

SKRIPSI

MODULARISASI ALGORITMA SHORTEST PATH PADA PERANGKAT LUNAK KIRI MENGGUNAKAN STRATEGY PATTERN



Muhammad Aldi Rivandi

NPM: 6182001029

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2025

UNDERGRADUATE THESIS

**MODULARIZATION OF THE SHORTEST PATH ALGORITHM
ON KIRI SOFTWARE USING STRATEGY PATTERN**



Muhammad Aldi Rivandi

NPM: 6182001029

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2025**

LEMBAR PENGESAHAN

MODULARISASI ALGORITMA SHORTEST PATH PADA PERANGKAT LUNAK KIRI MENGGUNAKAN STRATEGY PATTERN

Muhammad Aldi Rivandi

NPM: 6182001029

Bandung, «*tanggal*» «*bulan*» 2025

Menyetujui,

Pembimbing

Pascal Alfadian, Nugroho, M.Comp.

Ketua Tim Penguji

Anggota Tim Penguji

Rosa De Lima, M.T.

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

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

MODULARISASI ALGORITMA SHORTEST PATH PADA PERANGKAT LUNAK KIRI MENGGUNAKAN STRATEGY PATTERN

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

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

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» 2025



Muhammad Aldi Rivandi
NPM: 6182001029

ABSTRAK

Perangkat lunak KIRI merupakan aplikasi pencari rute transportasi publik yang arsitekturnya dibagi menjadi dua, yaitu Tirtayasa (*frontend*) dan NewMenjangan (*backend*). Saat ini Perangkat Lunak KIRI hanya memanfaatkan algoritma Dijkstra untuk perhitungan jalur terpendek. Penelitian ini bertujuan untuk mengembangkan perangkat lunak KIRI dengan menerapkan strategy pattern sebagai pendekatan modularisasi algoritma shortest path. Dengan penerapan strategy pattern, penambahan algoritma shortest path lainnya dapat dilakukan secara dinamis tanpa mengubah struktur kode utama.

Pada penelitian ini, akan ditambahkan atau diimplementasikan dua algoritma shortest path lainnya, yaitu Floyd-Warshall dan A-Star, yang dikembangkan dan diintegrasikan ke dalam backend KIRI sebagai strategi alternatif. Algoritma Floyd-Warshall dipilih karena kemampuannya dalam menghitung semua pasangan titik, sedangkan A-Star digunakan karena keunggulannya dalam efisiensi pencarian pada graf berbobot yang besar dan memiliki nilai heuristik lokasi.

Hasil pengujian menunjukkan bahwa modularisasi dengan strategy pattern meningkatkan fleksibilitas dalam pengelolaan algoritma pencarian jalur terpendek, serta memudahkan penambahan atau penggantian algoritma apabila akan dilakukan pemeliharaan atau pengembangan lebih lanjut. Selain itu, masing-masing algoritma menunjukkan performa yang berbeda sesuai dengan karakteristik graf dan skenario pencarian yang diuji, sehingga memberikan opsi lebih bagi pengguna dan pengembang sistem.

Kata-kata kunci: shortest path, strategy pattern, Dijkstra, Floyd-Warshall, A-Star, perangkat lunak KIRI, modularisasi

ABSTRACT

KIRI is a public transportation route-finding application whose architecture is divided into two components: Tirtayasa (*frontend*) and NewMenjangan (*backend*). Currently, the KIRI software only utilizes the Dijkstra algorithm for shortest path calculations. This research aims to enhance the KIRI software by implementing the strategy pattern as an approach to modularize the shortest path algorithms. By applying the strategy pattern, additional shortest path algorithms can be integrated dynamically without altering the core code structure.

In this study, two additional shortest path algorithms Floyd-Warshall and A-Star are implemented and integrated into the KIRI backend as alternative strategies. The Floyd-Warshall algorithm is chosen for its ability to compute all-pairs shortest paths, while the A-Star algorithm is selected for its efficiency in large, weighted graphs with heuristic location values.

The test results indicate that modularization using the strategy pattern improves the flexibility in managing shortest path algorithms and facilitates easier maintenance or future development involving algorithm addition or replacement. Furthermore, each algorithm demonstrates different performance characteristics depending on the graph type and search scenario, thereby offering more options for both users and system developers.

