogja.
c. Data terkait *Bus Rapid Transit* Trans Jogja yang digunakan berdasarkan data tahun 2022.
d. Tidak adanya data terpusat terkait Bus Rapid Transit Trans Jogja sehingga pembaharuan data perlu dilakukan secara manual.
e. Aplikasi memerlukan koneksi internet untuk dapat diakses dan bekerja dengan baik.
f. Aplikasi tidak memiliki fitur pelacakan bus secara langsung.
g. Aplikasi tidak memiliki fitur untuk memeriksa ketersediaan bus untuk setiap koridor.

1.4 Tujuan Penelitian

Berikut adalah tujuan dari penelitian ini:

a. Memanfaatkan basis data graf untuk menentukan rute transportasi *Bus Rapid Transit* Trans Jogja.
b. Membangun representasi rute transportasi *Bus Rapid Transit* Trans Jogja dalam model graf.
c. Mengimplementasikan model graf rute transportasi *Bus Rapid Transit* Trans Jogja dalam bentuk aplikasi berbasis web.
d. Membangun aplikasi berbasis web dengan memanfaatkan basis data graf untuk menentukan rute transportasi *Bus Rapid Transit* Trans Jogja.

1.5 Manfaat Penelitian

Berikut adalah manfaat dari penelitian ini:

a. Membantu pengguna *Bus Rapid Transit* Trans Jogja mencari rute secara mudah.
b. Memberikan informasi terkait rute transportasi *Bus Rapid Transit* Trans Jogja yang interaktif.
c. Memberikan contoh pemanfaatan basis data graf guna membantu menyelesaikan masalah kehidupan sehari-hari.

1.6 Metode Penelitian

Berikut adalah metode yang digunakan dalam penelitian ini:

a. Kajian Pustaka

   Tahap ini digunakan untuk mengumpulkan data dan informasi yang dibutuhkan dengan sumber seperti buku, jurnal, serta informasi lain yang dipublikasikan dan dapat diakses melalui internet.
b. Analisis Kebutuhan

   Tahap ini digunakan untuk mencari tahu apa saja kebutuhan calon pengguna *Bus Rapid Transit* Trans Jogja dibantu dengan data dan informasi yang telah didapatkan sebelumnya.
c. Pemodelan Domain Basis Data

   Tahap ini digunakan untuk merancang bagaimana struktur basis data yang akan digunakan sehingga dapat merepresentasikan data-data terkait *Bus Rapid Transit* Trans Jogja.
d. Implementasi dan Pengujian (*Backend*)

Tahap ini digunakan untuk pengembangan server berdasarkan rancangan yang telah dibuat. Nantinya server akan menyediakan data guna keperluan antarmuka. Setelah pengembangan selesai, dilakukan pengujian server untuk memastikan server dapat berjalan sesuai dengan rancangan. Langkah-langkah sebelumnya dilakukan sesuai dengan kebutuhan yang telah diteliti sebelumnya.

e. Perancangan dan Implementasi Aplikasi (*Frontend*)

Tahap ini digunakan untuk merancang antarmuka aplikasi dan pengembangan antarmuka aplikasi berdasarkan rancangan tersebut. Antarmuka nantinya akan mengonsumsi data yang telah disediakan oleh server. Langkah-langkah sebelumnya dilakukan sesuai dengan kebutuhan yang telah diteliti sebelumnya

f. Pengujian Sistem secara Menyeluruh

Tahap ini digunakan untuk memastikan server dan antarmuka yang telah dikembangkan bekerja sesuai dengan baik dan sesuai dengan kebutuhan yang telah diteliti sebelumnya.

1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan yang digunakan dalam laporan penelitian ini:

a. BAB I PENDAHULUAN

Bab ini memuat tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian dan sistematika penulisan dari pemanfaatan basis data graf pada aplikasi rute transportasi *Bus Rapid Transit* Trans Jogja.

b. BAB II LANDASAN TEORI

Bab ini memuat tentang dasar-dasar teori pemanfaatan basis data graf pada aplikasi rute transportasi *Bus Rapid Transit* Trans Jogja dan penulisan laporan penelitian tugas akhir.

c. BAB III METODOLOGI

Bab ini memuat tahap-tahap dalam pengembangan aplikasi rute transportasi *Bus Rapid Transit* Trans Jogja dengan memanfaatkan basis data graf mulai dari kajian pustaka, analisis kebutuhan, pemodelan domain basis data, perancangan aplikasi, dan rencana pengujian.

d. BAB IV HASIL DAN PEMBAHASAN

Bab ini memuat tentang implementasi pemanfaatan basis data graf pada aplikasi rute transportasi *Bus Rapid Transit* Trans Jogja dan hasil pengembangan aplikasi baik itu server maupun antarmuka, serta hasil pengujian baik server maupun sistem secara menyeluruh.

e. BAB V KESIMPULAN DAN SARAN

Bab ini memuat kesimpulan dari pemanfaatan basis data graf pada aplikasi rute transportasi *Bus Rapid Transit* Trans Jogja berdasarkan seluruh proses pengembangan aplikasi serta masukan berupa saran yang dapat menjadi pertimbangan bagi pengembangan ke depannya.

BAB II
LANDASAN TEORI

2.1 *Bus Rapid Transit*

Bus Rapid Transit (BRT) adalah sistem transportasi publik berbasis bus yang dirancang dengan kapasitas dan keandalan yang lebih baik dibandingkan sistem bus konvensional (Institute for Transportation & Development Policy, 2022b). Sebagai angkutan dalam perkotaan, BRT menawarkan pelayanan dengan mobilitas cepat, nyaman, dan berbiaya rendah. Pelayanan tersebut dimungkinkan melihat fitur-fitur yang dimiliki oleh BRT. Berdasarkan Standar BRT dari *Institute for Transportation & Development Policy* fitur dasar yang ada pada sebagian besar BRT adalah sebagai berikut (Institute for Transportation & Development Policy, 2022a):

a. Jalur khusus
b. Pembayaran tarif di luar bus
c. Prioritas persimpangan
d. Lantas *boarding*

BRT di Indonesia pertama kali diterapkan pada tahun 2004 di Jakarta dengan Trans Jakarta (PT. Transportasi Jakarta, 2016). Pada tahun 2022, setidaknya lebih dari 20 kota di Indonesia sudah menerapkan sistem BRT. Salah satu kota tersebut adalah Yogyakarta yang memiliki dua pelayanan BRT, yaitu Trans Jogja dan Teman Bus. Kota-kota yang sudah menerapkan BRT tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Daftar Layanan BRT di Indonesia

No.	Nama	Kota
1	Trans Jakarta	Jakarta
2	Trans Pakuan	Bogor
3	Trans Batik Solo	Surakarta
4	Trans Semarang	Semarang
5	Trans Jateng	Jawa Tengah
6	Trans Jogja Istimewa	Yogyakarta
7	Trans Bandung	Bandung
8	Trans Pasundan	Bandung
9	Trans Banjarkula	Banjarmasin
10	Trans Dewata	Denpasar
11	Trans Jayapura	Jayapura
12	Trans Pontianak	Pontianak
13	Trans Pekanbaku	Pekanbaru
14	Trans Deli	Medan
15	Trans Semanggi	Surabaya
16	Trans Suroboyo Bus	Surabaya
17	Trans Jatim	Surabaya
18	Trans Musi	Palembang
19	Trans Padang	Padang
20	Trans Mamminasata	Makassar
21	Trans Mebidang	Medan
22	Trans Lampung	Bandar Lampung
23	Trans Sarbagita	Denpasar
24	Trans Patriot	Bekasi
25	Trans GDC	Depok
26	Trans Margonda	Depok
27	Trans Kutaraja	Banda Aceh
28	Trans Siginjai	Jambi
29	Trans Palangkaraya	Palangkaraya

2.2 Trans Jogja

Trans Jogja, dengan merek dagang Trans Jogja Istimewa, merupakan penerapan sistem BRT yang beroperasi di Daerah Istimewa Yogyakarta sejak tahun 2008. Dalam pelaksanaannya, Trans Jogja dikelola dan dioperasikan oleh PT Jogja Tugu Trans (PT JTT) dan PT Anindya Mitra Internasional (PT AMI).

PT JTT sendiri merupakan konsorsium antara pengelola transportasi umum kota dan pedesaan di Yogyakarta yang terdiri dari Koperasi Pemuda Sleman, Kopata, Aspada, Kobutri, dan Puskopkar, dengan Perum DAMRI. Sedangkan, PT AMI adalah Badan Usaha Milik Daerah DIY yang bergerak di bidang barang dan jasa.

Trans Jogja memiliki 17 koridor dan 267 *shelter* yang tersebar di wilayah Kota Yogyakarta, Kabupaten Sleman, dan Kabupaten Bantul (Dinas Perhubungan Daerah Istimewa Yogyakarta, 2021, 2022). Sebanyak 129 bus dioperasikan dengan 12 bus digunakan sebagai

cadangan (Dinas Perhubungan Daerah Istimewa Yogyakarta, 2022). Informasi terkait koridor Trans Jogja dapat dilihat pada Tabel 2.2.

Tabel 2.2 Koridor yang Tersedia dalam Layanan BRT Trans Jogja

Koridor	Rute	Jenis Rute
1A	Terminal Prambanan – Malioboro	*Loop*
1B	Bandara Adisutjipto – Pasar Pathuk	*Loop*
2A	Terminal Condongcatur – XT *Square*	*Loop*
2B	Terminal Condongcatur – XT *Square*	*Loop*
3A	Bandara Adisutjipto – Terminal Ngabean	*Loop*
3B	Terminal Giwangan – Terminal Condongcatur	*Loop*
4A	Terminal Giwangan – UGM	*Loop*
4B	Terminal Giwangan – UGM	*Loop*
5A	Terminal Jombor – Babarsari	*Loop*
5B	Terminal Jombor – Bandara Adisutjipto	*Loop*
6A	Gamping – Ngabean	*Loop*
6B	Gamping – Ngabean	*Loop*
7	Terminal Giwangan – Babarsari	Dua Arah
8	Terminal Jombor – Jogokaryan	Dua Arah
9	Terminal Jombor – Terminal Giwangan	Dua Arah
10	Gamping – SGM (Kusumanegara)	Dua Arah
11	Terminal Giwangan – Terminal Condongcatur	Dua Arah

2.3 Teman Bus

Teman Bus adalah salah satu layanan yang menerapkan sistem BRT di DIY. Teman Bus sendiri merupakan implementasi program *Buy the Service* dari Kementerian Perhubungan Republik Indonesia (Kementerian Perhubungan Republik Indonesia, 2020a).

Sejak tahun 2020 (Kementerian Perhubungan Republik Indonesia, 2020a), DIY menjadi kota keempat yang mengoperasikan layanan ini setelah Bali. Selain DIY, kota-kota lain yang telah mengoperasikan Teman Bus antara lain adalah sebagai berikut: Palembang, Solo, Bali, Medan, Makassar, Banyumas, Bandung, Bogor, Surabaya, dan Banjarmasin.

Dalam pelayanannya di DIY (Kementerian Perhubungan Republik Indonesia, 2020b), Teman Bus memiliki 3 koridor yang tersebar di wilayah Kota Yogyakarta dan Kabupaten Sleman. Sebanyak 44 bus dioperasikan untuk melayani koridor yang ada. Teman Bus di Yogyakarta dioperasikan oleh PT JTT. Informasi terkait koridor Teman Bus di DIY dapat dilihat pada Tabel 2.3.

Tabel 2.3 Koridor yang Tersedia dalam Layanan Teman Bus Yogyakarta

Koridor	Rute	Jenis Rute
K1J	Terminal Condongcatur – Terminal Pakem	Dua Arah
K2J	Ngabean – Pusat Kuliner Belut Godean	Loop
K3J	Bandara Adisutjipto – Terminal Pakem	Dua Arah

2.4 Graf

Secara umum, graf adalah kumpulan titik yang saling berhubungan. Titik dapat disebut dengan *Node*, atau *Vertex* untuk tunggal, *Vertices* untuk jamak. Sedangkan, hubungan antar titik disebut *Edge* (Holton & Clark, 1995). Secara matematis, graf didefinisikan dengan G = (V, E) di mana G adalah sebuah graf yang terdiri dari dua himpunan terbatas. Himpunan tersebut adalah V yang berisi himpunan *node* dan E yang berisi himpunan *edge* (Levin, 2018).

Graf sendiri dibagi menjadi beberapa jenis berdasarkan karakteristik-karakteristik tertentu yang terdapat pada *edge*, seperti misalnya keberadaan bobot dan arah (Rosen, 2012). Selain bobot dan arah, keberadaan *edge* jamak dan *edge loop* juga menjadi dasar pembeda jenis graf. *Edge* jamak sendiri adalah kondisi di mana suatu *node* memiliki *edge* lebih dari satu. Sedangkan *edge loop* adalah kondisi di mana suatu *node* memiliki *edge* yang menghubungkan dengan *node* itu sendiri. Untuk lebih jelasnya, terkait jenis graf dapat dilihat pada Tabel 2.4.

Tabel 2.4 Jenis Graf Berdasarkan Karakteristik *Edge*

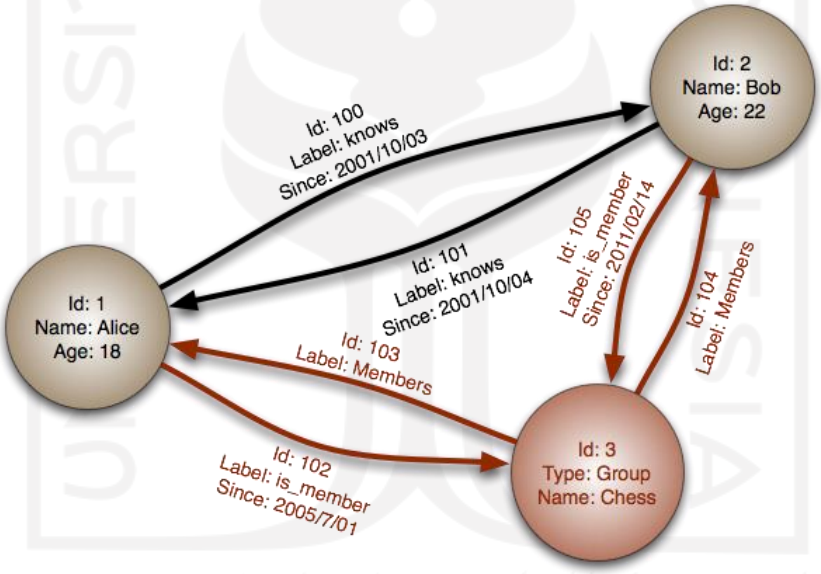
Nama	Jenis *Edge*	*Edge* Jamak	*Edge Loop*
Simple Graph	Tak Berarah	Tidak Ada	Tidak Ada
Multigraph	Tak Berarah	Ada	Tidak Ada
Pseudograph	Tak Berarah	Ada	Ada
Simple Directed Graph	Berarah	Tidak Ada	Tidak Ada
Directed Multighraph	Berarah	Ada	Ada
Mixed Graph	Berarah dan Tak Berarah	Ada	Ada

Implementasi dari graf dalam kehidupan sehari-hari dapat ditemukan pada BRT Trans Jogja. BRT Trans Jogja memiliki *shelter* dan koridor. Dalam konteks graf, setiap *shelter* dapat mewakili *node*. Sedangkan hubungan antar *shelter*, yang mana merupakan bagian dari koridor, dapat mewakili *edge*. Bus yang bergerak dari suatu *shelter* A terhubung oleh suatu koridor menuju *shelter* lain B, hanya dapat bergerak satu arah, yaitu dari A ke B, tidak bisa sebaliknya, dari B ke A. Melihat hal tersebut, maka BRT Trans Jogja dapat dikatakan berjenis graf berarah. Kemudian, dari suatu *shelter* A menuju *shelter* B dapat memiliki beberapa koridor. Melihat hal tersebut, maka BRT Trans Jogja dapat dikatakan berjenis *multigraph* dikarenakan keberadaan *edge* jamak. Selanjutnya, ketika jarak dan waktu tempuh dipertimbangkan, maka terdapat suatu

nilai pada hubungan antara *shelter* yang perlu diperhitungkan. Melihat hal tersebut, maka BRT Trans Jogja dapat dikatakan berjenis graf berbobot. Dari uraian sebelumnya, melihat keberadaan arah, bobot, dan *edge* jamak, maka BRT Trans Jogja dapat dikatakan berjenis *weighted directed multigraph*.

2.5 Basis Data Graf

Basis data graf adalah sebuah sistem manajemen basis data daring dengan operasi *Create, Read, Update, Delete* (CRUD) yang bekerja pada model data graf (Robinson et al., 2015). Secara sederhana, basis data graf adalah sebuah basis data yang menggunakan struktur graf yang memiliki komponen *node*, *edge*, dan *property* untuk merepresentasikan dan menyimpan data. *Node* merepresentasikan entitas dan *edge* merepresentasikan hubungan antar entitas.



Gambar 2.1 Ilustrasi Basis Data Graf

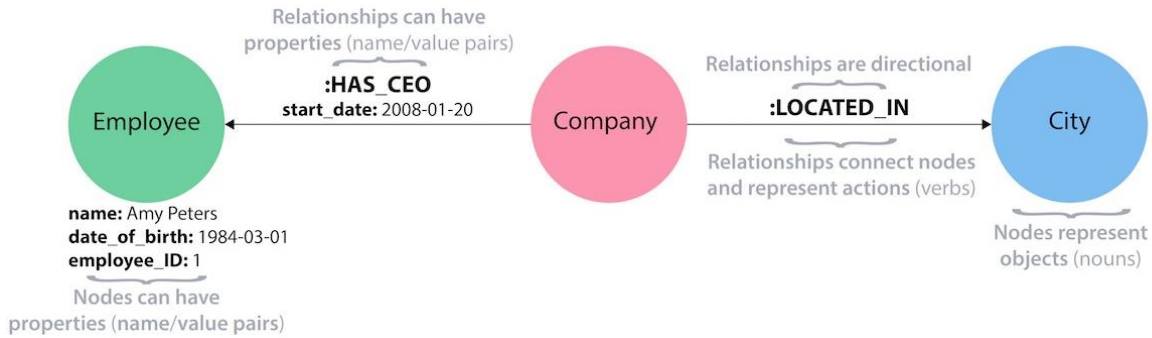
Pada basis data graf, data saling terhubung antara satu sama lain. Sebuah *edge* selalu memiliki *node* awal, *node* akhir, jenis, dan arah. Melihat karakteristik tersebut, dalam basis data graf hubungan antar entitas sama pentingnya dengan entitas itu sendiri. Hal ini memungkinkan permasalahan dunia nyata untuk dimodelkan dalam basis data graf. Selain itu, hubungan antar entitas yang tersimpan tadi, memungkinkan permintaan data diproses lebih cepat dikarenakan tidak diperlukannya JOIN tabel seperti pada basis data relasional (Bechberger & Perryman, 2020). Beberapa contoh platform basis data graf antara lain adalah

Amazon Neptune, Azure Cosmos DB, Oracle Spatial and Graph, OrientDB, DataStax, Dgraph, dan Neo4j.

2.6 *Labeled Property Graph*

Labeled Property Graph (LPG) adalah model data graf yang digunakan Neo4J. LPG sendiri merupakan salah satu jenis model data graf yang direpresentasikan dengan himpunan *node*, *relationship*, dan label. Secara non-formal, LPG termasuk *multigraph* berarah dan berlabel yang memiliki karakteristik khusus di mana setiap *node* dan *edge* dapat memiliki properti berisi himpunan pasangan *key-value*. Selain karakteristik khusus tersebut, ada beberapa karakteristik lain yang dimiliki LPG. Karakteristik-karakteristik tersebut antara lain sebagai berikut:

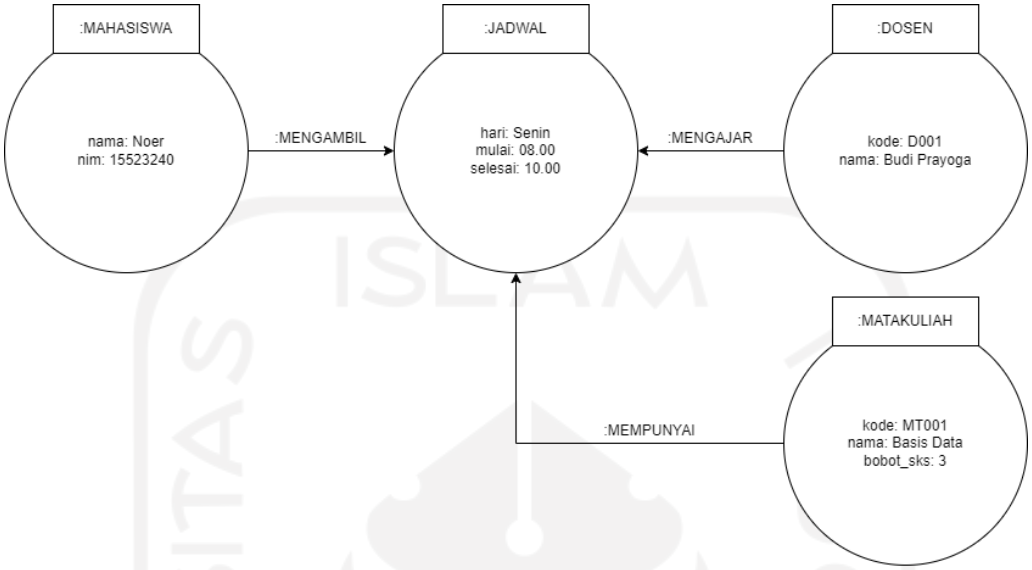
a. Memiliki *node* dan *relationship*.
b. *Node* memiliki *property* yang direpresentasikan dengan pasangan *key-value*.
c. *Node* dapat memiliki lebih dari satu label.
d. *Relationship* berlabel dan berarah, dan selalu memiliki *node* awal dan *node* akhir.
e. *Relationship* dapat memiliki *property* yang direpresentasikan dengan pasangan *key-value*.



Gambar 2.2 Ilustrasi *Labeled Property Graph*

Sebagai contoh, misalkan terdapat kebutuhan untuk melihat jadwal mata kuliah yang diambil mahasiswa dengan NIM tertentu berserta dosen yang mengampu mata kuliah tersebut. Berdasarkan kebutuhan tersebut, dapat diketahui beberapa hal yang dapat dimodelkan dalam bentuk LPG. Pertama, terdapat 4 *node* dengan label berbeda, yaitu MAHASISWA, DOSEN, MATAKULIAH, dan JADWAL. Kemudian terdapat 3 *relationship*, yaitu MENGAMBIL untuk *node* MAHASISWA ke *node* JADWAL, MENGAJAR untuk *node* DOSEN ke *node* JADWAL, dan MEMPUNYAI untuk *node* MATAKULIAH ke *node* JADWAL. Setelah

didapat *node*, *relationship*, dan label berdasarkan kebutuhan sebelumnya, hasil pemodelan dalam bentuk LPG dapat dilihat pada Gambar 2.3.