Keywords: shortest path, strategy pattern, Dijkstra, Floyd-Warshall, A-Star, KIRI software, modularization

*Tugas Akhir ini dipersiapkan untuk Tuhan Yang Maha Esa,
keluarga, dosen-dosen, dan teman-teman mahasiswa*

KATA PENGANTAR

Puji syukur dipanjatkan kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia beserta izinNya, penulis dapat menyelesaikan penulisan tugas akhir yang berjudul "Modularisasi Algoritma Shortest Path pada Perangkat Lunak KIRI Menggunakan Strategy Pattern". Penulis juga ingin mengucapkan terima kasih kepada pihak-pihak yang senantiasa memberikan dukungan serta bantuan selama penggerjaan tugas akhir ini. Penulis ingin mengucapkan terima kasih kepada:

1. Orang tua dan keluarga yang senantiasa memberikan dukungan, doa, motivasi, dan semangat selama penggerjaan tugas akhir ini.
2. Bapak Pascal Alfadian Nugroho, M.Comp. selaku dosen pembimbing yang senantiasa memberikan bimbingan, motivasi, dan waktunya dari awal hingga selesai sehingga tugas akhir ini dapat diselesaikan dengan baik.
3. Ibu Rosa De Lima, M.T. dan ... selaku dosen penguji yang telah memberikan kritik dan saran yang membangun.
4. Teman-teman yang senantiasa memberikan dukungan dan semangat, baik teman-teman mahasiswa UNPAR ataupun teman-teman lainnya.
5. Seluruh dosen serta staf dari Teknik Informatika UNPAR yang senantiasa memberikan bantuan dalam berbagai hal dari awal perkuliahan.

Penulis menyadari bahwa tugas akhir ini masih jauh dari sempurna. Oleh karena itu, penulis ingin mengucapkan permohonan maaf jika masih banyak kekurangan pada tugas akhir ini. Penulis berharap tugas akhir ini dapat bermanfaat bagi pembaca dan bagi pihak yang akan meneruskan penelitian ini.

Bandung, «bulan» 2025

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	4
2 LANDASAN TEORI	5
2.1 KIRI [1]	5
2.2 Design Pattern dan Strategy Pattern [2]	6
2.2.1 Contoh Kode Program	8
2.3 MySQL [3]	9
2.3.1 LineString [3]	10
2.4 Graf [4]	11
2.4.1 Graf Berarah dan Graf Tidak Berarah [4]	12
2.4.2 Graf Berbobot dan Graf Tidak Berbobot [4]	13
2.5 Algoritma Shortest Path	14
2.5.1 Algoritma Dijkstra [5]	14
2.5.2 Algoritma Floyd-Warshall [5]	16
2.5.3 Algoritma A* [6]	19
3 ANALISIS	21
3.1 Analisis Sistem Kini	21
3.1.1 Analisis Kelas	22
3.1.2 Pemodelan Graf	35
3.2 Analisis Sistem Usulan	35
3.2.1 Implementasi Strategy Pattern	36
3.2.2 Implementasi Algoritma A* dan Algoritma Floyd-Warshall	36
4 PERANCANGAN	37
4.1 Perancangan Kelas	37
4.2 Perancangan Antarmuka	38
4.3 Perancangan Protokol	39
5 IMPLEMENTASI DAN PENGUJIAN	41
5.1 Implementasi	41

5.1.1	Lingkungan Perangkat Keras	41
5.1.2	Lingkungan Perangkat Lunak	41
5.1.3	Penjelasan Kode Program	42
5.1.4	Implementasi Antarmuka	47
5.2	Pengujian Fungsional dan Eksperimental	47
5.2.1	Pengujian Fungsional	47
5.2.2	Pengujian Eksperimental	52
6	KESIMPULAN DAN SARAN	55
6.1	Kesimpulan	55
6.2	Saran	55
DAFTAR REFERENSI		57
A	KODE PROGRAM	59