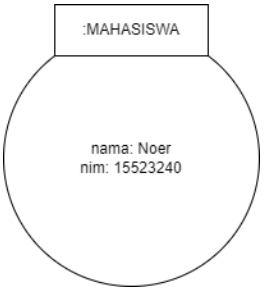
Gambar 2.3 Contoh *Labeled Property Graph*

2.7 *Cypher Query Language*

Cypher Query Language adalah bahasa *query* yang digunakan oleh Neo4J, bersamaan dengan digunakannya model data *Labeled Property Graph*. Berdasarkan dokumentasi Neo4J, Cypher merupakan bahasa *query* graf deklaratif yang memungkinkan penggunanya untuk melakukan *query*, pembaruan, dan administrasi terhadap basis data graf secara efisien dan ekspresif.

Cypher terinspirasi dari beberapa pendekatan yang berbeda dan dibangun berdasarkan penerapan yang sudah ada. Misalnya Cypher meminjam beberapa kata kunci dari bahasa lain seperti SQL, SPARQL, Haskell, dan Python. Cypher menggunakan sintaksis ASCII untuk mendeskripsikan pola visual graf. Sintaksis tersebut memberikan sebuah cara visual dan logis untuk mencocokkan pola *node* dan *relationship* dari sebuah graf.

Secara struktur, Cypher meminjam strukturnya dari SQL di mana *query* dibangun menggunakan klausa-klausa. Beberapa contoh klausa yang dapat digunakan untuk membaca data dari basis data graf antara lain adalah MATCH, WHERE, dan RETURN. Sebagai contoh, terdapat basis data graf dengan LPG seperti yang terlihat pada Gambar 2.4. *Query* yang dapat dibangun menggunakan klausa-klausa tersebut dapat dilihat pada Gambar 2.5.



Gambar 2.4 Contoh *Labeled Property Graph* Berlabel MAHASISWA

```
MATCH
    (noer:MAHASISWA)
WHERE
    noer.nim = '15523240'
RETURN
    noer

// atau

MATCH
    (noer:MAHASISWA { nim: '15523240' })
RETURN
    noer
```

Gambar 2.5 Contoh *Cypher Query Language*

Query Gambar 2.5 mengambil *node* berlabel MAHASISWA yang disimpan pada 'noer'. Kemudian dilakukan pencocokan pada 'noer' dengan batasan properti NIM bernilai 15523240. Setelahnya 'noer' akan dikembalikan. Jika data ditemukan maka 'noer' akan berisi *node* yang sesuai dengan batasan sebelumnya. Sebaliknya, jika data tidak ditemukan maka 'noer' akan berisi *string* kosong.

2.8 Neo4J

Neo4J adalah sebuah sistem manajemen basis data graf yang dikembangkan oleh Neo4J, Inc. Ditulis dalam bahasa Java, versi pertama Neo4J dirilis pada tahun 2010. Kode sumber Neo4J sendiri berlisensi GPLv3. Dalam operasinya, Neo4J menyimpan data dalam bentuk *node* dan *relationship* di mana masing-masing dapat memiliki *property* yang berisi pasangan *key-value*. Hal tersebut memungkinkan Neo4J untuk memproses permintaan dalam jumlah besar dengan performa tinggi.

Terdapat beberapa versi Neo4J yang dapat digunakan, antara lain sebagai berikut:

a. Neo4J AuraDB

Neo4J AuraDB adalah salah satu varian dari Neo4J yang dapat digunakan tanpa perlu memasang Neo4J pada suatu mesin (Neo4j Inc., 2022b). Neo4J menyediakan berbagai macam paket yang dapat pengguna pilih untuk menggunakan layanan AuraDB. Salah satu paket tersebut adalah paket gratis yang memiliki fitur satu basis data graf. (Neo4j Inc., 2022a)

b. Neo4J *Community Edition*

Neo4J Community Edition adalah salah satu varian dari Neo4J bagi pengguna yang dengan kebutuhan lebih. Varian ini dapat pengguna pasang pada suatu mesin.

c. Neo4J Desktop

Neo4J Desktop adalah salah satu varian dari Neo4J bagi pengguna yang baru menggunakan Neo4J. Kelebihan dari varian ini adalah adanya GUI yang dapat mempermudah pengguna untuk mengelola basis data graf Neo4J.

2.9 *Location Based Service*

Location Based Service (LBS) adalah istilah umum yang digunakan untuk layanan perangkat lunak yang menggunakan data dan informasi geografis untuk menyediakan layanan atau informasi kepada pengguna. LBS adalah layanan informasi yang dapat diakses dengan perangkat seluler melalui jaringan seluler. Layanan tersebut memanfaatkan kemampuan perangkat seluler untuk menunjukkan lokasi dari perangkat tersebut.

Menurut Steiniger, terdapat lima komponen penting yang diperlukan untuk menggunakan layanan berbasis lokasi (Steiniger et al., 2008). Komponen-komponen tersebut dapat dilihat pada Gambar 2.6. Komponen-komponen tersebut antara lain adalah sebagai berikut:

a. Mobile Device

Komponen pertama adalah perangkat yang berfungsi sebagai pengirim dan penerima permintaan informasi. Perangkat yang mungkin digunakan antara lain seperti PDA, Ponsel, dan Laptop. Informasi yang diterima dapat berupa suara, gambar maupun teks.

b. Positioning Component

Selanjutnya adalah komponen yang memiliki fungsi untuk menentukan posisi pengguna. Hal ini melihat informasi lokasi pengguna adalah informasi dasar yang diperlukan dalam layanan berbasis lokasi. Posisi pengguna bisa didapatkan melalui jaringan komunikasi maupun melalui GPS. Selain cara otomatis, pengguna juga dapat menentukan posisi secara manual.

c. *Communication Network*

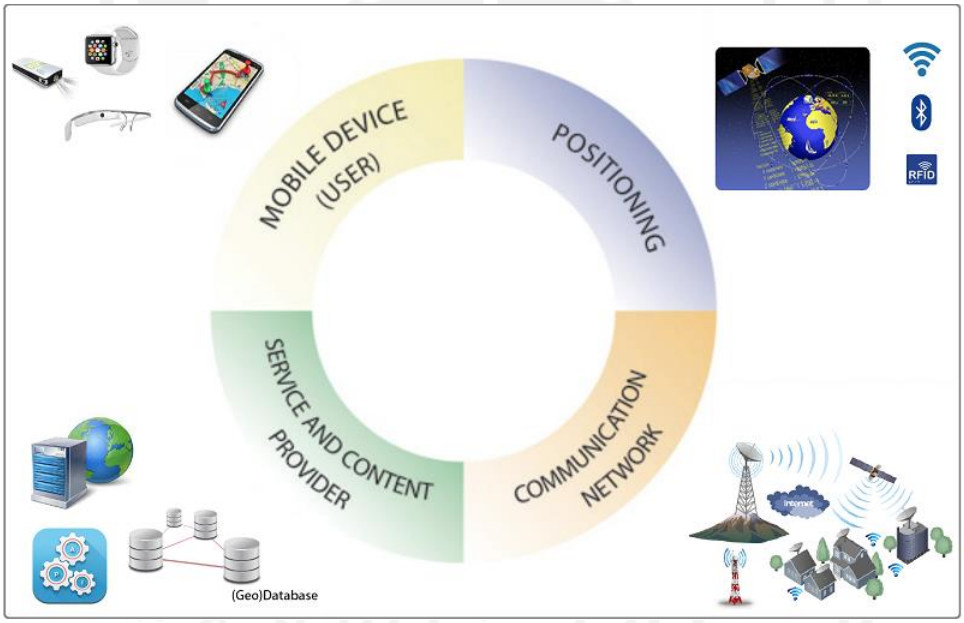
Komponen berikutnya adalah jaringan komunikasi yang memiliki fungsi sebagai jalur pertukaran permintaan informasi antara perangkat pengguna dengan penyedia layanan.

d. *Service and Application Provider*

Komponen lainnya adalah penyedia layanan yang bertanggung jawab terhadap pemrosesan permintaan informasi untuk setiap layanan yang disediakan. Layanan-layanan tersebut dapat berupa perhitungan posisi, pencarian rute, pencarian informasi spesifik dari suatu lokasi, dan lain sebagainya.

e. *Data and Content Provider*

Komponen terakhir adalah penyedia data dengan fungsi menyediakan data yang dibutuhkan oleh layanan untuk memenuhi permintaan pengguna.



Gambar 2.6 Komponen *Location Based Service*

2.10 *Geolocation*

Geopositioning, disebut juga *geolocation*, adalah proses penentuan atau perkiraan posisi geografis dari sebuah obyek. *Geolocation* menghasilkan koordinat geografis (garis lintang dan garis bujur). (Owusu et al., 2017)

2.11 *Geocoding*

Geocoding adalah sebuah proses pengubahan deskripsi berbasis teks dari sebuah lokasi, seperti alamat atau nama tempat, menjadi koordinat geografis, garis lintang dan garis bujur, untuk mengidentifikasi sebuah lokasi di permukaan bumi. Selain itu, ada pula *Reverse Geocoding* yang merupakan sebuah proses pengubahan koordinat geografis menjadi sebuah deskripsi sebuah lokasi. (Owusu et al., 2017)

2.12 *OpenStreetMap*

OpenStreetMap (OSM) adalah sebuah proyek kolaboratif basis data geografis dunia. Salah satu hasil dari proyek tersebut adalah peta dunia yang gratis. OSM dibangun oleh para relawan dan dirilis dengan lisensi *open-content*.

OSM menyediakan data dan layanan terkait peta dunia. Data tersebut dapat diakses melalui API. Sedangkan untuk layanan, terdapat beberapa layanan yang dapat digunakan pengguna, antara lain: *routing/navigation* dan *search/geocoding*.

Layanan *routing/navigation* menyediakan informasi mengenai rute dari suatu lokasi asal menuju lokasi tujuan. Sedangkan layanan *search/geocoding* menyediakan informasi terkait alamat dari suatu koordinat geografis dan sebaliknya.

2.13 *Haversine Formulae*

Rumus *Haversine* digunakan untuk menghitung jarak antara dua titik koordinat geografis pada permukaan bumi. Rumus *Haversine* didefinisikan sebagaimana yang tertera pada persamaan (2.1). Terdapat fungsi *Haversine* yang termasuk dalam rumus ini dengan pendefinisan seperti yang tertera pada persamaan (2.2)

$$hav\left(\frac{d}{r}\right) = hav(\varphi_2 - \varphi_1) + cos(\varphi_1) \cdot cos(\varphi_2) \cdot hav(\lambda_2 - \lambda_1) \quad (2.1)$$

$di\ mana$:
$d = jarak\ dalam\ meter$
$r = radius\ bumi = 6371000\ meter$
$hav = fungsi\ harvesine$
$\varphi_1, \varphi_2 = latitude\ titik\ 1\ dan\ latitude\ titik\ 2\ dalam\ radian$
$\lambda_1, \lambda_2 = longitude\ titik\ 1\ dan\ longitude\ titik\ 2\ dalam\ radian$

$$hav(\theta) = sin^2\left(\frac{\theta}{2}\right) \quad (2.2)$$

Berdasarkan persamaan (2.1) dan persamaan (2.2), untuk menghitung jarak d, digunakan *Inverse Haversine* pada $h = hav(\theta)$ seperti yang tertera pada persamaan (2.3).

$$\begin{aligned} d &= r \cdot archav(h) \\ &= 2r \cdot \arcsin\left(\sqrt{h}\right) \\ &= 2r \cdot \arcsin\left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \text{hav}(\lambda_2 - \lambda_1)}\right) \\ &= 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \end{aligned} \quad (2.3)$$

2.14 Tinjauan Aplikasi Sejenis

Penulis melakukan peninjauan pada beberapa aplikasi untuk melihat fitur apa saja yang terdapat pada aplikasi rute BRT Trans Jogja. Beberapa aplikasi yang penulis gunakan dalam tinjauan ini dapat dilihat pada Tabel 2.5

Tabel 2.5 Keterangan Aplikasi Sejenis

Aplikasi	Pengembang	Versi	Terakhir Diperbarui	Platform
Trans Jogja	PT Nusantara Global Inovasi	2.0.8	7 Juli 2022	Android
Trans Jogja Route	INFINITE UNY	1.0	28 Januari 2020	Android
Teman Bus	DIREKTORAT ANGKUTAN JALAN KEMENHUB	2.1.220722_12	21 Juli 2022	Android
Trans Semarang	Badan Layanan Umum Trans Semarang	2.1.8	18 Juli 2022	Android

Berdasarkan peninjauan tersebut, berikut fitur yang ada pada setiap aplikasi yang penulis temukan:

a. Trans Jogja, PT Nusantara Global Inovasi

Aplikasi Trans Jogja memiliki fitur seperti rekomendasi rute berdasarkan tujuan dan asal yang dapat ditentukan oleh pengguna, informasi rute setiap koridor beserta *shelter* dan armada yang tersedia, serta informasi bus terdekat dari lokasi pengguna. Selain itu, ada beberapa fitur tambahan, seperti AR *shelter*, pembelian tiket, dan riwayat perjalanan. Namun, pada aplikasi ini masih ditemukan beberapa kekurangan. Untuk dapat melihat informasi terkait BRT Trans Jogja pengguna perlu memasang aplikasi pada perangkat terlebih dahulu. Untuk dapat melihat informasi terkait jenis dan harga tiket yang tersebut, pengguna perlu *login* terlebih dahulu. Aplikasi hanya menyediakan informasi salah satu layanan BRT di Yogyakarta, yaitu Trans Jogja Istimewa. Tidak adalah detail informasi *shelter*.

b. Trans Jogja *Route,* Infinite UNY

Aplikasi Trans Jogja *Route* menyediakan informasi perihal koridor dalam bentuk gambar. Tersedia *form* pencarian lokasi tujuan yang nantinya akan dialihkan ke layanan Google Maps Direction. Namun, pada aplikasi ini masih ditemukan beberapa kekurangan. Untuk dapat melihat informasi terkait BRT Trans Jogja pengguna perlu memasang aplikasi pada perangkat terlebih dahulu. Aplikasi hanya menyediakan informasi salah satu layanan BRT di Yogyakarta, yaitu Trans Jogja Istimewa. Informasi rute koridor masih berupa gambar secara garis besar, tidak ditampilkan dalam bentuk peta. Selain itu, tidak ada informasi *shelter* mana saja yang termasuk dalam suatu rute koridor. Untuk penentuan rute aplikasi masih menggunakan antarmuka Google Maps Direction, bukan API. Aplikasi hanya menggunakan lokasi asal dan lokasi tujuan pengguna yang selanjutnya data tersebut akan

dikirimkan ke Google Maps Direction. Sebagian besar informasi *shelter* dan koridor berasal dari Google Maps.

c. Teman Bus, Direktorat Angkutan Jalan Kemenhub RI

Aplikasi Teman Bus memiliki fitur rekomendasi rute berdasarkan tujuan dan asal yang dapat ditentukan oleh pengguna. Selain itu, terdapat pencarian bus yang tersedia berdasarkan rekomendasi rute yang dipilih oleh pengguna. Tersedianya jadwal setiap armada untuk masing-masing koridor. Ada fitur lain seperti riwayat perjalanan, berita, dan pembayaran. Namun, pada aplikasi ini masih ditemukan beberapa kekurangan. Untuk dapat melihat informasi terkait BRT Trans Jogja pengguna perlu memasang aplikasi pada perangkat terlebih dahulu. Aplikasi hanya menyediakan informasi salah satu layanan BRT di Yogyakarta, yaitu Teman Bus. Peta jaringan yang disediakan oleh aplikasi masih berupa gambar. Informasi rute koridor dan *shelter* yang dilewati berada pada menu jadwal.

d. Trans Semarang, Badan Layanan Umum Trans Semarang

Aplikasi Trans Semarang memiliki beberapa fitur antara lain rekomendasi rute berdasarkan tujuan dan asal yang dapat ditentukan oleh pengguna, informasi bis terdekat, serta rute armada berdasarkan koridor dengan menampilkan *shelter* yang termasuk dalam rute tersebut. Terdapat pula fitur tambahan lainnya seperti AR *shelter*, tiket bis, dan pariwisata. Namun, pada aplikasi ini masih ditemukan beberapa kekurangan. Untuk dapat melihat informasi terkait BRT Trans Jogja pengguna perlu memasang aplikasi pada perangkat terlebih dahulu. Aplikasi hanya menyediakan informasi salah satu layanan BRT di Semarang, yaitu Trans Semarang. Untuk dapat melihat informasi terkait jenis dan harga tiket yang tersebut, pengguna perlu *login* terlebih dahulu. Tidak adalah detail informasi *shelter*.

Melihat hasil tinjauan pada aplikasi yang penulis temukan, dapat disimpulkan beberapa fitur utama yang ada pada aplikasi rute BRT Trans Jogja, antara lain:

a. Memiliki fitur rekomendasi rute berdasarkan asal dan tujuan.
b. Memiliki fitur melihat *shelter* terdekat.
c. Memiliki fitur melihat rute untuk setiap koridor.
d. Memiliki fitur jadwal armada yang tersedia untuk setiap koridor.

Selain itu, pada aplikasi yang penulis tinjau, sebagian besar masih memiliki kekurangan. Kekurangan-kekurangan yang penulis temukan antara lain:

a. Aplikasi berbasis *mobile* sehingga memerlukan pengguna untuk memasang aplikasi pada perangkat yang digunakan.

b. Aplikasi hanya menyediakan informasi salah satu layanan BRT yang ada di Yogyakarta.
c. Tidak ada fitur untuk melihat detail informasi *shelter*.

Melihat hasil tinjauan aplikasi-aplikasi tersebut, berikut fitur-fitur yang dapat penulis tambahkan pada aplikasi yang dikembangkan, antara lain:

a. Adanya fitur untuk melihat informasi *shelter* BRT Trans Jogja, baik secara keseluruhan, *shelter* berdasarkan rute, maupun *shelter* berdasarkan koridor.
b. Adanya fitur untuk melihat informasi *shelter* BRT Trans Jogja secara detail.
c. Tersedianya semua koridor dan *shelter* yang termasuk dalam layanan BRT di Yogyakarta, yaitu Trans Jogja Istimewa dan Teman Bus.
d. Aplikasi berbasis web sehingga pengguna tidak perlu memasang aplikasi pada perangkan untuk mengakses informasi terkait BRT Trans Jogja.

BAB III
METODOLOGI

3.1 Kajian Pustaka

Dalam penelitian ini, diperlukan data dan informasi yang berkaitan dengan BRT Trans Jogja. Data dan informasi tersebut didapatkan dari situs remis Dinas Perhubungan Provinsi Daerah Istimewa Yogyakarta (Dishub DIY). Baik dari informasi yang terdapat pada situs tersebut, dokumen resmi yang dapat diunduh publik, maupun Sistem Informasi Lalu Lintas Angkutan Jalan Dishub DIY.

3.2 Analisis Kebutuhan

3.2.1 Identifikasi Pengguna

Pada pengembangan aplikasi ini, dari kasus yang ada penulis mengidentifikasi hanya ada satu jenis pengguna. Pengguna tersebut adalah siapa saja yang menggunakan aplikasi yang dikembangkan ini. Pengguna dapat menggunakan aplikasi untuk mencari rute, melihat informasi *shelter*, koridor, dan informasi tambahan lainnya yang berkaitan dengan BRT Trans Jogja. Ketiadaan jenis pengguna dengan fungsi pengelolaan didasarkan pada data yang digunakan. Pada aplikasi ini, tidak ada kebutuhan atas pembuatan data baru begitu juga dengan pengubahan dan penghapusan data yang sudah ada.

3.2.2 Analisis Kebutuhan Proses

Berdasarkan analisis kebutuhan sebelumnya, berikut adalah proses-proses yang ada di dalam aplikasi:

a. Proses Melihat *Shelter* Terdekat
b. Proses Melihat Rekomendasi Rute
c. Proses Melihat Detail Rekomendasi Rute
d. Proses Melihat Informasi *Shelter*
e. Proses Melihat Informasi Koridor
f. Proses Melihat Informasi Tambahan
g. Proses Melihat Lokasi Pengguna

3.3 Pemodelan Domain Basis Data

Pemodelan basis data adalah tahap yang perlu dilalui dalam pengembangan aplikasi ini. Tahapan ini ditujukan agar basis data yang digunakan nantinya adalah basis data graf yang efektif, efisien, dan optimal, serta dapat memenuhi kebutuhan yang ada. Pemodelan dilakukan menggunakan model data *Labeled Property Graph*.

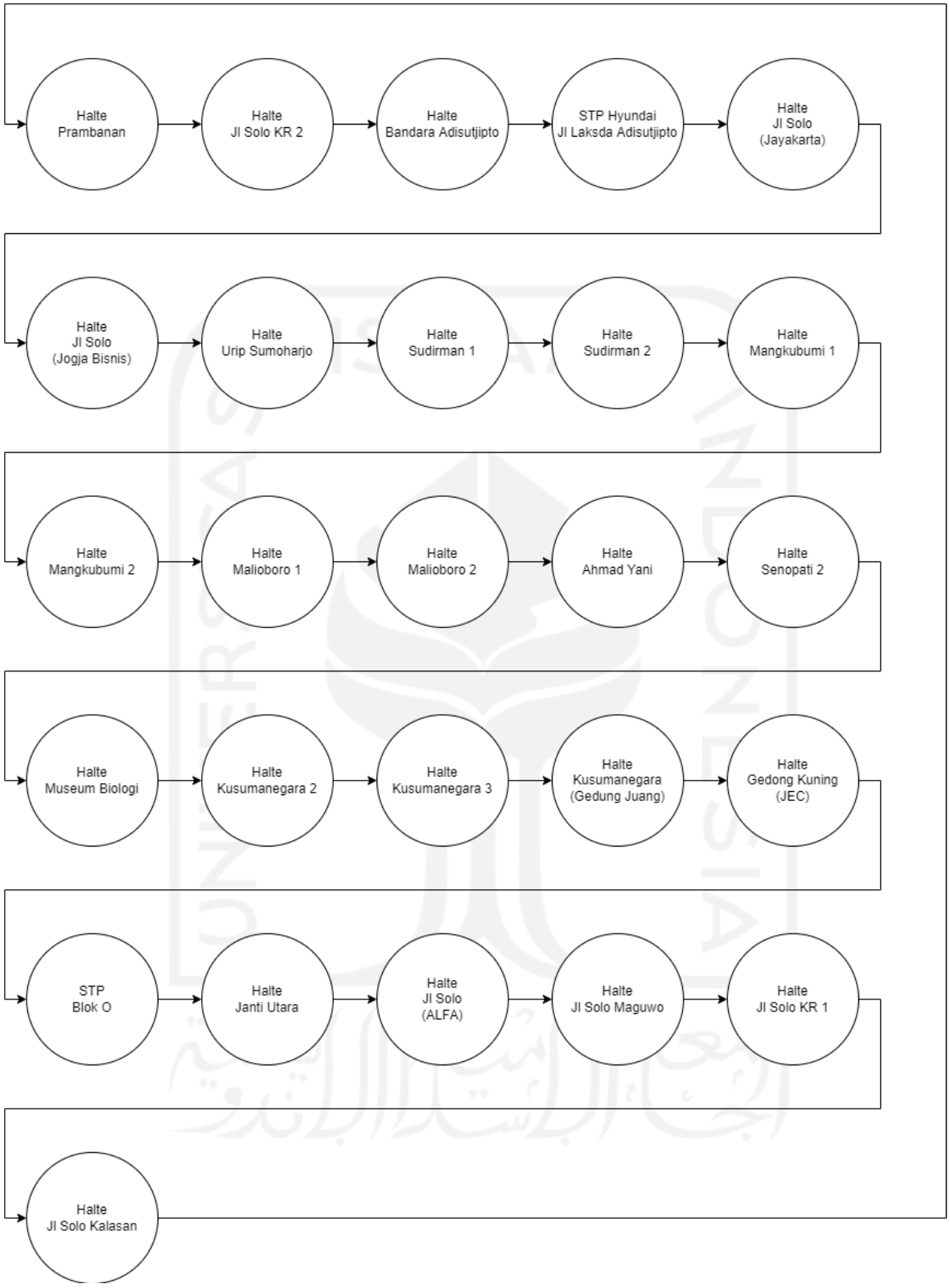
Dalam studi kasus BRT Trans Jogja, sebagai sebuah sistem transportasi, terdapat entitas yang mendukung keberlangsungan jalanannya sistem tersebut. *Shelter* yang menjadi titik pemberhentian armada dan naik-turunnya penumpang merupakan salah bagian dari sistem tersebut. Selain itu, terdapat pula koridor yang menunjukkan rute yang akan dilalui suatu armada. Dalam suatu koridor, armada akan melewati *shelter-shelter* dengan urutan tertentu. Suatu *shelter* bisa saja termasuk dalam satu atau lebih rute koridor.