DAFTAR GAMBAR

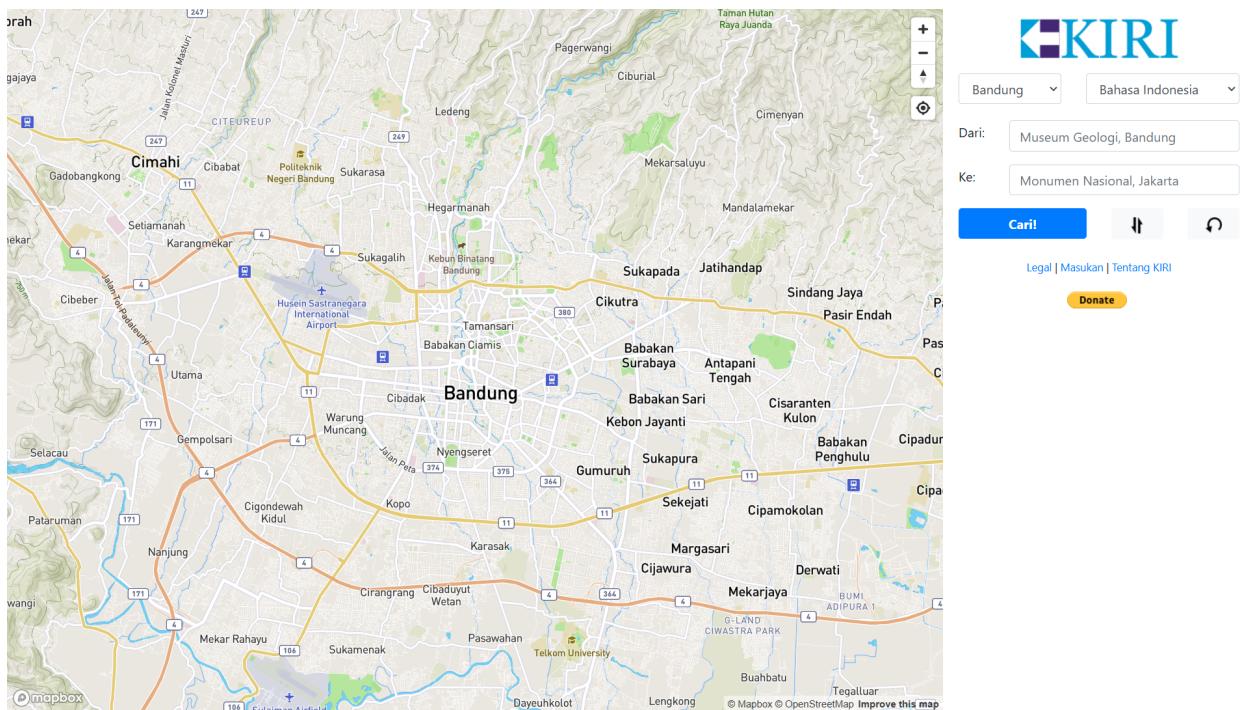
1.1	Tampilan halaman perangkat lunak KIRI	1
1.2	Tampilan perangkat lunak KIRI, setelah menerima masukan	2
2.1	Tampilan awal perangkat lunak KIRI	5
2.2	Tampilan perangkat lunak KIRI, setelah menerima masukan	6
2.3	Struktur Strategy Pattern	7
2.4	Graf	12
2.5	Graf Berarah	12
2.6	Graf Tidak Berbobot	13
3.1	Struktur Kelas NewMenjangan	21
3.2	Rute Ciroyom – Cicuheum	35
3.3	Struktur Strategy Pattern	36
4.1	Perancangan Kelas	37
4.2	Rancangan Antarmuka	38
5.1	Hasil Implementasi Antarmuka	47
5.2	Kondisi Saat Algoritma Floyd-Warshall Tidak Berhasil Dijalankan	48
5.3	Error Yang Menyebabkan Algoritma Floyd-Warshall Tidak Berhasil Dijalankan	48
5.4	Hasil Pengujian Algoritma Dijkstra Jalur 1	49
5.5	Hasil Pengujian Algoritma Dijkstra Jalur 2	49
5.6	Hasil Pengujian Algoritma Floyd-Warshall Jalur 1	50
5.7	Hasil Pengujian Algoritma Floyd-Warshall Jalur 2	50
5.8	Hasil Pengujian Algoritma A-Star Jalur 1	51
5.9	Hasil Pengujian Algoritma A-Star Jalur 2	51

BAB 1

PENDAHULUAN

1.1 Latar Belakang

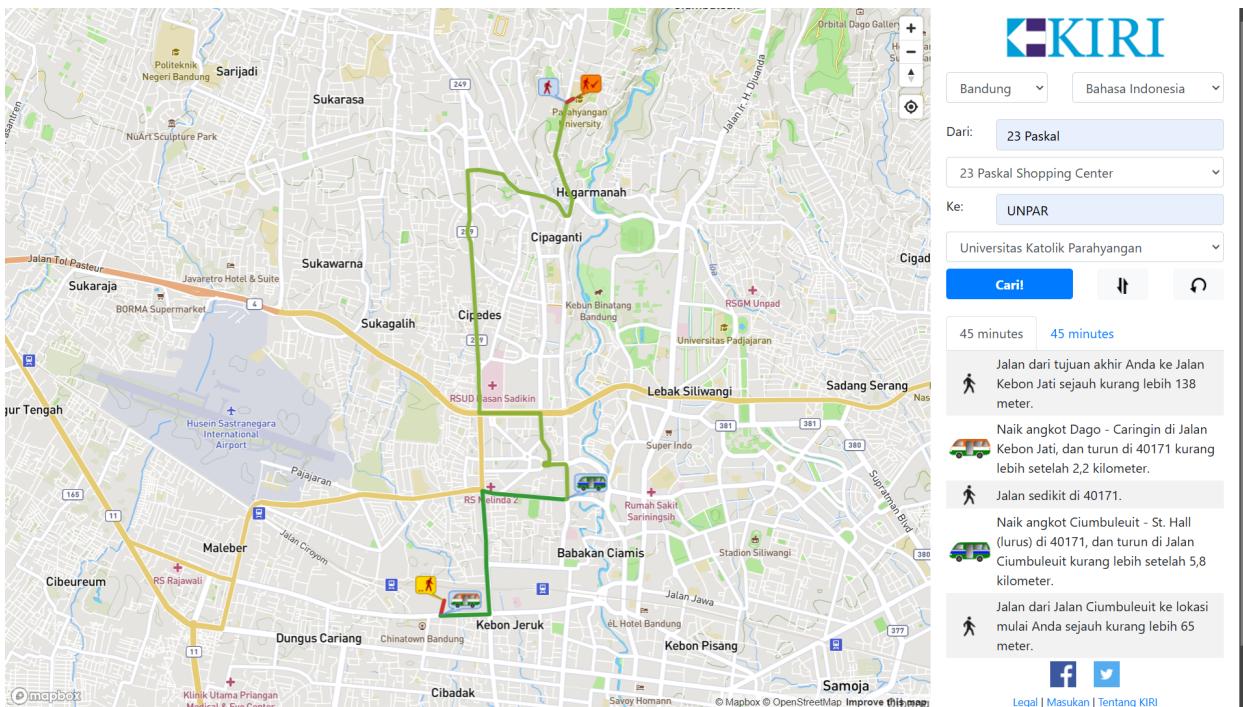
Perangkat lunak KIRI¹ (lihat Gambar 1.1) adalah perangkat lunak berbasis *web* yang dirancang untuk membantu pengguna menemukan rute perjalanan ketika menggunakan angkot untuk di Bandung serta TransJakarta dan Commuterline untuk di DKI Jakarta. Pada perangkat lunak KIRI, pengguna dapat memasukkan titik awal perjalanan dan titik tujuan. KIRI kemudian akan mencari berbagai alternatif rute yang bisa digunakan untuk mencapai tujuan tersebut.



Gambar 1.1: Tampilan halaman perangkat lunak KIRI

KIRI akan memberikan informasi mengenai langkah-langkah yang harus ditempuh oleh pengguna yang akan bepergian dari suatu tempat ke tempat tujuannya, mulai dari seberapa jauh pengguna harus berjalan untuk menaiki angkot yang bersangkutan, di mana pengguna harus naik atau turun angkot tersebut, seberapa jauh lagi pengguna harus berjalan sampai ke lokasi tujuan, dan seberapa lama estimasi waktu perjalanan yang akan ditempuh (lihat Gambar 1.2).

¹<https://projectkiri.id/> (diakses 14 Februari 2025)



Gambar 1.2: Tampilan perangkat lunak KIRI, setelah menerima masukan

Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Bagian *frontend*, yang dinamakan Tirtayasa dan dibangun menggunakan bahasa pemrograman PHP serta mengandalkan basis data MySQL untuk menyimpan serta mengelola data. Selain itu, Tirtayasa juga menggunakan framework CodeIgniter 3. Saat menerima permintaan pencarian, Tirtayasa meneruskannya ke bagian *backend*, yaitu NewMenjangan. Hasil dari NewMenjangan kemudian diformat agar dapat dibaca dengan baik oleh pengguna. Bagian ini diimplementasikan dalam bahasa pemrograman Java dan berperan penting dalam perhitungan rute optimal. [1]