Sebagai contoh, misalkan koridor 1A yang memiliki rute Bandara Adisutjipto — Pasar Pathuk. Untuk menyelesaikan satu putaran penuh rute koridor 1A, suatu armada akan melewati sebanyak 26 *shelter* mulai dari Halte Prambanan menuju Halte Jl. Solo Kalasan dengan urutan yang sudah ditentukan sebelumnya. *Shelter-shelter* yang termasuk dalam rute koridor 1A dapat dilihat pada Tabel 3.1.

Tabel 3.1 *Shelter-Shelter* yang Termasuk dalam Rute Koridor 1A

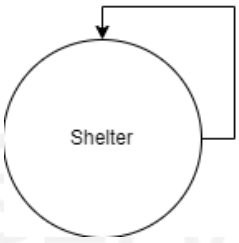
No	Nama *Shelter*	Jenis *Shelter*
1	Halte Prambanan	Permanen
2	Halte Jl Solo KR 2	Permanen
3	Halte Bandara Adisutjipto	Permanen
4	STP Hyundai Jl Laksda Adisutjipto	Portable
5	Halte Jl Solo (Jayakarta)	Permanen
6	Halte Jl Solo (Jogja Bisnis)	Permanen
7	Halte Urip Sumoharjo	Permanen
8	Halte Sudirman 1	Permanen
9	Halte Sudirman 2	Permanen
10	Halte Mangkubumi 1	Permanen
11	Halte Mangkubumi 2	Permanen
12	Halte Malioboro 1	Permanen
13	Halte Malioboro 2	Permanen
14	Halte Ahmad Yani	Permanen
15	Halte Senopati 2	Permanen
16	Halte Museum Biologi	Permanen
17	Halte Kusumanegara 2	Permanen
18	Halte Kusumanegara 3	Permanen
19	Halte Kusumanegera (Gedung Juang)	Permanen
20	Halte Gedong Kuning (JEC)	Permanen
21	STP Blok O	Portable
22	Halte Janti Utara	Permanen
23	Halte Jl Solo (ALFA)	Permanen
24	Halte Jl Solo Maguwo	Permanen
25	Halte Jl Solo KR 1	Permanen
26	Halte Jl Solo Kalasan	Permanen

Halte Prambanan termasuk dalam rute koridor 1A hanya ketika tujuan pemberhentian selanjutnya adalah Halte Jl Solo KR 2. Hal ini menunjukkan bahwa koridor yang menghubungkan suatu *shelter* dengan *shelter* lainnya merupakan hubungan yang penting. Selanjutnya, dapat dilakukan identifikasi entitas untuk memodelkan koridor 1A dalam bentuk graf. *Shelter-shelter* yang termasuk dalam koridor 1A dapat direpresentasikan dengan *node*. Sedangkan hubungan antara suatu *shelter* dengan *shelter* yang menjadi tujuan pemberhentian selanjutnya dapat direpresentasikan dengan *edge* yang menghubungkan dua *node shelter*. Model graf koridor 1A dapat dilihat pada Gambar 3.1.



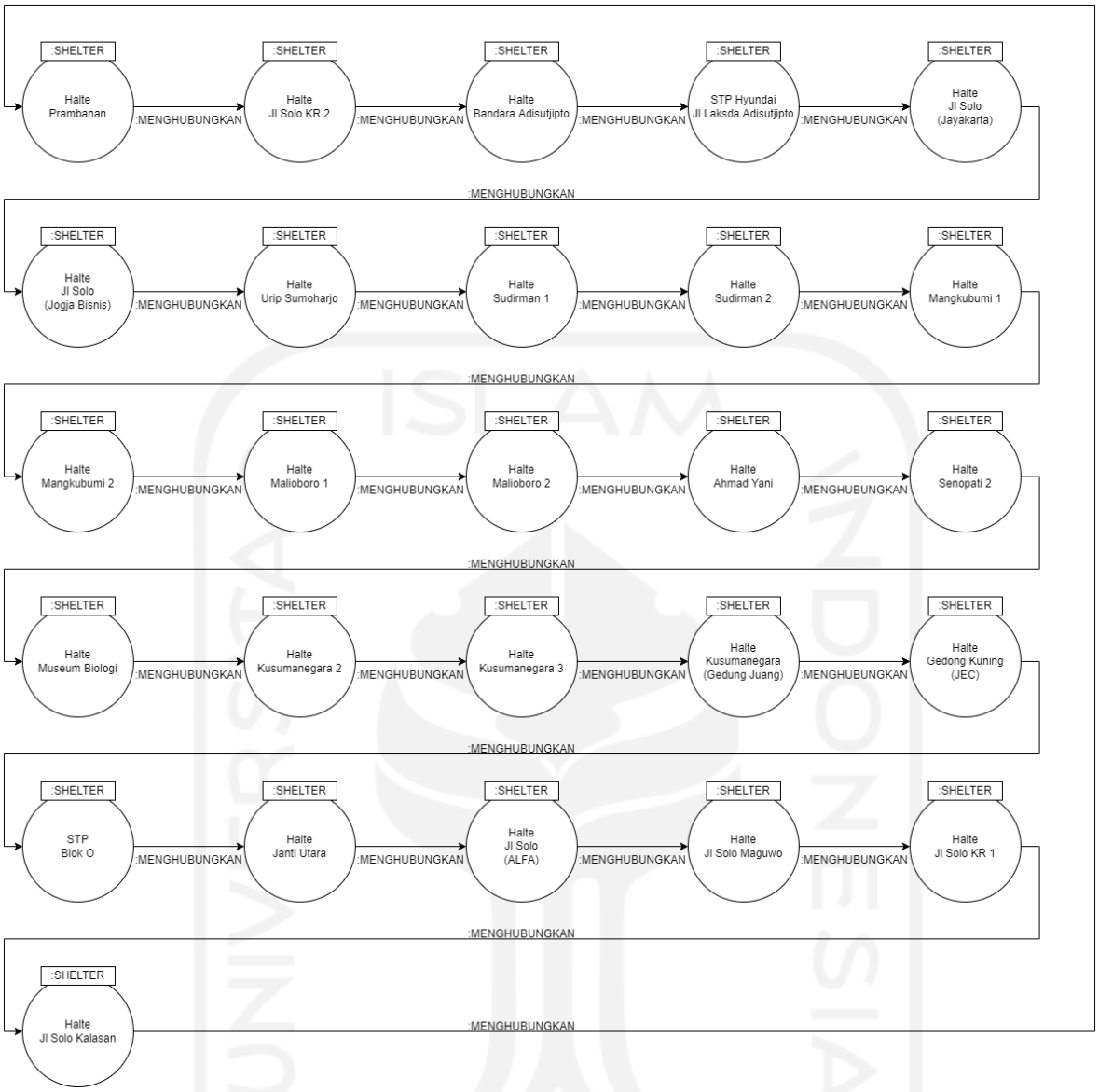
Gambar 3.1 Graf Koridor 1A

Melihat *node-node* yang pada dasarnya merupakan *shelter*, maka dapat dilakukan penyederhanaan dengan mengganti *node-node* tersebut menjadi satu *node shelter*. Hasil penyederhanaan dapat dilihat pada Gambar 3.2.

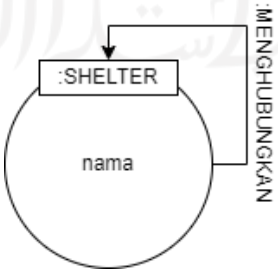


Gambar 3.2 Graf Koridor BRT Trans Jogja

Setelah diketahui bagaimana hubungan antara koridor dengan *shelter* dalam BRT Trans Jogja dan memodelkannya dalam bentuk graf, selanjutnya dapat dilakukan pemodelan menggunakan *Labeled Property Graph*. Sama seperti pemodelan dalam bentuk graf, perlu dilakukan identifikasi entitas untuk melakukan pemodelan menggunakan *Labeled Property Graph*. Berdasarkan Gambar 3.1 dan Gambar 3.2, suatu *shelter* dapat direpresentasikan dengan *node*. Sedangkan *edge* yang menghubungkan dua *shelter* dapat direpresentasikan sebagai *relationship*. Entitas selanjutnya yang perlu diidentifikasi adalah *label*. Node yang merupakan *shelter* dapat diberi label :SHELTER dan *relationship* antara dua *node* :SHELTER dapat diberi label :MENGHUBUNGKAN. Selain itu, terdapat pula entitas *property* yang bisa diidentifikasi dari data Trans Jogja milik SI-LLAJ. Setelah dilakukan identifikasi entitas, koridor 1A dapat dimodelkan menggunakan LPG seperti yang terlihat pada Gambar 3.3 dengan penyederhanaan menjadi Gambar 3.4.



Gambar 3.3 *Labeled Property Graph* Koridor 1A

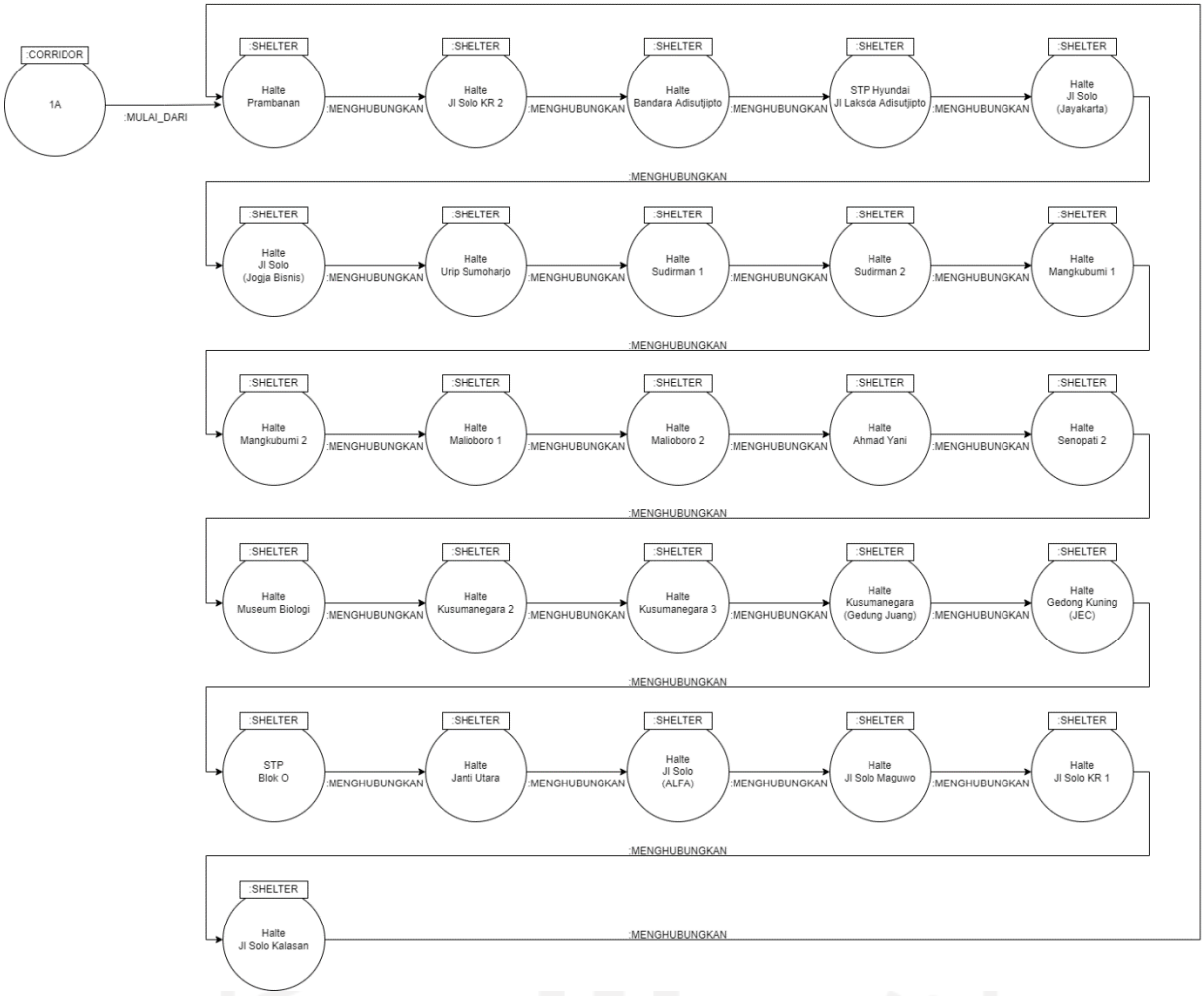


Gambar 3.4 *Labeled Property Graph* Koridor BRT Trans Jogja

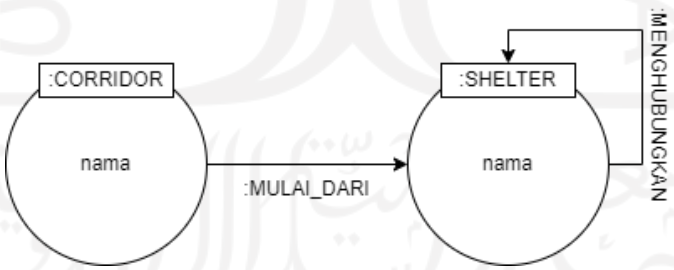
Labeled Property Graph pada Gambar 3.3 dan Gambar 3.4 sudah dapat merepresentasikan rute transportasi BRT Trans Jogja secara umum. Namun, melihat kebutuhan yang ada pada penelitian ini, kebutuhan untuk melihat *shelter* awal dari suatu koridor belum dapat terpenuhi. Untuk mengatasi masalah tersebut, dapat dilakukan penambahan entitas *node* berlabel :CORRIDOR dan *relationship* berlabel :MULAI_DARI yang menghubungkan *node* :CORRIDOR dengan *node* :SHELTER sebagai *shelter* awal suatu koridor. Hasil akhir identifikasi entitas dapat dilihat pada Tabel 3.2. Sedangkan, hasil pengubahan terhadap LPG koridor 1A dapat dilihat pada Gambar 3.5yang kemudian disederhanakan menjadi Gambar 3.6.

Tabel 3.2 Label dan Jenis Hasil Pemodelan Basis Data Graf

Label	**Jenis**	**Keterangan**
SHELTER	*Node*	Data Shelter
CORRIDOR	*Node*	Data Koridor
MENGHUBUNGKAN	*Relationship*	Hubungan antara *shelter* dengan *shelter*
MULAI_DARI	*Relationship*	Hubungan antara koridor dengan *shelter*

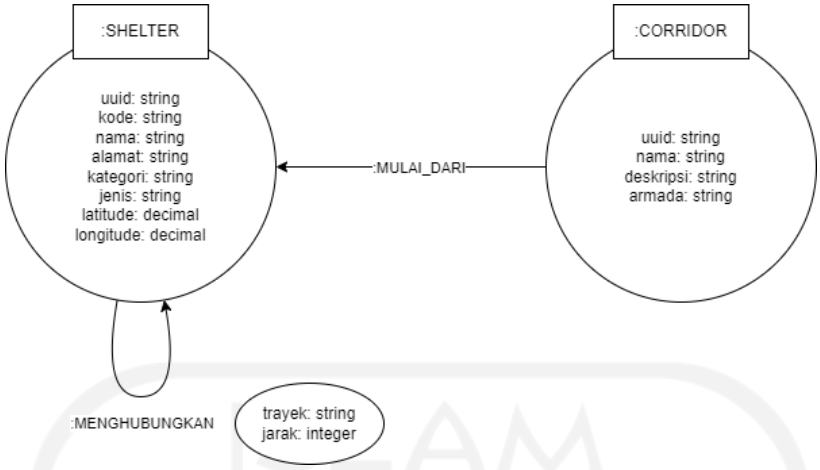


Gambar 3.5 Hasil Pemodelan Koridor 1A



Gambar 3.6 Hasil Pemodelan Koridor BRT Trans Jogja

Setelah menambahkan *property* untuk masing-masing entitas, hasil akhir pemodelan LPG untuk BRT Trans Jogja dapat dilihat pada Gambar 3.7.



Gambar 3.7 Hasil Pemodelan Rute Transportasi BRT Trans Jogja

Untuk memperjelas, berikut properti untuk masing-masing *node* dan *relationship* yang tertera pada Gambar 3.7.

a. *Node* SHELTER

Node berlabel SHELTER memiliki *property* UUID, Kode, Nama, Alamat, Kategori, Jenis, *Latitude*, dan *Longitude*. *Property Latitude* dan *Longitude* ditambahkan berdasarkan kebutuhan untuk mendapatkan *shelter* berdasarkan lokasi. Selain itu, properti ini digunakan untuk menandai *shelter* pada peta. Detail untuk setiap *property node* tersebut dapat dilihat pada Tabel 3.3.

Tabel 3.3 *Property Node* Berlabel SHELTER

Properti	Jenis	Keterangan
uuid	*string*	*Property* ini menyimpan data unik yang berfungsi sebagai penanda untuk tiap *shelter*.
kode	*integer*	*Property* ini menyimpan data terkait kode dari *shelter* terkait.
nama	*string*	*Property* ini menyimpan data terkait nama dari *shelter* terkait.
jenis_halte	*string*	*Property* ini menyimpan data terkait jenis dari *shelter* terkait.
kategori_halte	*string*	*Property* ini menyimpan data terkait kategori dari *shelter* terkait.
sumber_listrik	*string*	*Property* ini menyimpan data terkait sumber listrik dari *shelter* terkait.
sumber_dana	*string*	*Property* ini menyimpan data terkait sumber dana dari *shelter* terkait.
tahun	*integer*	*Property* ini menyimpan data terkait tahun dari *shelter* terkait.
kondisi	*string*	*Property* ini menyimpan data terkait kondisi dari shelter terkait.
keterangan	*string*	*Property* ini menyimpan data terkait keterangan dari *shelter* terkait.
latitude	*float*	*Property* ini menyimpan data terkait koordinat *latitude* dari *shelter* terkait.
longitude	*float*	*Property* ini menyimpan data terkait koordinat *longitude* dari *shelter* terkait.

b. *Node* CORRIDOR

Node berlabel CORRIDOR memiliki *property* UUID, Nama, Deskripsi, dan Armada. Detail untuk setiap *property node* tersebut dapat dilihat pada Tabel 3.4.

Tabel 3.4 *Property Node* Berlabel CORRIDOR

Properti	Jenis	Keterangan
uuid	*string*	*Property* ini menyimpan data unik yang berfungsi sebagai penanda untuk tiap koridor.
nama	*string*	*Property* ini menyimpan data terkait nama dari koridor terkait.
deskripsi	*string*	*Property* ini menyimpan data terkait awal dan akhir rute koridor terkait.
armada	*integer*	*Property* ini menyimpan data terkait jumlah armada yang terdapat pada koridor terkait.

c. *Relationship* MENGHUBUNGKAN

Relationship berlabel MENGHUBUNGKAN memiliki *property* Trayek dan Jarak. Detail untuk setiap *property relationship* tersebut dapat dilihat pada Tabel 3.5.

Tabel 3.5 *Property Relationship* Berlabel MENGHUBUNGKAN

Properti	Jenis	Keterangan
trayek	*string*	*Property* ini menyimpan data terkait koridor untuk *node* SHELTER yang terhubung pada *relationship* terkait.
jarak	*float*	*Property* ini menyimpan data terkait jarak antar *node* SHELTER yang terhubung pada *relationship* terkait.

d. *Relationship* MULAI_DARI

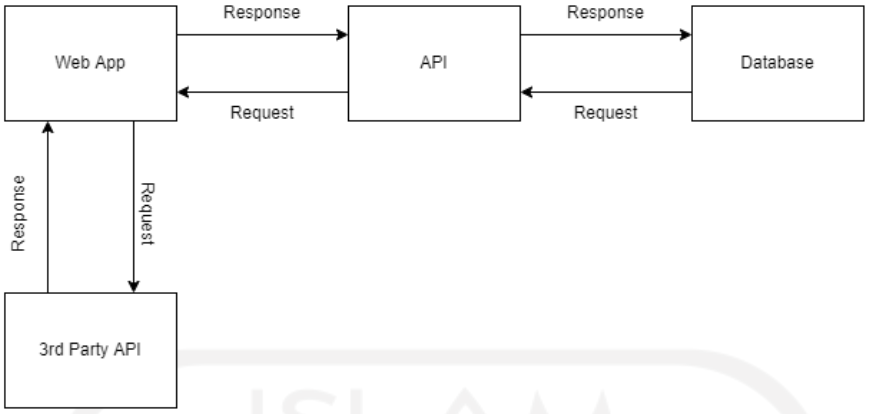
Relationship berlabel MULAI_DARI tidak memiliki *property*.

3.4 Perancangan Aplikasi

3.4.1 Perancangan Arsitektur Sistem

Perancangan arsitektur sistem merupakan tahapan yang perlu dilalui dalam pengembangan aplikasi ini. Pada arsitektur sistem ini terdapat empat komponen yaitu basis data, API yang merupakan *backend*, aplikasi web yang merupakan klien, dan API pihak ketiga. Tiap-tiap komponen berkomunikasi berdasarkan permintaan untuk kemudian memberikan respons kepada komponen yang melakukan permintaan.

Fokus utama dari aplikasi ini adalah pengimplementasian basis data graf pada rute BRT Trans Jogja. Untuk memenuhi kebutuhan tersebut digunakan Neo4J yang merupakan platform basis data graf. Selanjutnya, terdapat API yang berfungsi sebagai jalur komunikasi antara aplikasi web dengan basis data. API tersebut dikembangkan menggunakan ExpressJS yang berjalan di atas NodeJS. Untuk aplikasi web, dalam pengembangannya digunakan ReactJS bersama dengan Redux untuk manajemen *state*. Selain itu, digunakan pula Leaflet untuk menampilkan map dengan React-Leaflet sebagai *driver* antara Leaflet dan React. Komponen terakhir dalam yang digunakan dalam pengembangan aplikasi ini adalah API pihak ketiga. Digunakan *OpenStreetMap* untuk penyedia data map yang ditampilkan melalui Leaflet. Digunakan pula Leaflet-Routing-Machine untuk keperluan terkait rute. Secara umum, rancangan arsitektur sistem dari aplikasi yang dikembangkan dapat dilihat pada Gambar 3.8.

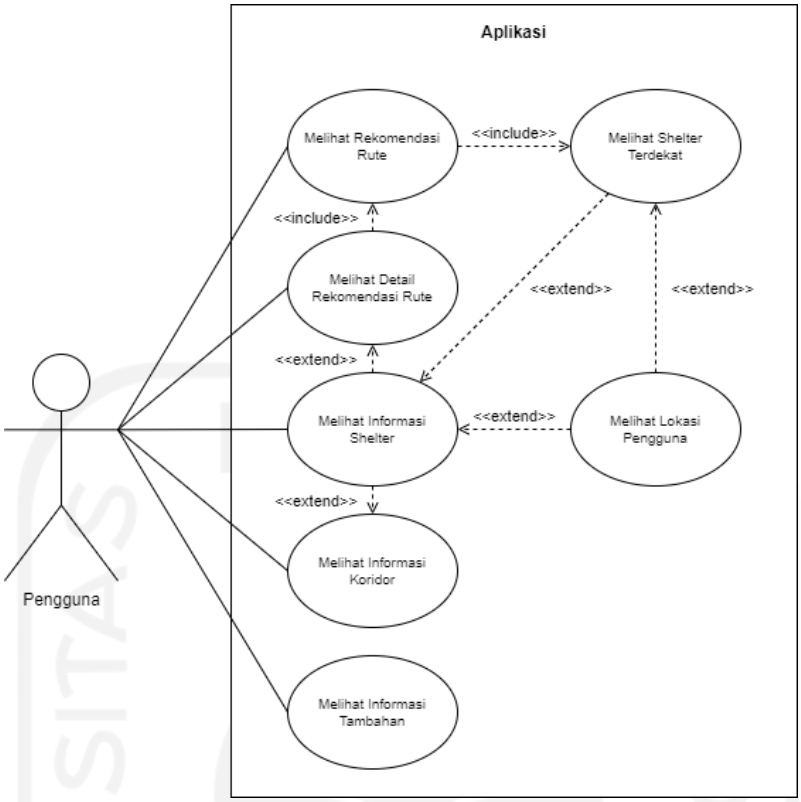


Gambar 3.8 Arsitektur Sistem Aplikasi yang Dikembangkan

3.4.2 Perancangan Fungsionalitas

Perancangan fungsionalitas merupakan tahapan di mana apa saja yang akan dilakukan oleh aplikasi dirancang. Untuk memodelkan hal tersebut digunakanlah *Use Case Diagram*. untuk mendeskripsikan bagaimana sistem merespons suatu permintaan dari suatu pengguna dengan kondisi-kondisi tertentu untuk mencapai suatu tujuan. Dengan kata lain, *Use Case Diagram* digunakan untuk menggambarkan interaksi antara pengguna dengan sistem.

Berdasarkan identifikasi pengguna yang telah dilakukan sebelumnya, hanya terdapat satu pengguna pada aplikasi yang dikembangkan. Pengguna memiliki beberapa aksi yang dapat dilakukan terhadap aplikasi. Aksi-aksi tersebut antara lain melihat *shelter* terdekat, rekomendasi rute beserta detailnya, informasi *shelter* beserta detailnya, informasi koridor beserta detailnya, informasi tambahan, dan lokasi terkini dari pengguna. Diagram pemodelan fungsionalitas ini dapat dilihat pada Gambar 3.9.



Gambar 3.9 Diagram *Use Case*

3.4.3 Perancangan Perilaku

Pemodelan perilaku dilakukan berdasarkan perancangan fungsionalitas yang telah dilakukan. Pada tahapan ini, dilakukan perancangan terkait bagaimana sistem bekerja. Alur kerja tersebut kemudian dimodelkan menggunakan *Activity Diagram*. Berikut ini adalah *Activity Diagram* berdasarkan *Use Case Diagram* seperti yang tertera pada Gambar 3.9.

Melihat *Shelter* Terdekat (UC01)

Proses melihat *shelter* terdekat adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait *shelter* terdekat dari lokasi yang ditentukan oleh pengguna. Proses tersebut dimulai ketika aplikasi menampilkan halaman awal. Selanjutnya, pengguna perlu menentukan lokasi asal dan lokasi tujuan. Setelah lokasi ditentukan, pencarian *shelter* terdekat dilakukan oleh pengguna. Kemudian, aplikasi akan melakukan pencarian *shelter* terdekat dari lokasi asal dan lokasi tujuan berdasarkan lokasi yang ditentukan sebelumnya.

Pencarian *shelter* terdekat dilakukan menggunakan rumus *Haversine* untuk mencari jarak antara dua titik dengan lokasi asal dan tujuan sebagai titik pertama dengan *shelter* yang dicari

sebagai titik kedua. Suatu *shelter* dapat dikatakan sebagai *shelter* yang dekat dengan suatu lokasi apabila antara jarak *shelter* tersebut dengan suatu lokasi kurang dari sama dengan 1000 meter. Sehingga, untuk mengetahui shelter-shelter yang termasuk dalam jangkauan tersebut dapat digunakan rumus *Haversine* seperti yang tertera pada persamaan (3.1).

$$S = 2r \cdot arcsin\left(\sqrt{\sin^2\left(\frac{\varphi - \alpha}{2}\right) + \cos\alpha \cdot \cos\varphi \cdot \sin^2\left(\frac{\lambda - \beta}{2}\right)}\right) \quad (3.1)$$

$Jika\ S \leq 1000, maka\ shelter\ S\ termasuk\ shelter\ yang\ dekat.$

$di\ mana$:
$S = jarak\ shelter\ S\ dengan\ lokasi\ asal/tujuan\ dalam\ meter$
$r = radius\ bumi = 6371000\ meter$
$\alpha = latitude\ lokasi\ asal/tujuan\ dalam\ radian$
$\beta = longitude\ lokasi\ asal/tujuan\ dalam\ radian$
$\varphi = latitude\ shelter\ i\ dalam\ radian$
$\lambda = longitude\ shelter\ i\ dalam\ radian$

Sebagai contoh, diketahui terdapat lokasi asal dengan titik koordinat *latitude-longitude* -7.757827542422809, 110.38649809310787 dan sebuah shelter dengan titik koordinat -7.753745356928487, 110.38398754572147. Untuk menghitung jarak shelter tersebut dengan lokasi asal, dapat digunakan rumus *Haversine* dengan perhitungan seperti yang tertera pada persamaan (3.3). Sebelumnya, titik koordinat diubah terlebih dahulu menjadi radian seperti yang tertera pada persamaan (3.2).

$$\begin{aligned}
\alpha &= -7.757827542422809 \cdot \pi/180 = -0.1353996334171781 \\
\varphi &= -7.753745356928487 \cdot \pi/180 = -0.13532838583962503 \\
\beta &= 110.38649809310787 \cdot \pi/180 = 1.9266078414711745 \\
\lambda &= 110.38398754572147 \cdot \pi/180 = 1.9265640241532545
\end{aligned} \quad (3.2)$$

$$S = 2r \cdot arcsin\left(\sqrt{\sin^2\left(\frac{\varphi-\alpha}{2}\right) + \cos\alpha \cdot \cos\varphi \cdot \sin^2\left(\frac{\lambda-\beta}{2}\right)}\right) \quad (3.3)$$

$$= 2r \cdot \arcsin\left(\sqrt{\begin{matrix}\sin^2\left(\frac{0.0000712475775530 5744}{2}\right) + \\ \cos(-0.1353996334171781) \cdot \\ \cos(-0.13532838583962503) \cdot \\ \sin^2\left(\frac{-0.00004381731792002519}{2}\right)\end{matrix}}\right)$$

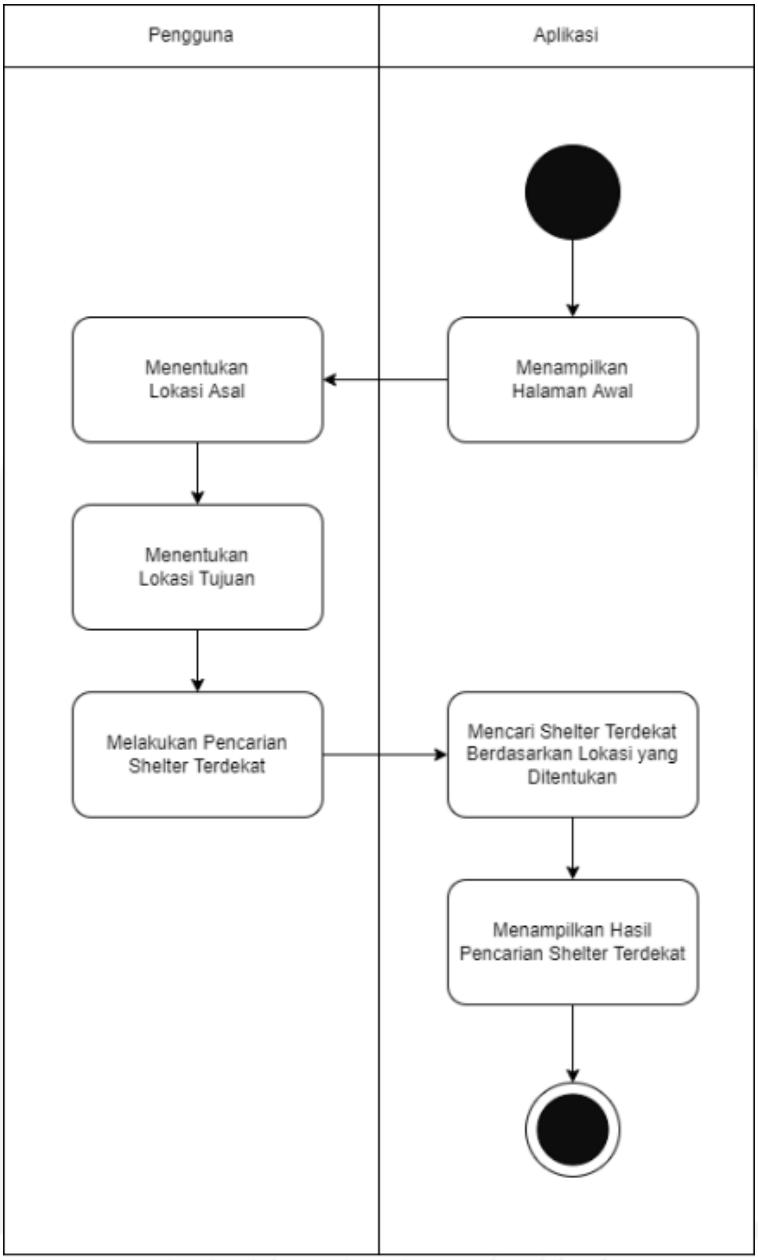
$$= 2r \cdot \arcsin\left(\sqrt{\begin{matrix}\sin^2(0.00003562378877652872) + \\ \cos(-0.1353996334171781) \cdot \\ \cos(-0.13532838583962503) \cdot \\ \sin^2(-0.00002190865896 0012595)\end{matrix}}\right)$$

$$= 2r \cdot \arcsin\left(\sqrt{1.7403022064720792}\right)$$

$$= 2 \cdot 6371000 \cdot 0.00004171692950 6875615$$

$$= 531.5571157766091$$

Berdasarkan perhitungan pada persamaan (3.3), jarak antara shelter dan lokasi asal adalah 531.56 meter. Sehingga, dapat dikatakan shelter termasuk dekat dengan lokasi asal. Proses berakhir ketika hasil pencarian untuk masing-masing lokasi ditampilkan oleh aplikasi kepada pengguna. *Activity Diagram* untuk proses ini dapat dilihat pada Gambar 3.10.

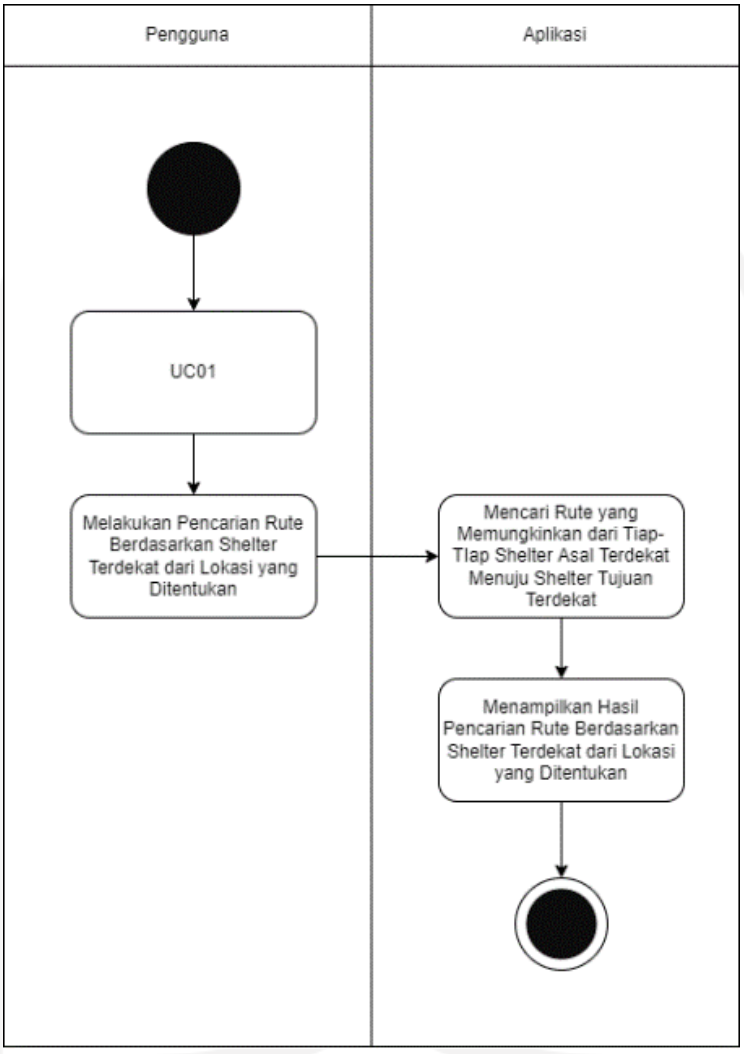


Gambar 3.10 *Activity Diagram* UC01

Melihat Rekomendasi Rute (UC02)

Proses melihat rekomendasi rute adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait alternatif rute berdasarkan *shelter* terdekat dari lokasi yang ditentukan sebelumnya pada UC01. Proses dimulai ketika pengguna melakukan aktivitas UC0. Setelah didapatkan *shelter* terdekat dari lokasi asal dan lokasi tujuan, pengguna melakukan pencarian rute berdasarkan *shelter* terdekat dari lokasi yang ditentukan. Selanjutnya, aplikasi mencari rute yang memungkinkan dari tiap-tiap *shelter* asal terdekat menuju *shelter* tujuan terdekat. Proses berakhir ketika hasil pencarian yang merupakan

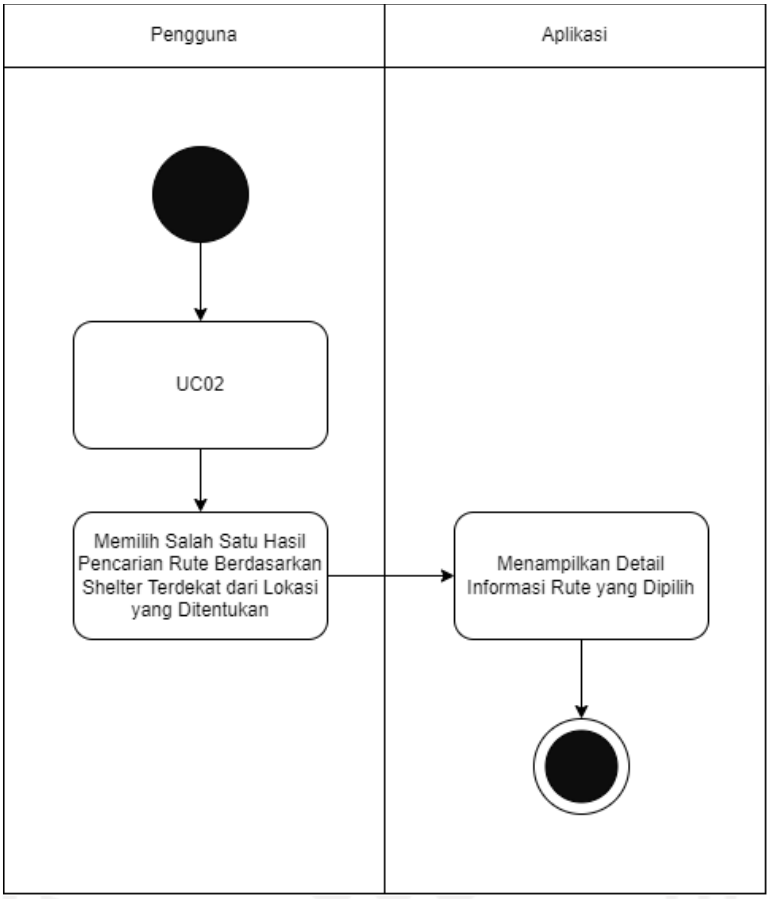
alternatif rute dari tiap-tiap *shelter* terdekat dari masing-masing lokasi ditampilkan oleh aplikasi kepada pengguna. *Activity Diagram* untuk proses ini dapat dilihat pada Gambar 3.11.



Gambar 3.11 *Activity Diagram* UC02

Melihat Detail Rekomendasi Rute (UC03)

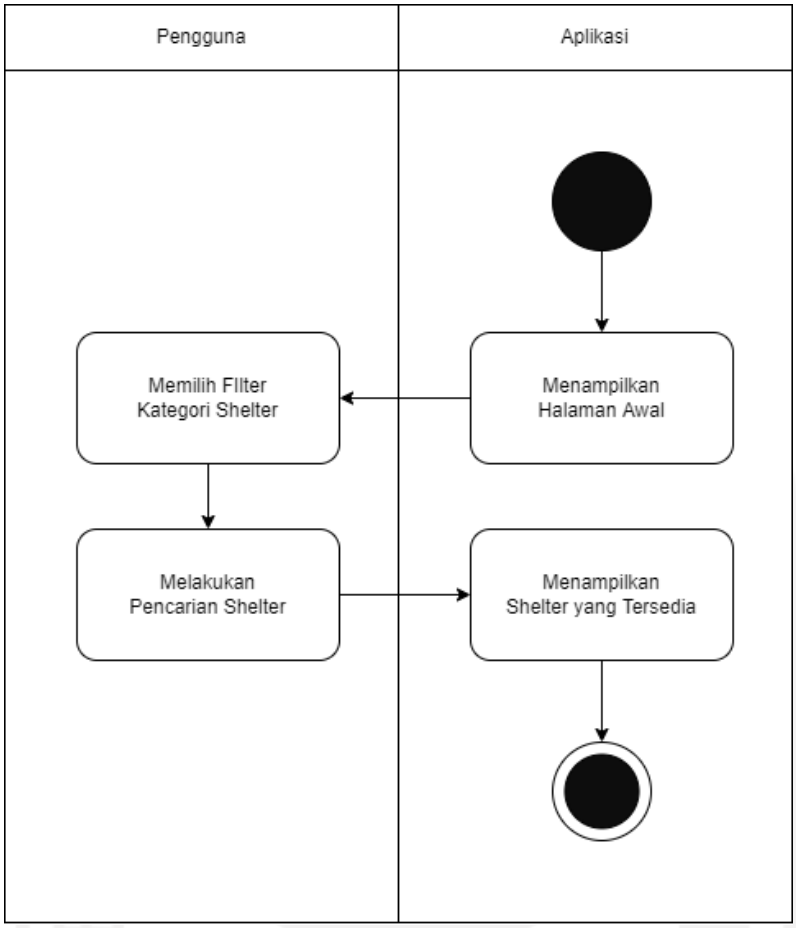
Proses melihat detail rekomendasi rute adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait alternatif rute yang dipilih sebelumnya. Alternatif rute yang dimaksud adalah alternatif rute yang dihasilkan dari pencarian pada UC02. Proses dimulai ketika pengguna melakukan aktivitas UC01. Setelah didapatkan *shelter* terdekat dari lokasi asal dan lokasi tujuan, pengguna melakukan aktivitas UC02. Kemudian, pengguna memilih salah satu alternatif rute yang ditampilkan. Proses berakhir ketika aplikasi menampilkan informasi terkait alternatif rute yang dipilih kepada pengguna. *Activity Diagram* dari proses ini dapat dilihat pada Gambar 3.12.



Gambar 3.12 *Activity Diagram* UC03

Melihat Informasi Shelter (UC04)

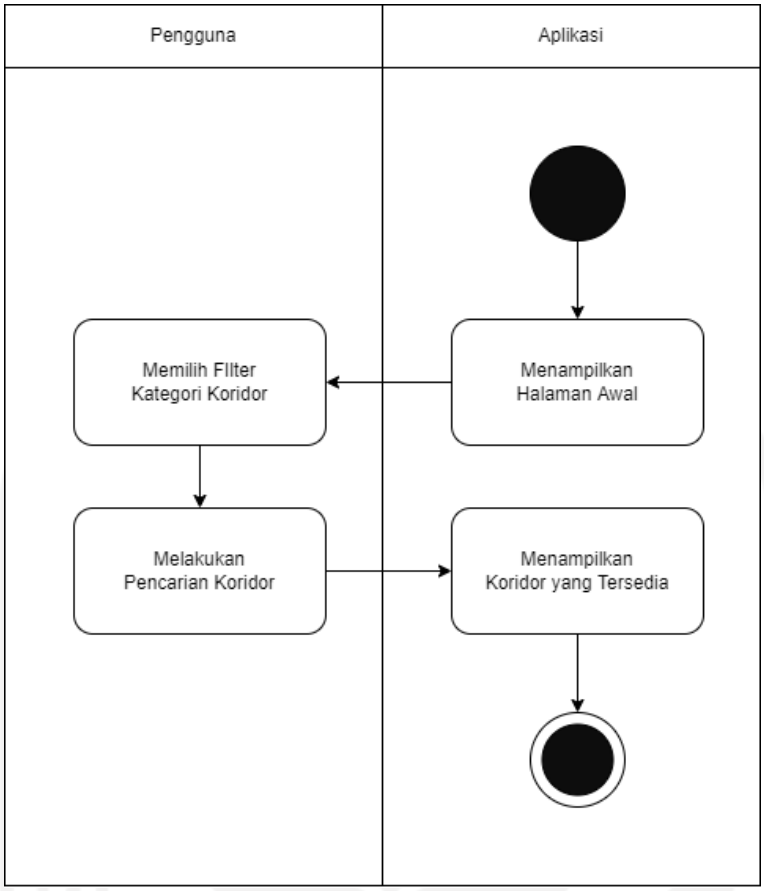
Proses melihat informasi *shelter* adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait *shelter*. Proses dimulai ketika aplikasi menampilkan halaman awal. Selanjutnya, pengguna memilih filter pencarian dengan kategori *shelter*. Kemudian, pencarian *shelter* dilakukan oleh pengguna. Proses berakhir ketika hasil pencarian *shelter* ditampilkan oleh aplikasi kepada pengguna. *Activity Diagram* dari proses ini dapat dilihat pada Gambar 3.13.