NewMenjangan merupakan program *daemon* yang berjalan secara otomatis saat server dinyalakan dan terus beroperasi hingga server dimatikan. *Daemon* sendiri adalah program komputer yang berjalan dilatar belakang dan tidak berinteraksi langsung dengan pengguna². Pada saat eksekusi, NewMenjangan terhubung ke basis data MySQL untuk mengambil data rute angkot yang tersimpan dalam format LineString. LineString adalah salah satu tipe data geometris dalam MySQL yang mewakili satu atau lebih segmen garis yang terhubung. LineString terdiri dari urutan titik (*point*) yang membentuk jalur atau lintasan³. Setiap titik pada LineString merepresentasikan lokasi potensial untuk penumpang naik atau turun. Dari data tersebut, NewMenjangan membangun *weighted graph* dalam memori (RAM) dalam bentuk *adjacency list* dan melakukan prakomputasi. Setiap titik pada LineString menjadi akan *node*, dan antara titik ke-*i* dan titik ke-(*i*+1) dihubungkan dengan *edge*. Jika ada dua titik dari rute angkot berbeda yang berdekatan (jarak di bawah konstanta tertentu), maka dibuatkan juga *edge*, yang menunjukkan kemungkinan seseorang dapat turun dari suatu angkot dan naik ke angkot lainnya untuk meneruskan perjalanan.

²<https://www.ibm.com/docs/en/aix/7.1?topic=processes-> (diakses 14 Februari 2025)

³<https://dev.mysql.com/doc/refman/8.4/en/gis-linestring-property-functions.html> (diakses 14 Februari 2025)

Saat NewMenjangan menerima permintaan pencarian dari titik A ke titik B, kedua titik tersebut dijadikan *node* sementara, dan dibuatkan *edge* sementara ke *node-node* yang sudah ada sebelumnya, jika jaraknya di bawah konstanta tertentu. Pencarian jarak terdekat pada graf tersebut dilakukan menggunakan algoritma Dijkstra versi teroptimasi (*priority queue* dengan struktur data *heap*). Proses ini dapat dilakukan secara paralel dengan aman (*thread-safe*) tanpa mengubah graf utama. Seperti pada judul tugas akhir ini, akan dilakukan modularisasi algoritma *shortest path* pada perangkat lunak KIRI menggunakan strategy pattern. Modularisasi adalah membagi perangkat lunak menjadi beberapa komponen yang terpisah, yang disebut modul, yang masing-masing dapat diakses dan diberi nama secara independen tetapi bekerja bersama untuk memenuhi kebutuhan sistem [7]. Saat ini algoritma yang digunakan KIRI masih terikat dengan algoritma Dijkstra. Oleh karena itu, pada tugas akhir ini akan diimplementasikan algoritma lainnya, yaitu algoritma A* dan Floyd-Warshall sebagai *concrete strategy*. Selain itu, akan dilakukan juga penerapan arsitektur kelas *strategy pattern* sehingga aplikasi KIRI akan menjadi lebih fleksibel dalam pemilihan algoritma *shortest path* yang akan digunakan dan juga memudahkan apabila akan dilakukan perubahan atau perbaikan pada suatu algoritma yang digunakan.

1.2 Rumusan Masalah

Rumusan masalah berdasarkan latar belakang yang sudah dipaparkan adalah sebagai berikut:

1. Bagaimana cara melakukan perubahan kode pada NewMenjangan untuk menerapkan strategy pattern?
2. Bagaimana mengimplementasikan algoritma A* dan Floyd Warshall sebagai *concrete strategy*?

1.3 Tujuan

Tujuan berdasarkan rumusan masalah yang sudah dipaparkan adalah sebagai berikut:

1. Melakukan perubahan arsitektur kelas dengan menerapkan strategy pattern.
2. Melakukan implementasi algoritma A* dan Floyd Warshall.

1.4 Batasan Masalah

Batasan masalah pada tugas akhir ini adalah ukuran graf yang sangat besar, sehingga algoritma Floyd-Warshall tidak dapat berjalan dengan baik apabila seluruh data digunakan.

1.5 Metodologi

Metodologi yang akan digunakan dalam pembuatan tugas akhir ini adalah sebagai berikut:

1. Melakukan eksplorasi fungsi-fungsi dan cara kerja perangkat lunak KIRI.
2. Mempelajari modul-modul yang terdapat pada Tirtayasa dan NewMenjangan.
3. Mempelajari bahasa pemrograman PHP dan framework CodeIgniter 3.
4. Melakukan studi literatur mengenai penerapan arsitektur kelas strategy pattern.
5. Mempelajari cara kerja algoritma Dijkstra, A*, dan Floyd Warshall.
6. Mengubah implementasi algoritma Dijkstra yang sudah ada ke dalam strategy pattern.
7. Mengimplementasikan algoritma A-star dan Floyd Warshall.
8. Melakukan pengujian dan eksperimen.
9. Menulis dokumen tugas akhir.

1.6 Sistematika Pembahasan

Tugas akhir ini akan disusun menjadi beberapa bab sebagai berikut:

- **Bab 1:** Pendahuluan

Bab ini berisi latar belakang, rumusan masalah,tujuan, batasan masalah, metodologi, dan sistematika pembahasan.

- **Bab 2:** Landasan Teori

Bab ini berisi dasar dari teori-teori yang dibutuhkan dalam penelitian ini, seperti KIRI, *Design Pattern* dan *Strategy Pattern*, MySQL, LineString, Graf, dan juga algoritma-algoritma *shortest path* yang akan diimplementasikan, yaitu algoritma Dijkstra, algoritma Floyd-Warshall, dan algoritma A*.

- **Bab 3:** Analisis

Bab ini berisi analisis dari sistem KIRI saat ini yang belum mengimplementasikan *Strategy Pattern* serta hanya mengimplementasikan algoritma Dijkstra dan juga analisi sistem usulan yang akan mengimplementasikan *Strategy Pattern* serta 2 algoritma *shortest path* lainnya, yaitu algoritma Floyd-Warshall, dan algoritma A*.

- **Bab 4:** Perancangan

Bab ini berisi rancangan dari pengembangan yang akan dilakukan perangkat lunak KIRI.

- **Bab 5:** Implementasi dan Pengujian

Bab ini berisikan pengimplementasian yang akan dilakukan untuk pengembangan perangkat lunak KIRI. Selain itu, bab ini juga berisi pengujian fungsi-fungsi pada perangkat lunak KIRI setelah dilakukan pengembangan.

- **Bab 6:** Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengembangan perangkat lunak kiri dan juga saran untuk pengembangan berikutnya.