Gambar 3.13 *Activity Diagram* UC04

Melihat Informasi Koridor (UC05)

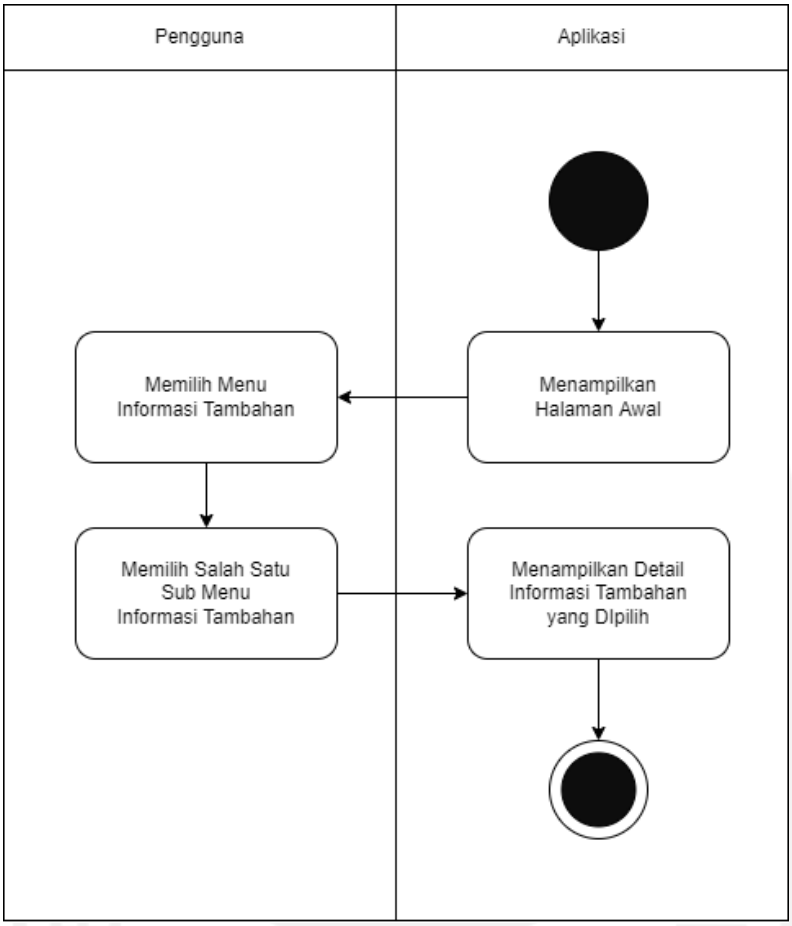
Proses melihat informasi koridor adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait koridor. Proses dimulai ketika aplikasi menampilkan halaman awal. Selanjutnya, pengguna memilih filter pencarian dengan kategori koridor. Kemudian, pencarian koridor dilakukan oleh pengguna. Proses berakhir ketika hasil pencarian koridor ditampilkan oleh aplikasi kepada pengguna. *Activity Diagram* dari proses ini dapat dilihat pada Gambar 3.14.



Gambar 3.14 *Activity Diagram* UC05

Melihat Informasi Tambahan (UC06)

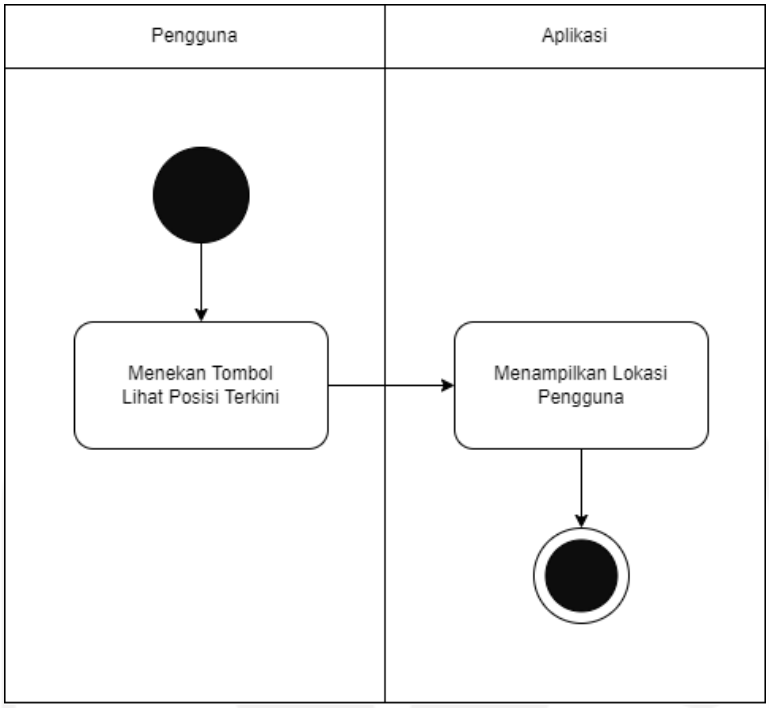
Proses melihat informasi tambahan adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait BRT Trans Jogja. Informasi yang dimaksud adalah informasi-informasi lain seperti misalnya Tarif, Cara Pembayaran, dan Jam Kerja. Proses dimulai ketika aplikasi menampilkan halaman awal. Selanjutnya, pengguna memilih menu informasi tambahan. Kemudian, pengguna memilih salah satu sub menu pada menu informasi tambahan. Proses berakhir ketika aplikasi menampilkan detail informasi terkait sub menu dipilih kepada pengguna. *Activity Diagram* dari proses ini dapat dilihat pada Gambar 3.15.



Gambar 3.15 *Activity Diagram* UC06

Melihat Lokasi Pengguna (UC07)

Proses melihat lokasi pengguna adalah proses di mana pengguna berinteraksi dengan aplikasi untuk melihat informasi terkait lokasi terkini pengguna. Proses dimulai ketika pengguna menekan tombol untuk melihat posisi terkini. Proses berakhir ketika aplikasi menampilkan lokasi terkini kepada pengguna. *Activity Diagram* dari proses ini dapat dilihat pada Gambar 3.16.



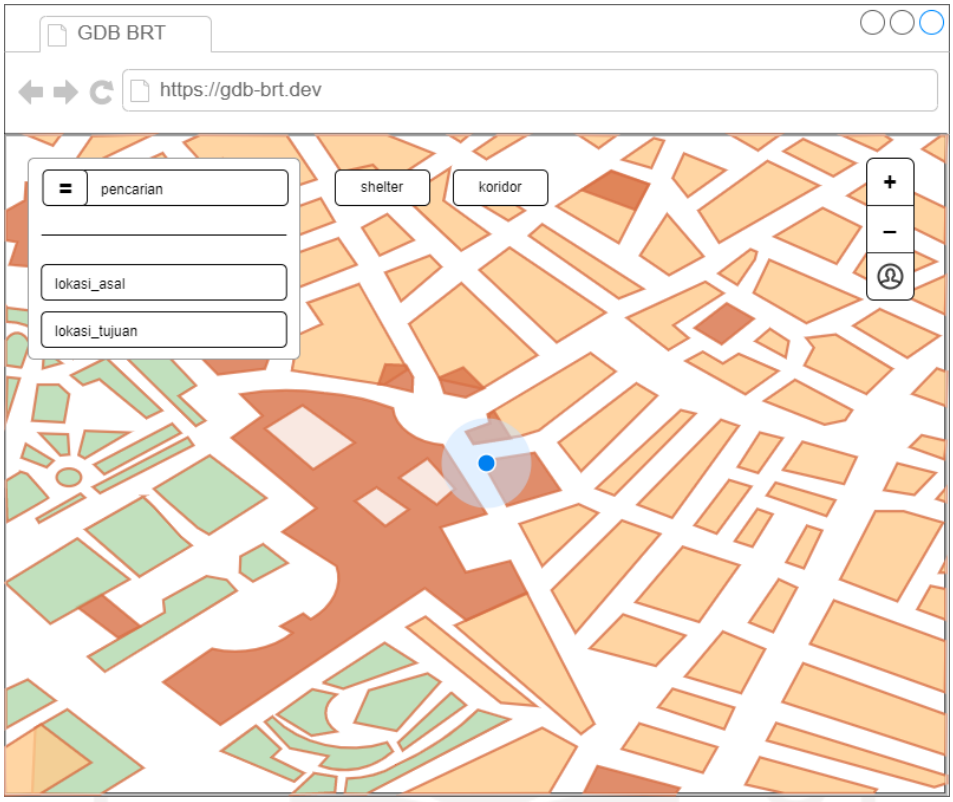
Gambar 3.16 *Activity Diagram* UC07

3.4.4 Perancangan Antarmuka

Tahapan selanjutnya adalah perancangan antarmuka di mana tampilan awal aplikasi dirancang berdasarkan analisis kebutuhan yang telah dilakukan. Pada aplikasi yang dikembangkan, rancangan antarmuka dibagi berdasarkan perilaku yang telah dirancang sebelumnya. Berikut rancangan antarmuka untuk masing-masing fitur yang ada pada aplikasi yang dikembangkan.

Perancangan Antarmuka Halaman Utama

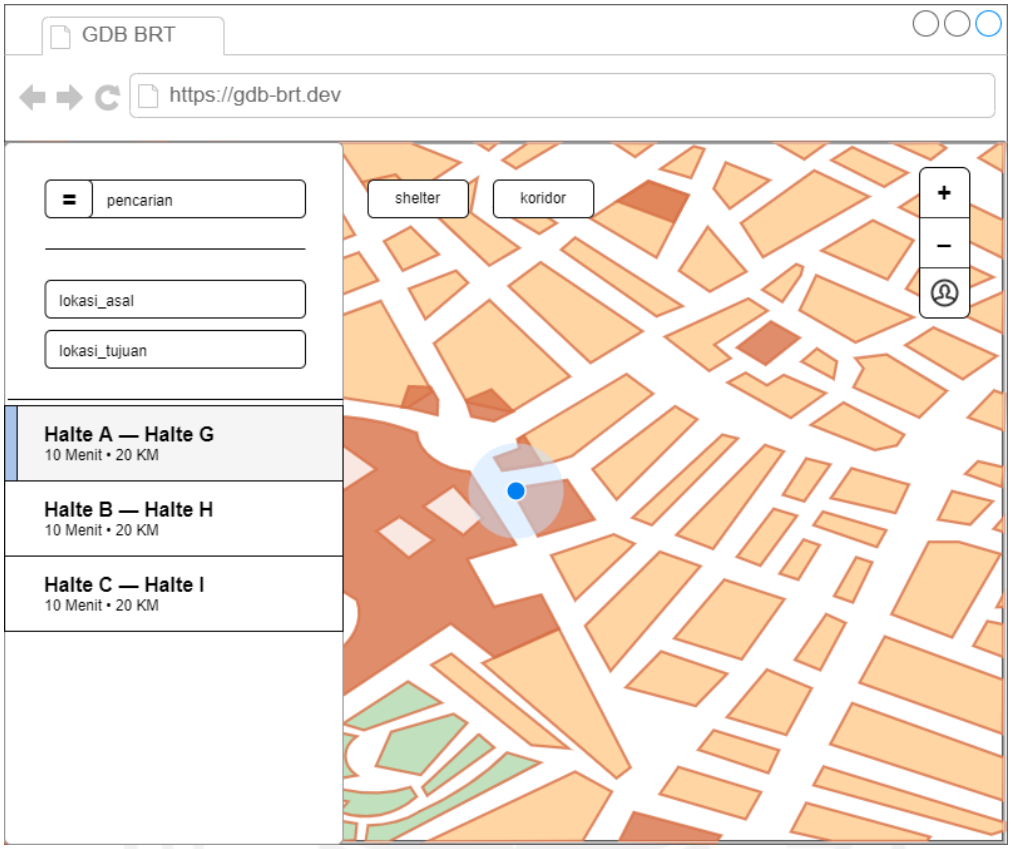
Halaman Utama adalah antarmuka yang menampilkan peta Provinsi Daerah Istimewa Yogyakarta. Pada antarmuka ini pengguna dapat melihat informasi lokasi terkini pengguna. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.17.



Gambar 3.17 Rancangan Antarmuka Halaman Utama

Perancangan Antarmuka Pencarian Rute

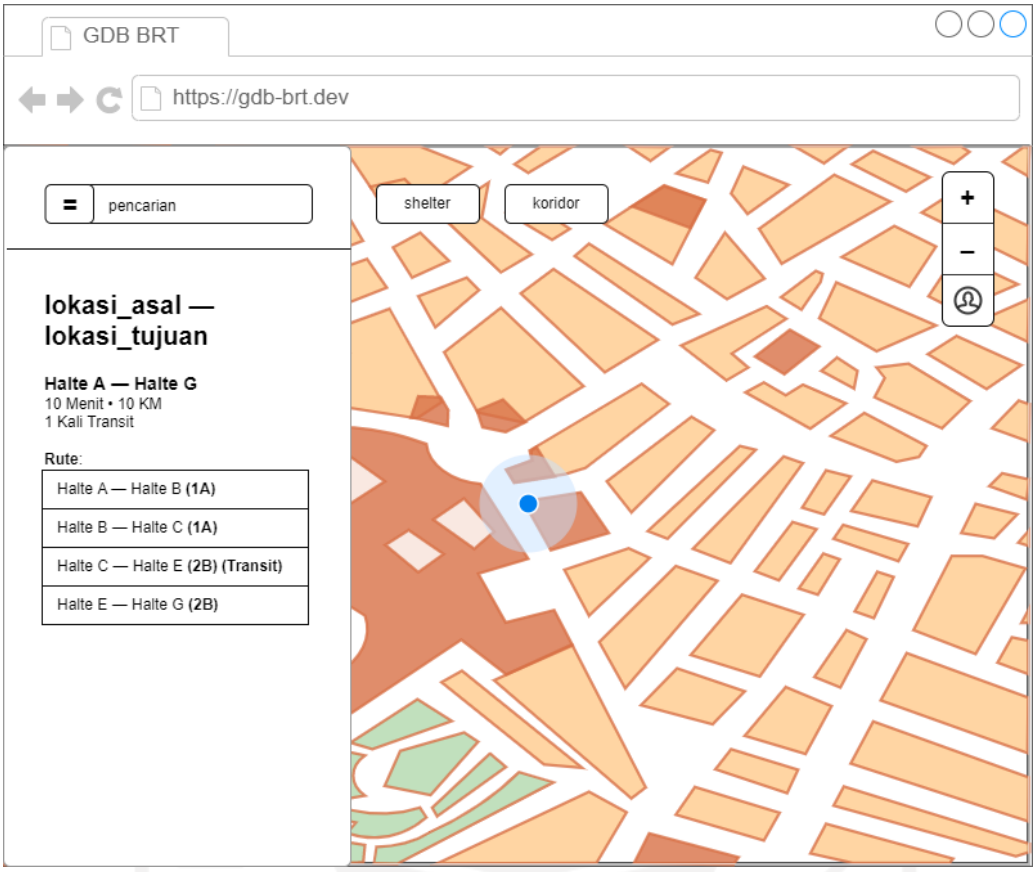
Antarmuka Pencarian Rute Rekomendasi digunakan ketika pengguna akan melakukan pencarian rute rekomendasi. Pada antarmuka ini pengguna dapat melihat informasi terkait *shelter* terdekat dan alternatif rute rekomendasi setelah memasukkan lokasi asal dan lokasi tujuan. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.18.



Gambar 3.18 Rancangan Antarmuka Pencarian Rute

Perancangan Antarmuka Detail Rute Rekomendasi

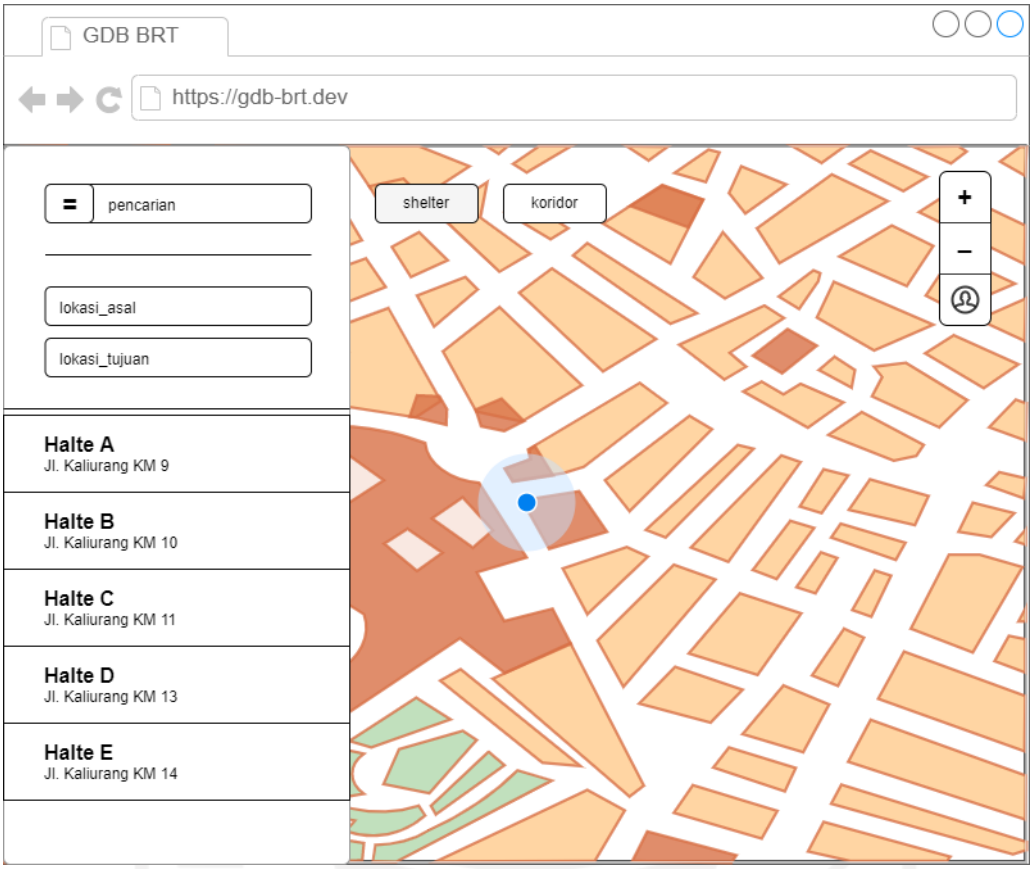
Antarmuka Detail Rute Rekomendasi digunakan ketika pengguna memilih salah satu alternatif rute rekomendasi. Pada antarmuka ini pengguna dapat melihat detail informasi rute yang dipilih. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.19.



Gambar 3.19 Rancangan Antarmuka Detail Rute Rekomendasi

Perancangan Antarmuka Informasi Shelter

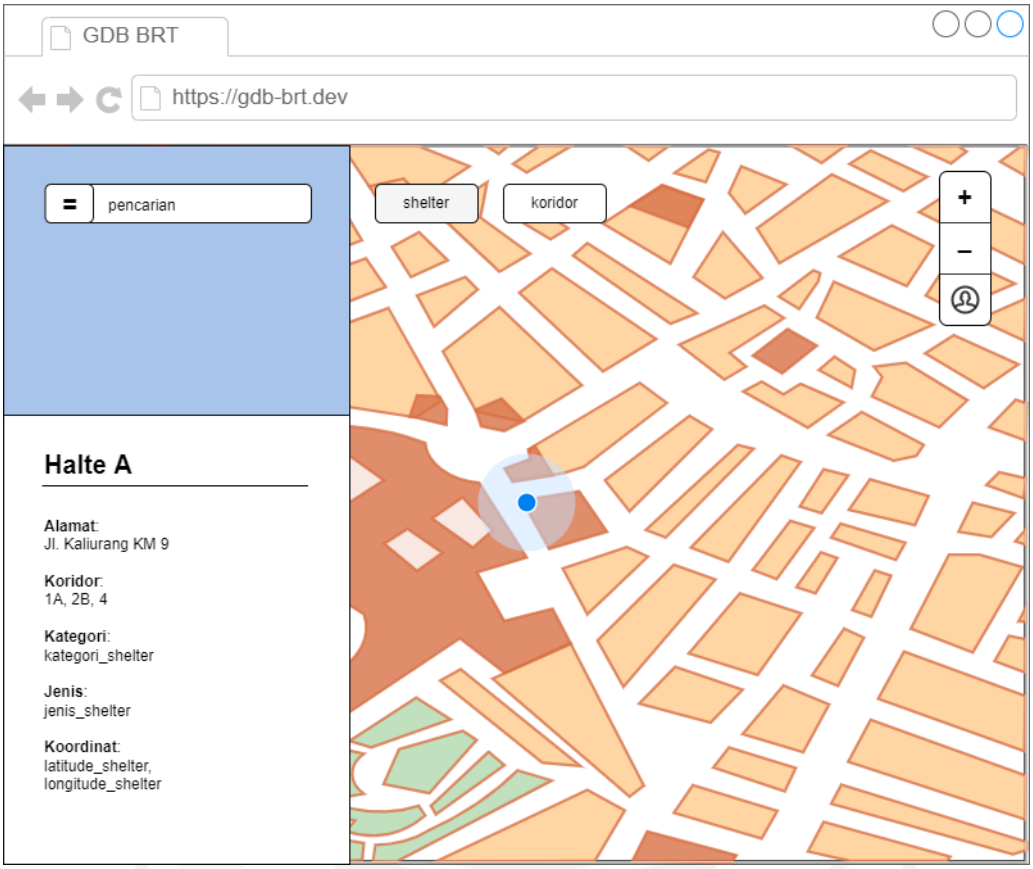
Antarmuka Informasi Shelter digunakan ketika pengguna melakukan pencarian berkategori *shelter*. Pada antarmuka ini pengguna dapat melihat informasi terkait semua *shelter* BRT Trans Jogja. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.20.