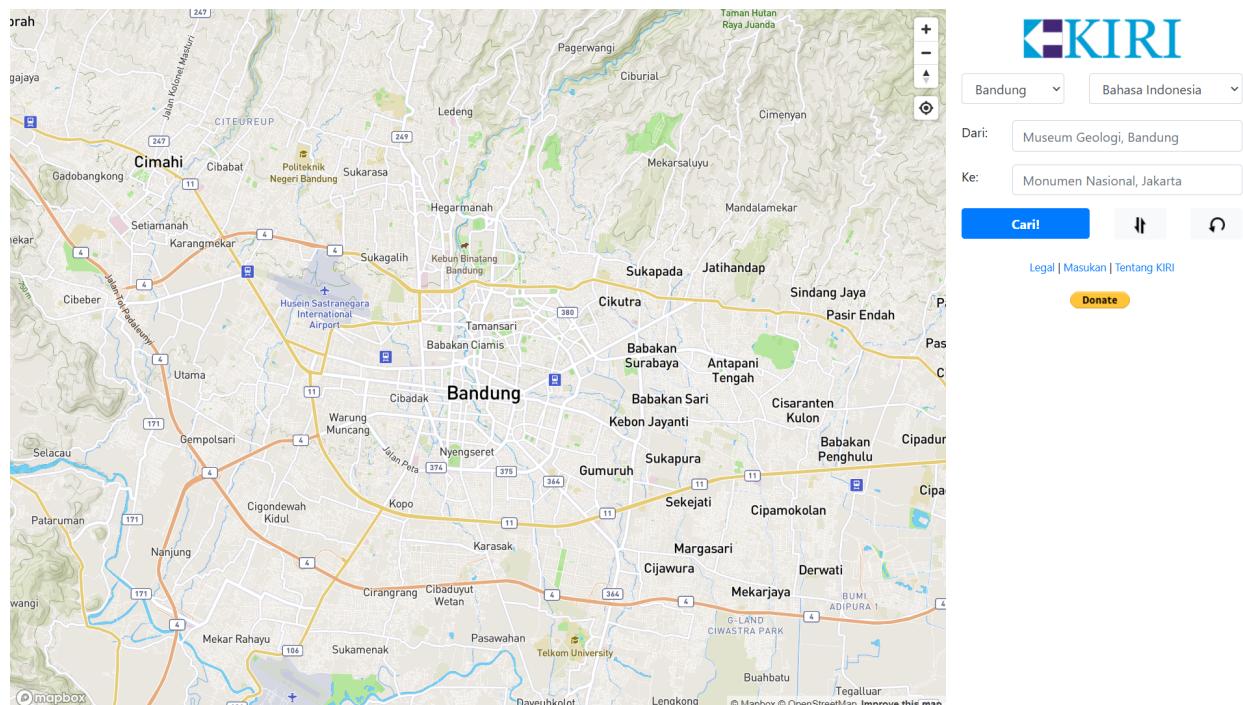
BAB 2

LANDASAN TEORI

Bab ini akan membahas dasar-dasar teori dalam pengembangan perangkat lunak KIRI. Pembahasan akan dimulai dari perangkat lunak KIRI, *design pattern* dan *strategy pattern* beserta contoh kode penerapan *strategy pattern*, MySQL beserta pembahasan mengenai *LineString*, penjelasan mengenai graf termasuk graf berarah dan graf tidak berarah serta graf berbobot dan graf tidak berbobot, dan algoritma *shortest path* yang didalamnya akan dijelaskan juga algoritma-algoritma *shortest path* yang diimplementasikan, yaitu Dijkstra, Floyd-Warshall, dan A-Star.

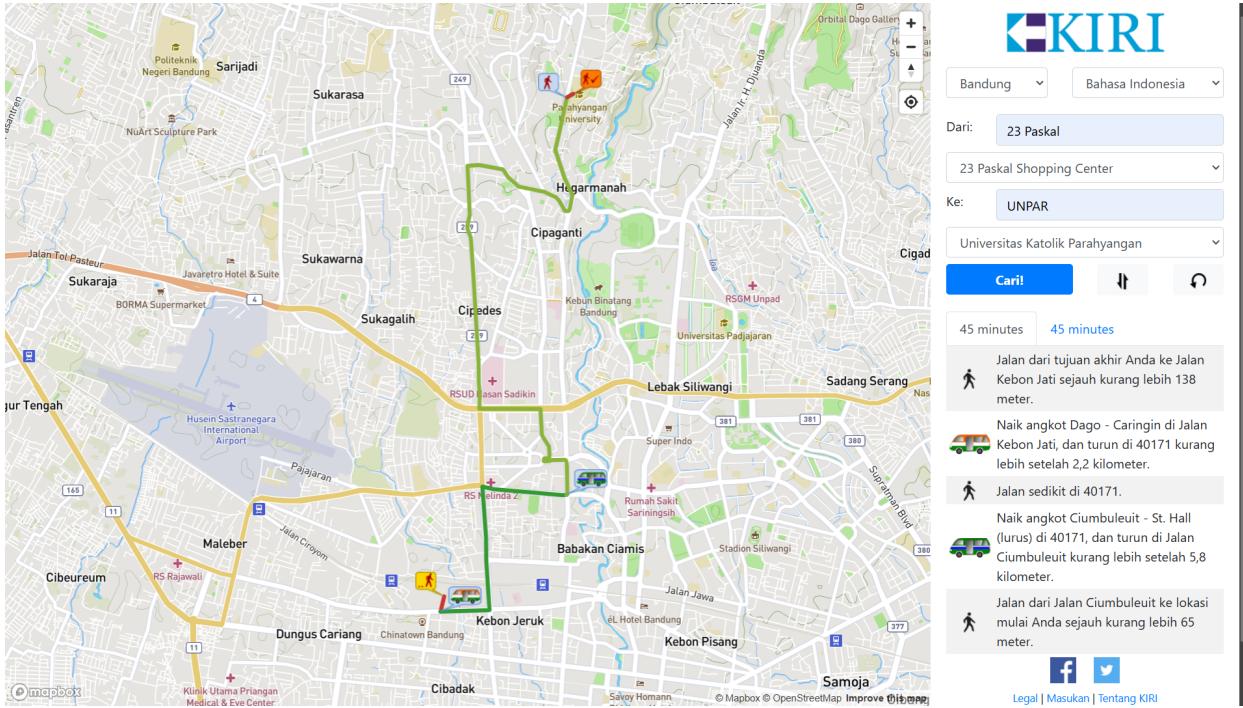
2.1 KIRI [1]

KIRI (lihat Gambar 2.1) adalah aplikasi navigasi transportasi umum berbasis web yang menyediakan rute antara dua lokasi geografis menggunakan transportasi publik. KIRI dirancang untuk melayani kebutuhan pengguna angkot (angkutan kota) di Bandung serta TransJakarta dan Commuterline di DKI Jakarta. Salah satu keunggulan KIRI dibandingkan layanan seperti Google Maps atau Moovit adalah kemampuannya memahami karakteristik transportasi publik, di mana penumpang dapat naik atau turun di sepanjang jalan tanpa terbatas pada halte tertentu.