Gambar 3.20 Rancangan Antarmuka Informasi *Shelter*

Perancangan Antarmuka Detail Informasi Shelter

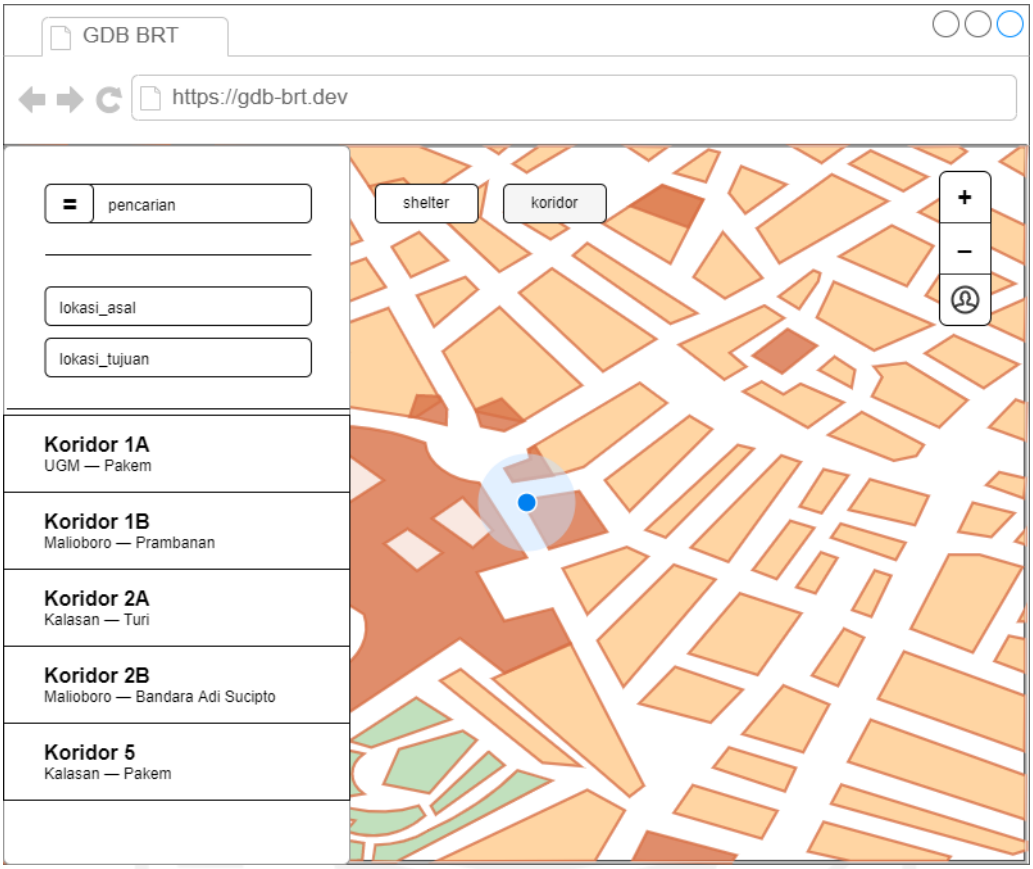
Antarmuka Detail Informasi Shelter digunakan ketika pengguna memilih suatu *shelter*. Pada antarmuka ini pengguna dapat melihat detail informasi *shelter* yang dipilih. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.21.



Gambar 3.21 Rancangan Antarmuka Detail Informasi *Shelter*

Perancangan Antarmuka Informasi Koridor

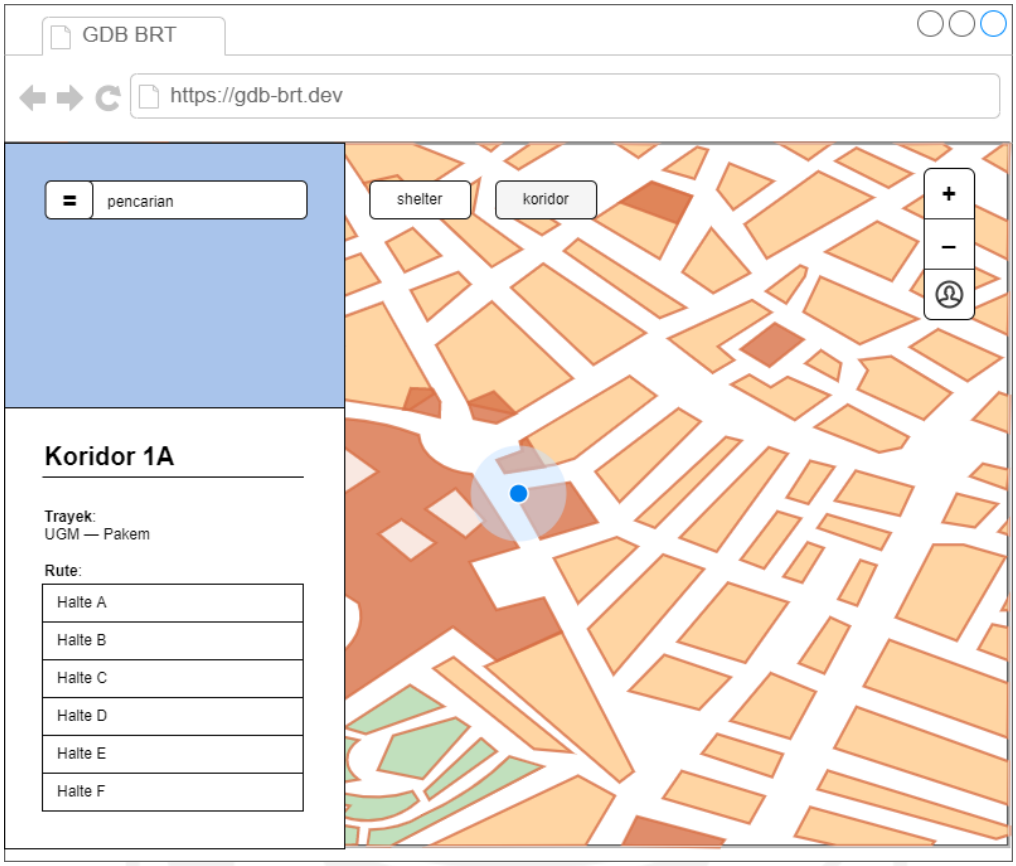
Antarmuka Informasi Koridor digunakan ketika pengguna melakukan pencarian berkategori koridor. Pada antarmuka ini pengguna dapat melihat informasi terkait semua koridor BRT Trans Jogja. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.22.



Gambar 3.22 Rancangan Antarmuka Informasi Koridor

Perancangan Antarmuka Detail Informasi Koridor

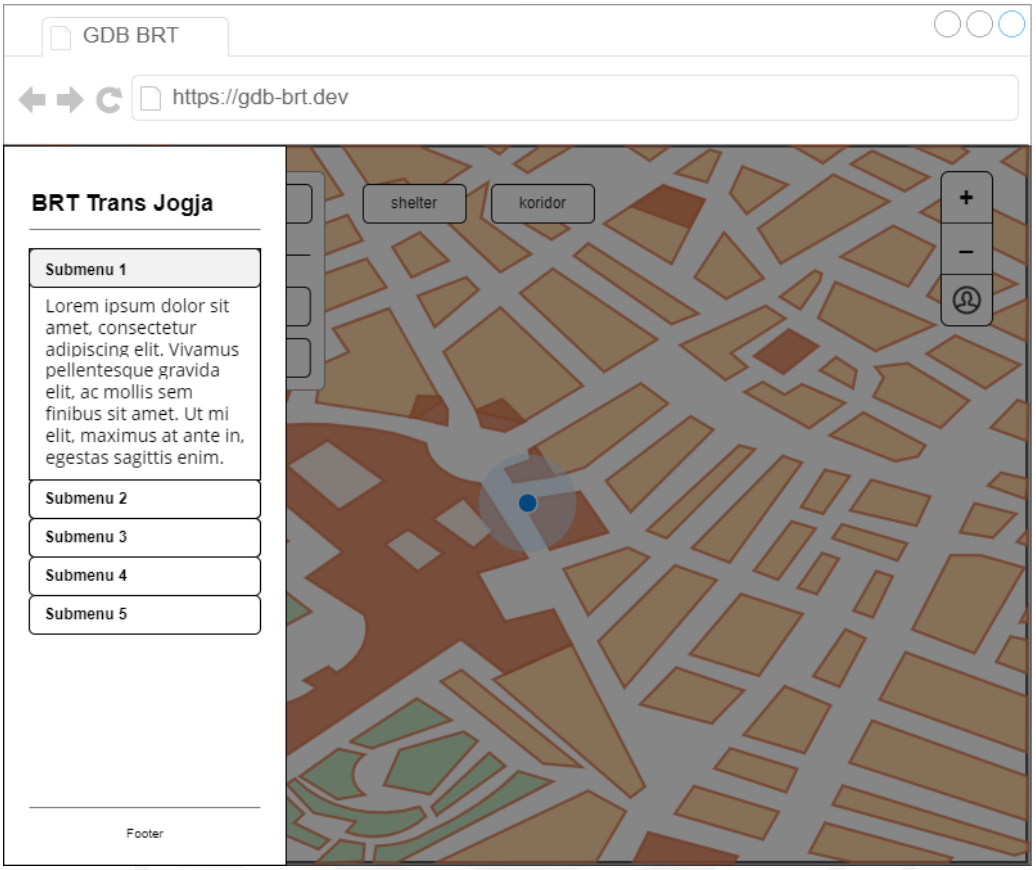
Antarmuka Detail Informasi Koridor digunakan ketika pengguna memilih suatu koridor. Pada antarmuka ini pengguna dapat melihat detail informasi koridor yang dipilih. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.23.



Gambar 3.23 Rancangan Antarmuka Detail Informasi Koridor

Perancangan Antarmuka Informasi Tambahan

Antarmuka Informasi Tambahan digunakan ketika pengguna memilih menu tambahan. Pada antarmuka ini pengguna dapat melihat informasi tambahan terkait BRT Trans Jogja seperti Tarif, Cara Pembayaran, dan Jam Kerja. Rancangan antarmuka terkait dapat dilihat pada Gambar 3.24.



Gambar 3.24 Rancangan Antarmuka Informasi Tambahan

3.4.5 Perancangan Permintaan dan Respons API

Guna memperjelas kebutuhan masukan dan keluaran API dari aplikasi yang akan dikembangkan, diperlukan rancangan yang dapat digunakan sebagai panduan ketika melakukan permintaan kepada API. Hal ini memungkinkan untuk format permintaan dan respons API yang lebih tertata, sehingga mempermudah penggunaan dari API yang akan dikembangkan. Pada aplikasi yang dikembangkan, terdapat 3 *endpoint* API, yaitu: *Shelters*, *Corridors*, dan *Route*. Berikut rancangan permintaan dan respons API untuk masing-masing *endpoint* dari aplikasi yang akan dikembangkan.

Endpoint /shelters

Endpoint shelters digunakan untuk memperoleh data terkait *shelter* BRT Trans Jogja. Pada *endpoint* ini, terdapat dua aksi yang dapat dilakukan, yaitu Mengambil Data Semua *Shelter*, Mengambil Data Suatu *Shelter*, dan Mengambil Data *Shelter* Terdekat. Berikut rancangan untuk masing-masing aksi:

a. Mengambil Data Semua *Shelter*

Aksi ini digunakan untuk memperoleh data semua *shelter* BRT Trans Jogja. Detail dari aksi ini dapat dilihat pada Tabel 3.6.

Tabel 3.6 Rancangan Permintaan API untuk Mengambil Data Semua *Shelter*

HTTP Method	GET
URL	/api/shelters
Headers	Accept: application/json
Parameters	-
Keterangan	-
Contoh Permintaan	{}

Contoh Respons	{
	status:'success',
	message: '',
	length: 0,
	data: [
	{
	uuid: '',
	kode: '',
	nama: '',
	jenis_halte: '',
	kategori_halte: '',
	sumber_dana: '',
	sumber_listrik: '',
	tahun: : '',
	kondisi: '',
	keterangan: '',
	latitude: '',
	longitude: ''
	},
	...
	]
	}

b. Mengambil Data Suatu *Shelter*

Aksi ini digunakan untuk memperoleh data suatu *shelter* BRT Trans Jogja berdasarkan ID yang ditentukan. Detail dari aksi ini dapat dilihat pada Tabel 3.7.

Tabel 3.7 Rancangan Permintaan API untuk Mengambil Data Suatu *Shelter*

HTTP Method	GET
URL	/api/shelters/:id
Headers	Accept: application/json

Parameters	{
	id: shelterUUID
	}

Keterangan	-

Contoh Permintaan	{
	id: '',
	}

Contoh Respons	{
	status:'success',
	message: '',
	data: {
	uuid: '',
	kode: '',
	nama: '',
	jenis_halte: '',
	kategori_halte: '',
	sumber_dana: '',
	sumber_listrik: '',
	tahun: : '',
	kondisi: '',
	keterangan: '',
	latitude: '',
	longitude: ''
	}
	}

c. Mengambil Data *Shelter* Terdekat

Aksi ini digunakan untuk memperoleh data *shelter* terdekat dari lokasi yang ditentukan menggunakan garis lintang dan garis bujur. Detail dari aksi ini dapat dilihat pada Tabel 3.8.

Tabel 3.8 Rancangan Permintaan API untuk Mengambil Data Shelter *Terdekat*

HTTP Method	GET
URL	/api/shelters/nearby/:lat/:long
Headers	Accept: application/json

Parameters	{
	lat: latitude,
	long: longitude,
	}

Keterangan	-

Contoh Permintaan	{
	lat: 0,
	long: 0
	}

Contoh Respons	{
	status:'success',
	message: '',
	length: 0,
	data: [
	{
	uuid: '',
	kode: '',
	nama: '',
	jenis_halte: '',
	kategori_halte: '',
	sumber_dana: '',
	sumber_listrik: '',
	tahun: : '',
	kondisi: '',
	keterangan: '',
	latitude: '',
	longitude: ''
	},
	...
	]
	}

Endpoint /corridors

Endpoint corridors digunakan untuk memperoleh data terkait koridor BRT Trans Jogja. Pada *endpoint* ini, terdapat dua aksi yang dapat dilakukan, yaitu Mengambil Data Semua Koridor, Mengambil Data Suatu Koridor, dan Mengambil Data Shelter Awal Suatu Koridor. Berikut rancangan untuk masing-masing aksi:

a. Mengambil Data Semua Koridor

Aksi ini digunakan untuk memperoleh data semua koridor BRT Trans Jogja. Detail dari aksi ini dapat dilihat pada Tabel 3.9.

Tabel 3.9 Rancangan Permintaan API untuk Mengambil Data Semua Koridor

HTTP Method	GET
URL	/api/corridors
Headers	Accept: application/json
Parameters	-
Keterangan	-
Contoh Permintaan	{}

Contoh Respons	{
	status: 'success',
	message: '',
	length: 0,
	data: [
	{
	uuid: '',
	nama: '',
	deskripsi: '',
	armada: '',
	},
	...
	]
	}

b. Mengambil Data Suatu Koridor

Aksi ini digunakan untuk memperoleh data suatu koridor BRT Trans Jogja berdasarkan ID yang ditentukan. Detail dari aksi ini dapat dilihat pada Tabel 3.10.

Tabel 3.10 Rancangan Permintaan API untuk Mengambil Data Suatu Koridor

HTTP Method	GET
URL	/api/corridors/:id
Headers	Accept: application/json

Parameters	{
	id: corridorUUID
	}

Keterangan	-

Contoh Permintaan	{
	id: ''
	}

Contoh Respons	{
	status: 'success',
	message: '',
	data: {
	uuid: '',
	nama: '',
	deskripsi: '',
	armada: '',
	}
	}

c. Mengambil Data *Shelter* Awal Suatu Koridor

Aksi ini digunakan untuk memperoleh data *shelter* awal suatu koridor BRT Trans Jogja berdasarkan ID yang ditentukan. Detail dari aksi ini dapat dilihat pada Tabel 3.11.

Tabel 3.11 Rancangan Permintaan API untuk Mengambil Data *Shelter* Awal Suatu Koridor

HTTP Method	GET
URL	/api/corridors/:id/start
Headers	Accept: application/json

Parameters	{
	id: corridorUUID
	}

Keterangan	-

Contoh Permintaan	{
	id: ''
	}

Contoh Respons	{
	status:'success',
	message: '',
	data: {
	uuid: '',
	kode: '',
	nama: '',
	jenis_halte: '',
	kategori_halte: '',
	sumber_dana: '',
	sumber_listrik: '',
	tahun: : '',
	kondisi: '',
	keterangan: '',
	latitude: '',
	longitude: ''
	}
	}

d. Mengambil Data Rute Suatu Koridor

Aksi ini digunakan untuk memperoleh data rute dari suatu koridor BRT Trans Jogja berdasarkan ID yang ditentukan. Detail dari aksi ini dapat dilihat pada Tabel 3.12.

Tabel 3.12 Rancangan Permintaan API untuk Mengambil Data Rute Suatu Koridor

HTTP Method	GET
URL	/api/corridors/:id/path
Headers	Accept: application/json

Parameters	{
	id: corridorUUID
	}

Keterangan	-

Contoh Permintaan	{
	id: ''
	}

Contoh Respons	{
	status:'success',
	message: '',
	data: {
	start: Object of Shelter,
	end: Object of Shelter,
	length: '',
	path: [
	{
	from: Object of Shelter,
	to: Object of Shelter,
	trayek: '',
	},
	...
	],
	}
	}

Endpoint /routes

Endpoint routes digunakan untuk memperoleh data terkait rute dalam BRT Trans Jogja. Pada *endpoint* ini, hanya terdapat satu aksi yaitu Rekomendasi Rute. Aksi ini digunakan untuk memperoleh data rute yang memungkinkan dari *shelter* awal menuju *shelter* tujuan yang ditentukan. Detail dari aksi ini dapat dilihat pada Tabel 3.13.

Tabel 3.13 Rancangan Permintaan API untuk Mengambil Data Rute

HTTP Method	GET
URL	/api/routes/:origin/:target
Headers	Accept: application/json

Parameters	{
	origin:[shelterUUID, ...],
	target: [shelterUUID, ...]
	}

Keterangan	-

Contoh Permintaan	{
	origin:['', ...],
	target:['', ...]
	}

Contoh Respons	{
	status:'success',
	message: '',
	data: [
	{
	start: Object of Shelter,
	end: Object of Shelter,
	length: '',
	path: [
	{
	from: Object of Shelter,
	to: Object of Shelter,
	trayek: '',
	},
	...
	],
	}
	...
	]
	}

3.5 Rencana Pengujian

Untuk memastikan aplikasi yang dikembangkan dapat berjalan dengan baik dan benar serta sesuai dengan kebutuhan yang dirancang sebelumnya, tahap yang perlu dilewati adalah tahap pengujian. Pada penelitian ini dilakukan pengujian *Black Box* untuk mengetahui apakah aplikasi yang dikembangkan sesuai dengan rancangan. Pengujian ini dilakukan oleh penulis dengan skenario tertentu. Pengujian *Black Box* dilakukan pada Aplikasi dan API. Rancangan untuk masing-masing pengujian dapat dilihat pada Tabel 3.14 dan Tabel 3.15.

Tabel 3.14 Rancangan Pengujian *Black Box* untuk *Frontend*

No.	Fungsionalitas	Aktivitas	Hasil yang Diharapkan
2	Melihat Rekomendasi Rute	Menentukan lokasi asal dan lokasi tujuan	Aplikasi dapat menampilkan alternatif rute rekomendasi berdasarkan lokasi yang ditentukan pengguna
3	Melihat Detail Rekomendasi Rute	Memilih alternatif rekomendasi rute	Aplikasi dapat menampilkan detail rute rekomendasi pilihan pengguna
4	Melihat Informasi *Shelter*	Memilih kategori *shelter* dan salah satu shelter	Aplikasi dapat menampilkan semua *shelter* dan detail *shelter* pilihan pengguna.
5	Melihat Informasi Koridor	Memilih kategori koridor dan salah satu koridor	Aplikasi dapat menampilkan semua koridor dan detail koridor pilihan pengguna berupa rute pada peta
6	Melihat Informasi Tambahan	Memilih informasi pada menu	Aplikasi dapat menampilkan informasi tambahan sesuai dengan pilihan pengguna
7	Melihat Lokasi Pengguna	Menekan tombol lihat posisi terkini	Aplikasi dapat menampilkan lokasi terkini pengguna pada peta

Tabel 3.15 Rancangan Pengujian *Black Box* untuk API

No.	Fungsionalitas	Aktivitas	Hasil yang Diharapkan
1	Melihat *Shelter* Terdekat	Menentukan lokasi asal dan lokasi tujuan	Aplikasi dapat menampilkan *shelter* terdekat dari lokasi yang ditentukan pengguna
2	Melihat Rekomendasi Rute	Menentukan lokasi asal dan lokasi tujuan	Aplikasi dapat menampilkan alternatif rute rekomendasi berdasarkan lokasi yang ditentukan pengguna
3	Melihat Detail Rekomendasi Rute	Memilih alternatif rekomendasi rute	Aplikasi dapat menampilkan detail rute rekomendasi pilihan pengguna
4	Melihat Informasi *Shelter*	Memilih kategori *shelter* dan salah satu *shelter*	Aplikasi dapat menampilkan semua *shelter* dan detail *shelter* pilihan pengguna.
5	Melihat Informasi Koridor	Memilih kategori koridor dan salah satu koridor	Aplikasi dapat menampilkan semua koridor dan detail koridor pilihan pengguna berupa rute pada peta

BAB IV
HASIL DAN PEMBAHASAN

4.1 Implementasi

4.1.1 Implementasi Basis Data Graf

Pada implementasi basis data graf, model data BRT Trans Jogja yang sudah dirancang sebelumnya menggunakan *Labeled Property Graph* dibuat pada Neo4J bervarian AuraDB dengan *query* berbahasa Cypher. Model tersebut dapat dilihat pada Gambar. Berikut *query* yang digunakan untuk masing-masing *node* dan *relationship*.

Implementasi *Node* SHELTER

Node berlabel SHELTER mempunyai *property* nama, kategori_halte, jenis_halte, sumber_listrik, sumber_dana, tahun, kondisi, keterangan, *latitude*, dan *longitude*. Untuk membuat *node* tersebut digunakan *query* seperti yang terlihat pada Gambar 4.1.

```
CREATE
    (shelter:SHELTER {
        uuid: apoc.create.uuid(),
        nama: 'STP Hyundai Jl Laksda Adisutjipto',
        kategori_halte: 'STP',
        jenis_halte: 'Halte Portable',
        sumber_listrik: 'Tenaga Surya',
        sumber_dana: 'APBD DIY',
        tahun: '2007',
        kondisi: 'BAIK',
        keterangan: null,
        latitude: '-7.78353695665',
        longitude: '110.428338929'
    })
RETURN
    shelter
```

Gambar 4.1 Implementasi *Cypher Query Language* untuk *Node* Berlabel SHELTER

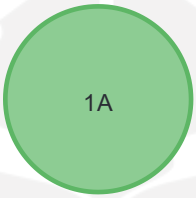
Gambar 4.2 Hasil Implementasi *Node* Berlabel SHELTER

Implementasi *Node* CORRIDOR

Node berlabel CORRIDOR mempunyai *property* nama, deskripsi, dan armada. Untuk membuat *node* tersebut digunakan *query* seperti yang terlihat pada Gambar 4.3.

```
CREATE
    (corridor:CORRIDOR {
        uuid: apoc.create.uuid(),
        nama: '1A',
        deskripsi: 'Terminal Prambanan - Malioboro',
        armada: '10'
    })
RETURN
    corridor
```

Gambar 4.3 Implementasi *Cypher Query Language* untuk *Node* Berlabel CORRIDOR



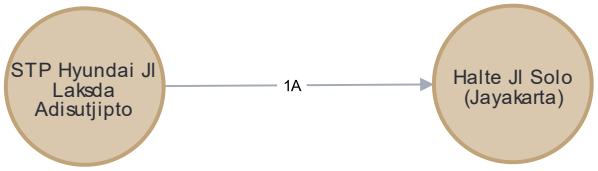
Gambar 4.4 Hasil Implementasi *Node* Berlabel CORRIDOR

Implementasi *Relationship* MENGHUBUNGKAN

Relationship berlabel MENGHUBUNGKAN mempunyai *property* trayek. *Relationship* ini menghubungkan *node* SHELTER dengan *node* SHELTER. Untuk membuat *relationship* tersebut digunakan *query* seperti yang terlihat pada Gambar 4.5.

```
MATCH
    (origin:SHELTER),
    (target:SHELTER)
WHERE
    origin.uuid = [UUID]
AND
    target.uuid = [UUID]
CREATE
    (origin)-[rel:MENGHUBUNGKAN { trayek: '1A' }]->(target)
RETURN
    origin, target, rel
```

Gambar 4.5 Implementasi *Cypher Query Language* untuk *Relationship* Berlabel MENGHUBUNGKAN



Gambar 4.6 Hasil Implementasi *Relationship* Berlabel MENGHUBUNGKAN

Implementasi *Relationship* MULAI_DARI

Relationship berlabel MULAI_DARI tidak mempunyai *property*. *Relationship* ini menghubungkan *node* CORRIDOR dengan *node* SHELTER. Untuk membuat *relationship* tersebut digunakan *query* seperti yang terlihat pada Gambar 4.7.

```
MATCH
    (origin:CORRIDOR),
    (target:SHELTER)
WHERE
    origin.uuid = [UUID]
AND
    target.uuid = [UUID]
CREATE
    (origin)-[rel:MULAI_DARI]->(target)
RETURN
    origin, target, rel
```

Gambar 4.7 Implementasi *Cypher Query Language* untuk *Relationship* Berlabel MULAI_DARI

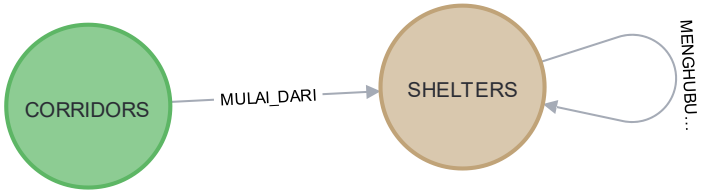
Gambar 4.8 Hasil Implementasi *Relationship* Berlabel MULAI_DARI

Pemeriksaan Skema

Menggunakan *query-query* sebelumnya, data BRT Trans Jogja yang telah didapatkan sebelumnya dimasukkan ke dalam basis data. Setelah semua data berhasil dimasukkan, perlu dipastikan skema dari basis data sesuai dengan model yang telah dirancang sebelumnya. Untuk melihat skema dari basis data graf yang ada dapat digunakan *query* di bawah ini. Hasil tersebut dapat dilihat pada Gambar 4.9.

```
CALL db.schema.visualization()
```

Gambar 4.9 *Cypher Query Language* untuk Memeriksa Skema Basis Data Graf



Gambar 4.10 Skema Basis Data Graf

4.1.2 Implementasi API

API diimplementasikan menggunakan Node.js dan Express.js. Selain itu API menggunakan neo4j-javascript-driver untuk menghubungkan API dengan Neo4J. Berikut implementasi API pada aplikasi yang dikembangkan.

Implementasi *Endpoint /shelters*

a. Melihat Semua *Shelter*

Dalam pengimplementasian *endpoint /shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.11.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/shelter.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get('/', validateInput([]), async (req, res) =>
    _routeResponse(req, res, service.getAll)
)

module.exports = router
```

Gambar 4.11 Implementasi Rute *Endpoint* API untuk Melihat Semua Shelter

Pada Gambar 4.11 dapat dilihat *service* yang digunakan untuk *endpoint /shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang

nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.12.

```
const shelterModel = require('../models/shelter.model')
const _responseHelper = require('../utils/helper/response.helper')

const getAll = async (params, query, path) =>
    shelterModel
        .getAll(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess(
                'OK',
                path,
                query.offset
                    ? { ...query, ...result }
                    : { length: result.length, data: result }
            )
        )
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getAll
}
```

Gambar 4.12 Implementasi *Service* pada API untuk Melihat Semua Shelter

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian untuk semua *node* berlabel :SHELTER berdasarkan permintaan yang ada, yaitu melihat semua *shelter* yang ada. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.13.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const { _mapCorridor, _mapShelter } =
require('../utils/helper/model.helper')

const getAll = async (params, query) => {
    let CQLquery = `
        MATCH (shelters:SHELTER)
        RETURN shelters
        ORDER BY shelters.jenis_halte, shelters.kode`
    let CQLparams = {}

    if (query.offset && query.size) {
        CQLquery =
            'MATCH (shelters:SHELTER) ' +
            'RETURN shelters ' +
            'SKIP $offset ' +
            'LIMIT $size'
```

```
        CQLparams = {
            offset: neo4j.int(query.offset * query.size),
            size: neo4j.int(query.size)
        }
    }

    return _neo4jHelper(CQLquery, CQLparams, _mapShelter)
}

module.exports = {
    getAll
}
```

Gambar 4.13 Implementasi Model pada API untuk Melihat Semua *Shelter*

b. Melihat Suatu *Shelter*

Dalam pengimplementasian *endpoint /shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.14.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/shelter.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get('/:id', validateInput(['id']), async (req, res) =>
    _routeResponse(req, res, service.getById)

module.exports = router
```

Gambar 4.14 Implementasi Rute *Endpoint* API untuk Melihat Suatu *Shelter*

Pada Gambar 4.14 dapat dilihat *service* yang digunakan untuk *endpoint /shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.15.

```
const shelterModel = require('../models/shelter.model')
const _responseHelper = require('../utils/helper/response.helper')

const getById = async (params, query, path) =>
    shelterModel
```

```
        .getById(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess('OK', path, result)
        )
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getById
}
```

Gambar 4.15 Implementasi *Service* pada API untuk Melihat Suatu *Shelter*

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian untuk *node* berlabel :SHELTER yang memiliki *property* UUID tertentu. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.16.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const { _mapCorridor, _mapShelter } =
require('../utils/helper/model.helper')

const getById = async (params, query) =>
    _neo4jHelper(
        `MATCH
            (shelters:SHELTER)
        WHERE
            shelters.uuid = $id
        RETURN
            shelters`,
        {
            id: params.id
        },
        _mapShelter
    )

module.exports = {
    getById
}
```

Gambar 4.16 Implementasi Model pada API untuk Melihat Suatu *Shelter*

c. Melihat *Shelter* Terdekat

Dalam pengimplementasian *endpoint* /*shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.17.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/shelter.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get(
    '/nearby/:lat/:long',
    validateInput(['lat', 'long']),
    async (req, res) => _routeResponse(req, res, service.getNearby)
)

module.exports = router
```

Gambar 4.17 Implementasi Rute *Endpoint* API untuk Melihat *Shelter* Terdekat

Pada Gambar 4.17 dapat dilihat *service* yang digunakan untuk *endpoint /shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.18.

```
const shelterModel = require('../models/shelter.model')
const _responseHelper = require('../utils/helper/response.helper')

const getNearby = async (params, query, path) =>
    shelterModel
        .getNearby(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess(
                'OK',
                path,
                query.offset
                    ? { ...query, ...result }
                    : { length: result.length, data: result }
            )
        )
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getNearby
}
```

Gambar 4.18 Implementasi *Service* pada API untuk Melihat *Shelter* Terdekat

Untuk mencari shelter terdekat, digunakan rumus *harvesine* untuk menghitung jarak antara dua titik koordinat *langitude-longitude*. Dalam Neo4j terdapat fungsi *Cypher point.distance()* untuk menghitung jarak antara dua titik koordinat menggunakan rumus tersebut. Fungsi tersebut diimplementasikan dalam pengembangan model untuk *endpoint*

ini. Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Sehingga, setiap *shelter* yang memilik jarak kurang dari 1000 meter dari lokasi yang ditentukan akan diproses dan dikembalikan ke peminta. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.19.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const { _mapCorridor, _mapShelter } =
require('../utils/helper/model.helper')

const getNearby = async (params, query) =>
    _neo4jHelper(
        `MATCH (shelter:SHELTER)
        WITH
            shelter,
            point.distance(
                point({
                    longitude: toFloat($long),
                    latitude: toFloat($lat),
                    srid:4326
                }),
                point({
                    longitude: toFloat(shelter.longitude),
                    latitude: toFLoat(shelter.latitude)
                })
            ) as DISTANCE WHERE DISTANCE < 1000
        RETURN apoc.create.vNode(
                [
                    'SHELTER'
                ],
                {
                    uuid: shelter.uuid,
                    nama: shelter.nama,
                    kategori_halte: shelter.kategori_halte,
                    jenis_halte: shelter.jenis_halte,
                    sumber_listrik: shelter.sumber_listrik,
                    sumber_dana: shelter.sumber_dana,
                    tahun: shelter.tahun,
                    kondisi: shelter.kondisi,
                    keterangan: shelter.keterangan,
                    latitude: shelter.latitude,
                    longitude: shelter.longitude,
                    jarak: DISTANCE
                }
            ) AS shelters
        ORDER BY DISTANCE
        LIMIT 5`,
        {
            lat: params.lat,
            long: params.long
        },
        _mapShelter
```

```
    )

module.exports = {
    getNearby
}
```

Gambar 4.19 Implementasi Model pada API untuk Melihat *Shelter* Terdekat

Implementasi *Endpoint /corridors*

a. Melihat Semua Koridor

Dalam pengimplementasian *endpoint /shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.20.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/corridor.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get('/', validateInput([]), async (req, res) =>
    _routeResponse(req, res, service.getAll)
)

module.exports = router
```

Gambar 4.20 Implementasi Rute *Endpoint* API untuk Melihat Semua Koridor

Pada Gambar 4.20 dapat dilihat *service* yang digunakan untuk *endpoint /shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.21.

```
const corridorModel = require('../models/corridor.model')
const _responseHelper = require('../utils/helper/response.helper')

const getAll = async (params, query, path) =>
    corridorModel
        .getAll(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess(
                'OK',
                path,
```

```
                query.offset
                    ? { ...query, ...result }
                    : { length: result.length, data: result }
            )
        )
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getAll
}
```

Gambar 4.21 Implementasi *Service* pada API untuk Melihat Semua Koridor

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian untuk semua *node* berlabel :CORRIDOR berdasarkan permintaan yang ada, yaitu melihat semua koridor yang ada. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.22.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const {
    _mapCorridor,
    _mapPath,
    _mapShelter
} = require('../utils/helper/model.helper')

const getAll = async (params, query) => {
    let CQLquery =
        'MATCH (corridors:CORRIDOR) RETURN corridors ORDER BY
corridors.nama'
    let CQLparams = {}

    if (query.offset && query.size) {
        CQLquery =
            'MATCH (corridors:CORRIDOR) ' +
            'RETURN corridors ' +
            'SKIP $offset ' +
            'LIMIT $size'

        CQLparams = {
            offset: neo4j.int(query.offset * query.size),
            size: neo4j.int(query.size)
        }
    }

    return _neo4jHelper(CQLquery, CQLparams, _mapCorridor)
}

module.exports = {
    getAll
}
```

Gambar 4.22 Implementasi Model pada API untuk Melihat Semua Koridor

b. Melihat Suatu Koridor

Dalam pengimplementasian *endpoint /shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.23.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/corridor.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get('/:id', validateInput(['id']), async (req, res) =>
    _routeResponse(req, res, service.getById)
)

module.exports = router
```

Gambar 4.23 Implementasi Rute *Endpoint* API untuk Melihat Suatu Koridor

Pada Gambar 4.23 dapat dilihat *service* yang digunakan untuk *endpoint /shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.24.

```
const corridorModel = require('../models/corridor.model')
const _responseHelper = require('../utils/helper/response.helper')

const getById = async (params, query, path) =>
    corridorModel
        .getById(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess('OK', path, result)
        )
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getById
}
```

Gambar 4.24 Implementasi *Service* pada API untuk Melihat Suatu Koridor

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian untuk *node* berlabel :CORRIDOR yang memiliki *property* UUID tertentu. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.25.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const {
    _mapCorridor,
    _mapPath,
    _mapShelter
} = require('../utils/helper/model.helper')

const getById = async (params, query) =>
    _neo4jHelper(
        `MATCH
            (corridors:CORRIDOR)
        WHERE
            corridors.uuid = $id
        RETURN
            corridors`,
        {
            id: params.id
        },
        _mapCorridor
    )

module.exports = {
    getById
}
```

Gambar 4.25 Implementasi Model pada API untuk Melihat Suatu Koridor

c. Melihat *Shelter* Awal Suatu Koridor

Dalam pengimplementasian *endpoint /shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.26.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/corridor.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse
```

```
router.get('/:id/start', validateInput(['id']), async (req, res) =>
    _routeResponse(req, res, service.getStart)
)

module.exports = router
```

Gambar 4.26 Implementasi Rute *Endpoint* API untuk Melihat *Shelter* Awal Suatu Koridor

Pada Gambar 4.26 dapat dilihat *service* yang digunakan untuk *endpoint* /*shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.27.

```
const corridorModel = require('../models/corridor.model')
const _responseHelper = require('../utils/helper/response.helper')

const getStart = async (params, query, path) =>
    corridorModel
        .getStart(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess('OK', path, result)
        )
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getStart
}
```

Gambar 4.27 Implementasi *Service* pada API untuk Melihat Shelter Awal Suatu Koridor

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian *node* berlabel :CORRIDOR yang memiliki *property* UUID tertentu. Selain itu, *node* :CORRIDOR tersebut harus juga memiliki hubungan :MULAI_DARI dengan *node* :SHELTER. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.28.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const {
    _mapCorridor,
    _mapPath,
    _mapShelter
} = require('../utils/helper/model.helper')
```

```
const getStart = async (params, query) =>
    _neo4jHelper(
        `MATCH
            (origin:CORRIDOR)-[:MULAI_DARI]->(shelters:SHELTER)
        WHERE
            origin.uuid = $id
        RETURN
            shelters`,
        {
            id: params.id
        },
        _mapShelter
    )

module.exports = {
    getStart
}
```

Gambar 4.28 Implementasi Model pada API untuk Melihat *Shelter* Awal Suatu Koridor

d. Melihat Rute Suatu Koridor

Dalam pengimplementasian *endpoint* /*shelters*, pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.29.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const validate = require('../middlewares/token.middleware')
const service = require('../services/corridor.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get('/:id/path', validateInput(['id']), async (req, res) =>
    _routeResponse(req, res, service.getPath)
)

module.exports = router
```

Gambar 4.29 Implementasi Rute *Endpoint* API untuk Melihat Rute Suatu Koridor

Pada Gambar 4.29 dapat dilihat *service* yang digunakan untuk *endpoint* /*shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.30.

```
const corridorModel = require('../models/corridor.model')
const _responseHelper = require('../utils/helper/response.helper')

const getPath = async (params, query, path) =>
    corridorModel
        .getPath(params, query)
        .then(async (result) => {
            return _responseHelper.responseSuccess('OK', path, result)
        })
        .catch((error) => _responseHelper.responseError(path, error))

module.exports = {
    getPath
}
```

Gambar 4.30 Implementasi *Service* pada API untuk Melihat Rute Suatu Koridor

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian *path* dengan hubungan berlabel :MENGHUBUNGKAN dan memiliki *property* trayek tertentu yang menghubungkan *node-node* berlabel :SHELTER. Pencarian diawali dengan mencari *shelter* awal dari koridor yang diminta. Setelah itu, nama dari koridor dan UUID dari *shelter* awal digunakan pada pencarian *path* sebelumnya. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.31.

```
const neo4j = require('neo4j-driver')
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const {
    _mapCorridor,
    _mapPath,
    _mapShelter
} = require('../utils/helper/model.helper')

const getPath = async (params, query) =>
    _neo4jHelper(
        `MATCH
            (origin:CORRIDOR {uuid:$id})-[:MULAI_DARI]->(target:SHELTER)
        OPTIONAL MATCH
            path = (target)-[:MENGHUBUNGKAN* {trayek:origin.nama}]-
>(next:SHELTER)
        RETURN
            path
        ORDER BY length(path) DESC
        LIMIT 1`,
        {
            id: params.id
        },
        _mapPath
    )

module.exports = {
```

```
    getPath
}
```

Gambar 4.31 Implementasi Model pada API untuk Melihat Rute Suatu Koridor

Implementasi *Endpoint /routes*

Dalam pengimplementasian *endpoint /shelters,* pertama penulis menentukan rute yang nantinya digunakan sebagai jalur komunikasi antara api dengan aplikasi. Implementasi ini dilakukan pada berkas *route* yang di dalamnya berisi *endpoint shelter* dan *service* apa yang digunakan untuk melayani jalur tersebut. Implementasi ini dapat dilihat pada Gambar 4.32.

```
const router = require('express').Router()
const validateInput =
require('../middlewares/params.middleware').validateInput
const service = require('../services/route.service')
const _routeResponse =
require('../utils/helper/route.helper')._routeResponse

router.get(
    '/:origin/:target',
    validateInput(['origin', 'target']),
    async (req, res) => _routeResponse(req, res, service.getRoutes)
)

module.exports = router
```

Gambar 4.32 Implementasi Rute *Endpoint* API untuk Melihat Rute

Pada Gambar 4.32 dapat dilihat *service* yang digunakan untuk *endpoint /shelters*. *Service* tersebut berisi logika seperti model apa yang digunakan dan pemrosesan data yang nantinya akan dikembalikan kepada peminta. Implementasi dari *service* ini dapat dilihat pada Gambar 4.33.

```
const routeModel = require('../models/path.model')
const _responseHelper = require('../utils/helper/response.helper')

const getRoutes = async (params, query, path) => {
    return routeModel
        .getBetween(params, query)
        .then(async (result) =>
            _responseHelper.responseSuccess('OK', path, result)
        )
        .catch((error) => _responseHelper.responseError(path, error))
}

module.exports = {
    getRoutes
}
```

Gambar 4.33 Implementasi *Service* pada API untuk Melihat Rute

Model yang digunakan pada *endpoint* ini berisi *query* Cypher yang melakukan permintaan pada basis data graf. Data yang didapatkan dari basis data graf nantinya akan diproses menjadi objek yang dapat digunakan oleh peminta. Pada model ini, dilakukan pencarian semua kemungkinan *path* dengan hubungan berlabel :MENGHUBUNGKAN yang diawali dan diakhiri *node* :SHELTER dengan *property* UUID tertentu. Implementasi dari model ini beserta *query* yang digunakan dapat dilihat pada Gambar 4.34.

```
const _neo4jHelper = require('../utils/helper/neo4j.helper')
const { _mapPath } = require('../utils/helper/model.helper')

const getBetweenOne = async (params, query) => {
    console.log(params)
    return _neo4jHelper(
        `
        MATCH
            path = shortestPath((origin:SHELTER)-[:MENGHUBUNGKAN*..]-
>(target:SHELTER))
        WHERE
            origin.uuid = $origin
        AND
            target.uuid = $target
        RETURN path
        `,
        {
            origin: params.origin,
            target: params.target
        },
        _mapPath
    )
}

const getBetween = (params, query) =>
    Promise.all(
        params.origin.map((a, indexA) =>
            Promise.resolve()
                .then(() =>
                    Promise.all(
                        params.target.map((b, indexB) =>
                            getBetweenOne({
                                origin: a,
                                target: b
                            }).then((data) => data)
                        )
                    )
                )
                .then((data) => data)
        )
    ).then((data) => data)

module.exports = {
    getBetween
}
```

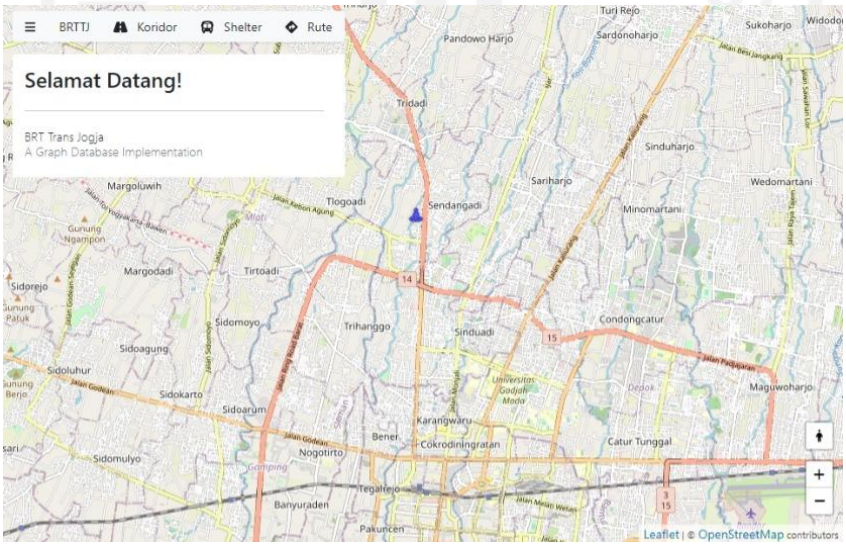
Gambar 4.34 Implementasi Model pada API untuk Melihat Rute

4.1.3 Implementasi Aplikasi

Aplikasi diimplementasikan menggunakan React.js. Selain, itu digunakan Redux dengan tambahan React-Redux untuk membantu manajemen *state* pada aplikasi React.js. Untuk keperluan peta, digunakan Leaflet.js dengan tambahan React-Leaflet dan Leaflet-Route-Machine beserta OpenStreetMap sebagai penyedia data peta dan server *route machine*. Selain itu, digunakan Nominatim untuk keperluan *geocoding*. Berikut implementasi dari aplikasi yang dikembangkan.

Implementasi Halaman Utama

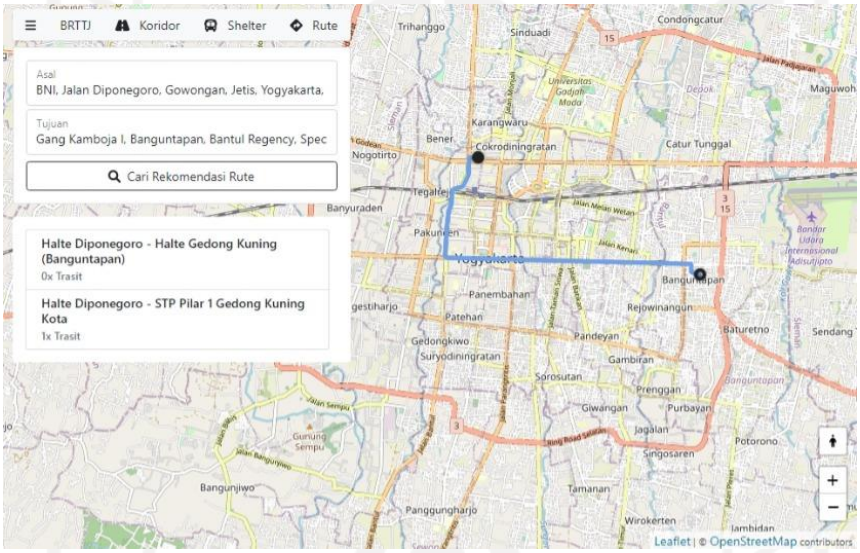
Halaman utama adalah antarmuka yang pertama kali muncul ketika pengguna menggunakan aplikasi yang dikembangkan. Pada halaman ini ditampilkan peta yang berpusat pada lokasi terkini pengguna. Ditampilkan pula form untuk menentukan lokasi asal dan lokasi tujuan. Lokasi tersebut dapat ditentukan dengan memasukkan alamat pada *form* yang ada atau dengan menekan lokasi pada peta. Selain itu, pada halaman ini pengguna dapat melihat lokasi terkini pada peta, menentukan lokasi untuk melihat rute rekomendasi, dan melihat informasi tambahan terkait BRT Trans Jogja. Tampilan dari halaman ini dapat dilihat pada Gambar 4.35.



Gambar 4.35 Hasil Implementasi Halaman Utama

Implementasi Fitur Melihat Rekomendasi Rute

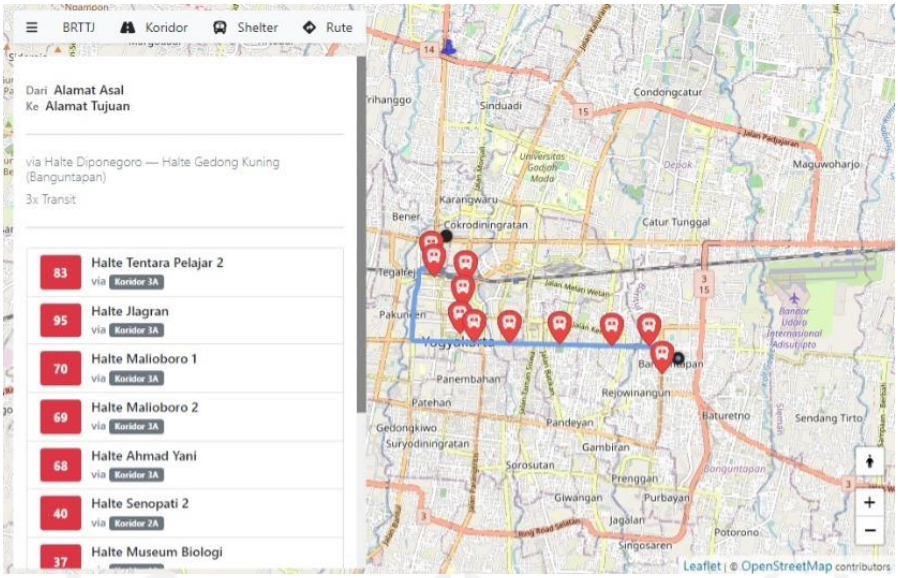
Untuk melihat rekomendasi rute, pengguna perlu mengatur lokasi asal dan lokasi tujuan untuk melihat *shelter-shelter* terdekat dari masing-masing lokasi. Pencarian *shelter* terdekat dilakukan dengan mencari *shelter-shelter* yang termasuk dalam radius 1 Km dari lokasi-lokasi yang telah ditentukan sebelumnya. Setelah shelter-shelter terdekat ditemukan, aplikasi akan mencari alternatif rute terpendek yang memungkingkan dari masing-masing shelter asal menuju shelter tujuan. Kemudian alternatif-alternatif tersebut ditampilkan pada peta dan *sidebar* dengan urutan yang didasarkan pada perkiraan jarak dan waktu tempuh. Pengguna dapat memilih salah satu dari alternatif yang ada. Tampilan dari fitur ini dapat dilihat pada Gambar 4.36.



Gambar 4.36 Hasil Implementasi Fitur Melihat Rute Rekomendasi

Implementasi Fitur Melihat Detail Rekomendasi Rute

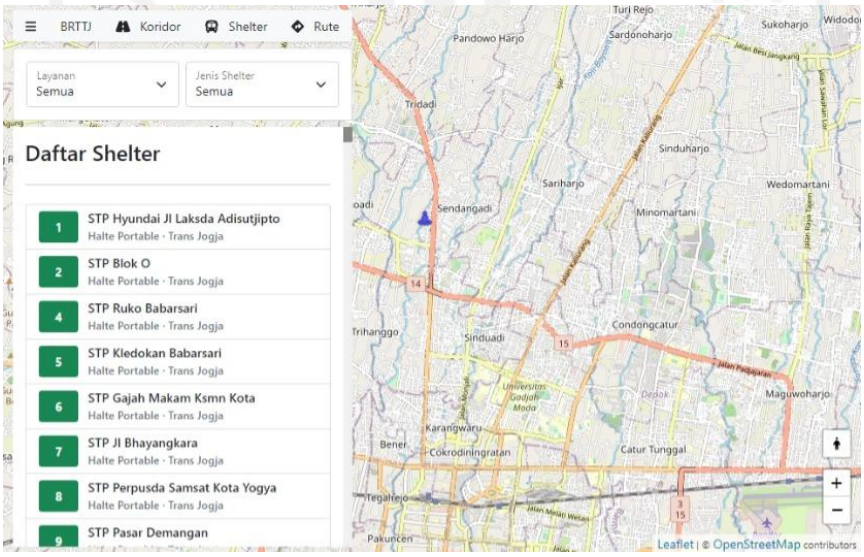
Setelah pengguna memilih salah satu alternatif rute rekomendasi, informasi rute dan *shelter* yang dilewati akan ditampilkan pada peta dan *sidebar*. Tampilan dari fitur ini dapat dilihat pada Gambar 4.37.



Gambar 4.37 Hasil Implementasi Fitur Melihat Detail Rute Rekomendasi

Implementasi Fitur Melihat Informasi *Shelter*

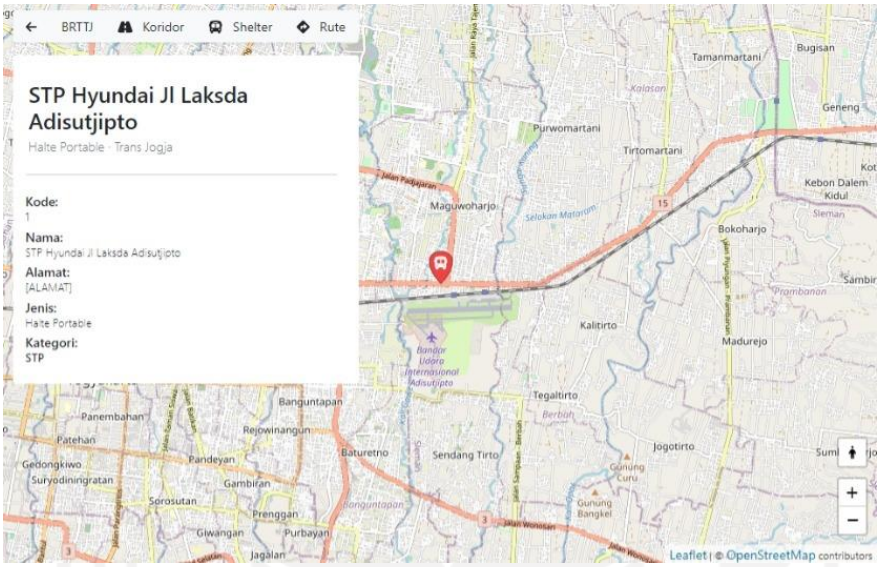
Pengguna dapat melihat daftar *shelter* BRT Trans Jogja dengan menekan tombol kategori yang terdapat pada bagian atas halaman. Daftar tersebut nantinya akan ditampilkan pada peta dan *sidebar* yang dapat pengguna pilih. Tampilan dari fitur ini dapat dilihat pada Gambar 4.38.



Gambar 4.38 Hasil Implementasi Fitur Melihat Informasi *Shelter*

Implementasi Fitur Melihat Detail Informasi *Shelter*

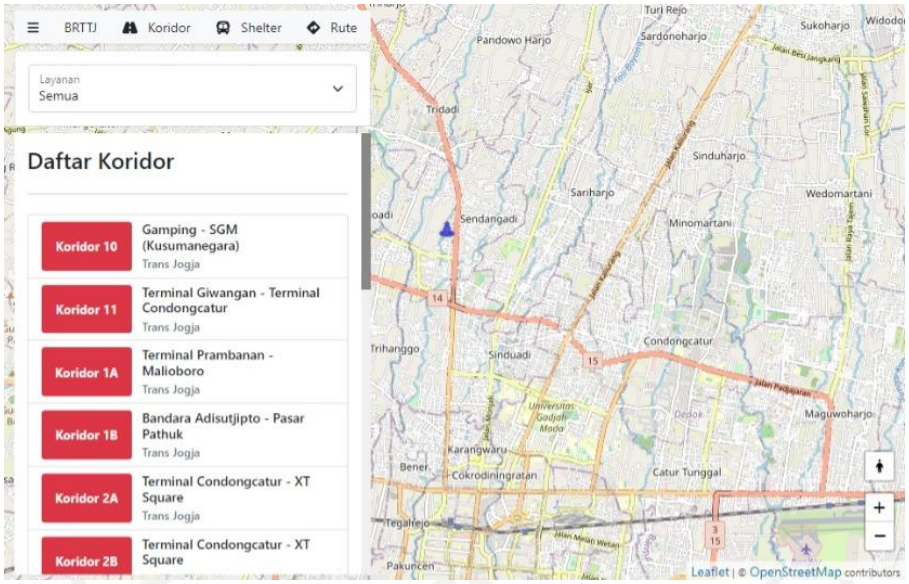
Setelah pengguna memilih suatu *shelter*, baik dari *sidebar* maupun dari peta, informasi mengenai detail *shelter* tersebut akan ditampilkan pada *sidebar*. Tampilan dari fitur ini dapat dilihat pada Gambar 4.39.



Gambar 4.39 Hasil Implementasi Fitur Melihat Detail Informasi *Shelter*

Implementasi Fitur Melihat Informasi Koridor

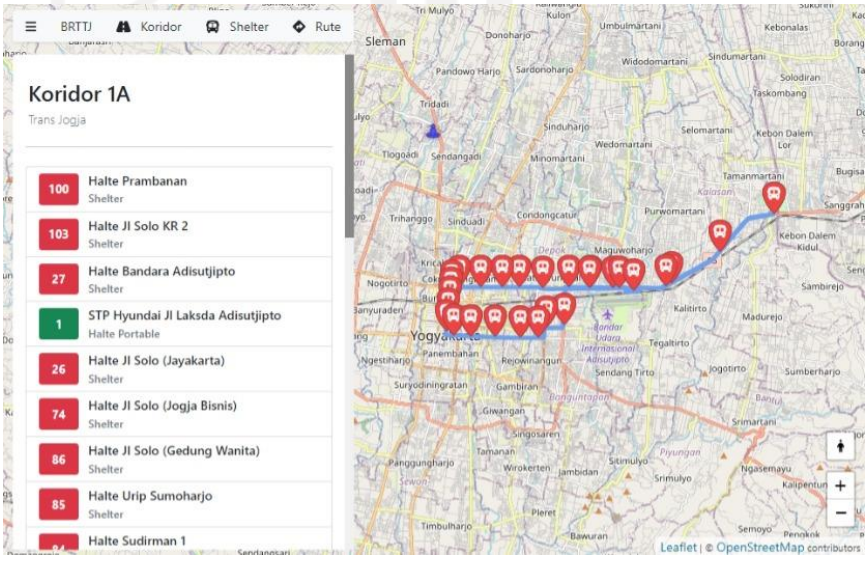
Pengguna dapat melihat daftar koridor BRT Trans Jogja dengan menekan tombol kategori yang terdapat pada bagian atas halaman. Daftar tersebut nantinya akan ditampilkan pada *sidebar* yang dapat pengguna pilih. Tampilan dari fitur ini dapat dilihat pada Gambar 4.40.



Gambar 4.40 Hasil Implementasi Fitur Melihat Informasi Koridor

Implementasi Fitur Melihat Detail Informasi Koridor

Setelah pengguna memilih suatu koridor, informasi mengenai detail koridor tersebut akan ditampilkan pada *sidebar*. Informasi seperti rute dan *shelter* yang termasuk di dalamnya juga akan ditampilkan pada peta. Tampilan dari fitur ini dapat dilihat pada Gambar 4.41.

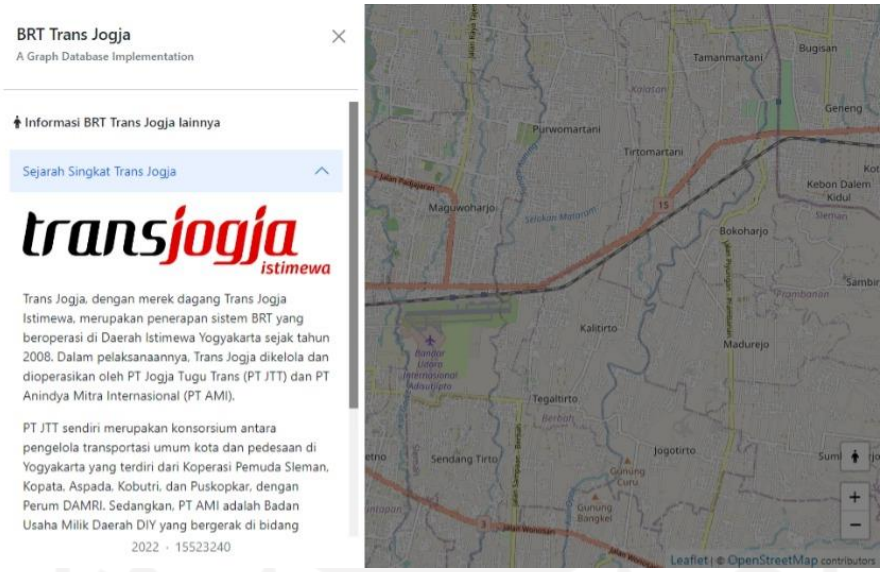


Gambar 4.41 Hasil Implementasi Fitur Melihat Detail Informasi Koridor

Implementasi Fitur Melihat Informasi Tambahan

Pada *sidebar* terdapat tombol yang dapat pengguna gunakan untuk melihat informasi lain terkait BRT Trans Jogja. Setelah tombol ditekan, akan muncul *sidebar* lain berisi informasi

yang terbagi menjadi beberapa submenu. Tampilan dari fitur ini dapat dilihat pada Gambar 4.42.



Gambar 4.42 Hasil Implementasi Fitur Melihat Informasi Tambahan

Implementasi Fitur Melihat Lokasi Pengguna

Pada peta tersedia tombol yang dapat pengguna gunakan untuk melihat lokasi terkini. Setelah tombol ditekan, peta akan terpusat pada lokasi terkini pengguna. Tampilan dari fitur ini dapat dilihat pada Gambar 4.43.

Gambar 4.43 Hasil Implementasi Fitur Melihat Lokasi Pengguna

4.2 Pengujian

Untuk memastikan aplikasi dapat berjalan dengan baik dan benar, dilakukan pengujian seperti yang telah direncanakan sebelumnya. Pengujian pertama yang dilakukan adalah pengujian fungsionalitas untuk *backend* menggunakan pengujian *Black Box*. Pengujian dilakukan oleh penulis pada API dengan mengirimkan permintaan yang berisi parameter tertentu. Hasil dari pengujian ini dapat dilihat pada Tabel 4.1 dan Tabel 4.2.

Tabel 4.1 Hasil Pengujian *Black Box* untuk *Frondend*

No.	Fungsionalitas	Aktivitas	Hasil yang Diharapkan	Hasil Pengujian
2	Melihat Rekomendasi Rute	Menentukan lokasi asal dan lokasi tujuan	Aplikasi dapat menampilkan rute rekomendasi berdasarkan lokasi yang ditentukan pengguna	Berhasil
3	Melihat Detail Rekomendasi Rute	Memilih alternatif rekomendasi rute	Aplikasi dapat menampilkan detail rekomendasi rute pilihan pengguna	Berhasil
4	Melihat Informasi *Shelter*	Memilih kategori *shelter* dan salah satu *shelter*	Aplikasi dapat menampilkan informasi semua *shelter* dan *shelter* pilihan pengguna	Berhasil
5	Melihat Informasi Koridor	Memilih kategori koridor dan salah satu koridor	Aplikasi dapat menampilkan informasi semua koridor dan koridor pilihan pengguna	Berhasil
6	Melihat Informasi Tambahan	Memilih informasi pada menu	Aplikasi dapat menampilkan informasi tambahan sesuai dengan pilihan pengguna	Berhasil
7	Melihat Lokasi Pengguna	Menekan tombol lihat posisi terkini	Aplikasi dapat menampilkan lokasi pengguna pada peta	Berhasil

Tabel 4.2 Hasil Pengujian *Black Box* untuk API

No.	Fungsionalitas	Aktivitas	Hasil yang Diharapkan	Hasil Pengujian
1	Melihat *Shelter* Terdekat	Menentukan lokasi asal dan lokasi tujuan	Aplikasi dapat menampilkan *shelter* terdekat dari lokasi yang ditentukan pengguna	Berhasil
2	Melihat Rekomendasi Rute	Menentukan lokasi asal dan lokasi tujuan	Aplikasi dapat menampilkan rute rekomendasi berdasarkan lokasi yang ditentukan pengguna	Berhasil
3	Melihat Detail Rekomendasi Rute	Memilih alternatif rekomendasi rute	Aplikasi dapat menampilkan detail rekomendasi rute pilihan pengguna	Berhasil
4	Melihat Informasi *Shelter*	Memilih kategori *shelter* dan salah satu *shelter*	Aplikasi dapat menampilkan informasi semua *shelter* dan *shelter* pilihan pengguna	Berhasil
5	Melihat Informasi Koridor	Memilih kategori koridor dan salah satu koridor	Aplikasi dapat menampilkan informasi semua koridor dan koridor pilihan pengguna	Berhasil

Berdasarkan hasil pengujian seperti yang tertera pada Tabel 4.1 dan Tabel 4.2, dapat disimpulkan bahwa aplikasi berjalan dengan baik dan sesuai dengan kebutuhan yang telah dirancang sebelumnya.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah melalui tahapan penelitian, perancangan, implementasi, dan pengujian pada penelitian tugas akhir ini, telah berhasil dikembangkan aplikasi rekomendasi rute BRT Trans Jogja yang memanfaatkan basis data graf. Aplikasi mampu menyediakan informasi seputar BRT Trans Jogja. Informasi yang dapat disediakan oleh aplikasi antara lain seperti:

a. Telah berhasil memanfaatkan basis data graf untuk menentukan rute transportasi umum BRT Trans Jogja.
b. Telah berhasil memodelkan rute transportasi umum BRT Trans Jogja menggunakan graf, *labeled property graph*, dan basis data graf.
c. Telah berhasil mengimplementasikan model graf rute transportasi umum BRT Trans Jogja.
d. Telah berhasil mengembangkan aplikasi rute transportasi umum BRT Trans Jogja yang memanfaatkan basis data graf.

Namun, ditemukan pula beberapa kelemahan pada aplikasi yang dikembangkan. Kelemahan tersebut antara lain sebagai berikut:

a. Pada perangkat yang tidak dilengkapi GPS, penentuan otomasi lokasi pengguna ditentukan berdasarkan IP dari ISP (penyedia internet) yang digunakan, sehingga akurasi lokasi agak kurang.
b. Belum adanya estimasi waktu dan jarak yang akurat.

5.2 Saran

Pada penelitian Pemanfaatan Basis Data Graf pada Aplikasi Rute Transportasi Umum dengan studi kasus Trans Jogja, aplikasi yang dikembangkan masih memiliki kekurangan dan dapat dikembangkan lebih jauh. Beberapa saran yang dapat diberikan untuk pengembangan ke depannya antara lain sebagai berikut:

a. Melakukan wawancara dengan Dinas Perhubungan DIY dan/atau pihak terkait untuk mendapatkan data perihal BRT Trans Jogja seperti misalnya kebutuhan, *shelter*, koridor, armada, dan jam operasi agar aplikasi serta model basis data yang dikembangkan dapat lebih relevan.

b. Adanya pengujian yang dilakukan oleh Dinas Perhubungan DIY, calon pengguna BRT Trans Jogja, dan pihak terkait agar aplikasi yang dibangun sesuai dengan kebutuhan yang ada.
c. Adanya penggalian data yang lebih mendalam terkait *shelter* BRT Trans Jogja layanan Teman Bus.
d. Penambahan fitur pelacakan armada secara langsung untuk setiap koridor.
e. Penambahan fitur pemeriksaan ketersediaan armada untuk setiap koridor.
f. Penambahan fitur perkiraan keberangkatan dan kedatangan armada untuk setiap *shelter* dan koridor.
g. Penambahan fitur pengubahan rute untuk setiap koridor.
h. Penambahan fitur manajemen *shelter* dan koridor.
i. Menggunakan penyedia layanan *route machine* yang lebih akurat untuk membantu penentuan perkiraan jarak dan waktu tempuh serta *mapping* rute pada peta.
j. Menyediakan antarmuka untuk pengaksesan aplikasi melalui perangkat *mobile*.
k. Aplikasi dapat dikembangkan lebih lanjut menjadi *Progressive Web Application* (PWA).
l. Penambahan *error handling* pada aplikasi khususnya ketika berkomunikasi dengan API, baik API yang berurusan dengan basis data graf maupun API pihak ketiga.
m. Dasar pengambilan keputusan untuk rekomendasi rute berdasarkan lokasi asal dan tujuan pengguna dapat dikembangkan lebih lanjut sehingga rekomendasi yang dihasilkan nantinya lebih relevan, sesuai, dan berguna bagi calon pengguna BRT Trans Jogja.

DAFTAR PUSTAKA

Badan Pengembangan dan Pembinaan Bahasa Kemendikbud RI. (2016a). *Angkutan – KBBI Daring*. https://kbbi.kemdikbud.go.id/entri/angkutan

Badan Pengembangan dan Pembinaan Bahasa Kemendikbud RI. (2016b). *Transportasi – KBBI Daring*. https://kbbi.kemdikbud.go.id/entri/transportasi

Bechberger, D., & Perryman, J. (2020). *Graph Databases in Action: Examples in Gremlin*. Manning.

Dinas Perhubungan Daerah Istimewa Yogyakarta. (2021). *SI-LLAJ Dishub D.I. Yogyakarta*. https://sillaj-dishub.jogjaprov.go.id/halte

Dinas Perhubungan Daerah Istimewa Yogyakarta. (2022). *Trans Jogja*. https://dishub.jogjaprov.go.id/layanan/trans-jogja

Holton, D., & Clark, J. (1995). *A First Look at Graph Theory*. World Scientific Publishing Co. Pte. Ltd.

Institute for Transportation & Development Policy. (2022a). *The BRT Standard – Institute for Transportation and Development Policy*. https://www.itdp.org/2016/06/21/the-brt-standard/

Institute for Transportation & Development Policy. (2022b). *What is BRT? – Institute for Transportation and Development Policy*. https://www.itdp.org/library/standards-and-guides/the-bus-rapid-transit-standard/what-is-brt/

Kementerian Perhubungan Republik Indonesia. (2020a). *Teman Bus*. https://temanbus.com/

Kementerian Perhubungan Republik Indonesia. (2020b). *Yogyakarta – Teman Bus*. https://temanbus.com/yogyakarta/

Levin, O. (2018). *Discrete Mathematics: An Open Introduction*.

Neo4j Inc. (2022a). *Fully Managed Graph Database Service | Neo4j AuraDB*. https://neo4j.com/cloud/platform/aura-graph-database/

Neo4j Inc. (2022b). *Neo4j AuraDB overview - Neo4j Aura*. https://neo4j.com/docs/aura/auradb/

Owusu, C., Lan, Y., Zheng, M., Tang, W., & Delmelle, E. (2017). Geocoding Fundamentals and Associated Challenges. *Geospatial Data Science Techniques and Applications*.

PT. Transportasi Jakarta. (2016). *Sejarah – PT Transportasi Jakarta*. https://transjakarta.co.id/tentang-transjakarta/sejarah/

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases* (Second Edition). O'Reilly Media.

Rosen, K. H. (2012). *Discrete Mathematics and Its Applications* (Seventh Edition). McGraw-Hill.

Steiniger, S., Neun, M., Edwardes, A., & Lenz, B. (2008). *Foundations of LBS*. http://www.e-cartouche.ch