Gambar 2.1: Tampilan awal perangkat lunak KIRI

KIRI akan memberikan informasi mengenai langkah-langkah yang harus ditempuh oleh pengguna yang akan berpergian dari satu tempat ke tempat tujuannya, mulai dari seberapa jauh pengguna harus berjalan untuk menaiki angkot yang bersangkutan, di mana pengguna harus naik atau turun angkot tersebut, seberapa jauh lagi pengguna harus berjalan sampai ke lokasi tujuan, dan seberapa lama estimasi waktu perjalanan yang akan ditempuh (lihat Gambar 2.2).



Gambar 2.2: Tampilan perangkat lunak KIRI, setelah menerima masukan

Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Yang pertama, yaitu Tirtayasa yang merupakan bagian *frontend* dari KIRI, dan bertanggung jawab sebagai antarmuka pengguna untuk browser web. Komponen ini mengubah nama tempat yang dimasukkan pengguna menjadi koordinat geografis dengan menggunakan bantuan Google Maps. Tirtayasa sendiri dibangun menggunakan PHP dan Framework CodeIgniter 4.

Selanjutnya, NewMenjangan, yang merupakan bagian *backend* dari KIRI dan dibangun dengan bahasa pemrograman Java serta digunakan untuk memproses permintaan navigasi. Komponen ini memuat semua jalur transportasi umum dalam bentuk graf dan menggunakan algoritma Dijkstra untuk menghitung rute optimal. Algoritma ini dipercepat dengan penggunaan struktur data heap, yang membuatnya efisien untuk jalur yang kompleks.

2.2 Design Pattern dan Strategy Pattern [2]

Design Pattern adalah solusi umum yang telah terbukti efektif untuk mengatasi masalah desain berulang dalam pengembangan perangkat lunak berorientasi objek. Solusi ini dirancang agar dapat digunakan kembali di berbagai konteks tanpa harus disesuaikan secara berlebihan. Sebagai contoh, pola desain membantu memecah masalah desain menjadi struktur yang lebih modular dan fleksibel, sehingga mempermudah pengembangan dan pemeliharaan perangkat lunak.

Pada dasarnya, design pattern memiliki empat elemen utama. Pertama, nama pola yang memberikan cara singkat untuk menyebut masalah desain tertentu, solusinya, dan konsekuensi dari penerapannya. Kedua, masalah, yaitu deskripsi konteks atau situasi di mana pola desain ini relevan. Ketiga, solusi yang berupa abstraksi dari elemen-elemen desain dan kolaborasinya tanpa menyebutkan implementasi konkret. Keempat, konsekuensi yang mencakup hasil dari penerapan

pola, termasuk dampak pada fleksibilitas, efisiensi, dan pengelolaan sistem.

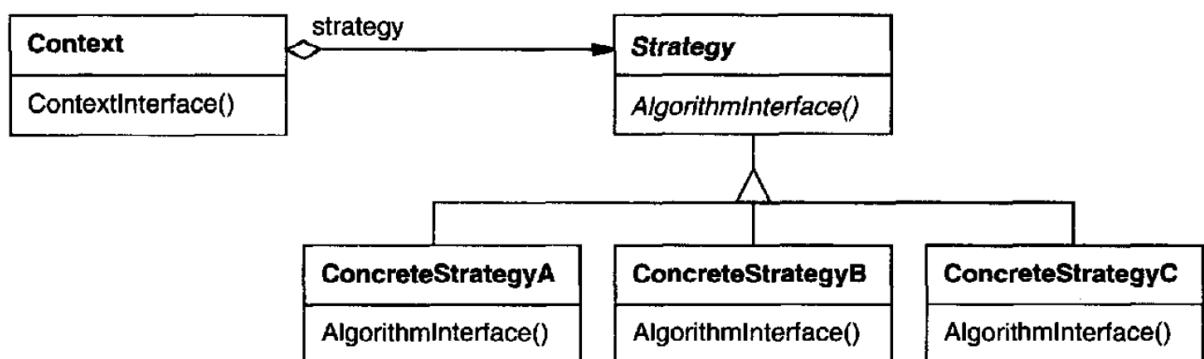
Penggunaan *design pattern* juga memungkinkan sistem menjadi lebih adaptif terhadap perubahan kebutuhan. Pola seperti *Strategy* mempermudah pergantian algoritma di runtime, sedangkan *Factory Method* membantu mengurangi ketergantungan pada implementasi spesifik dengan menyediakan cara fleksibel untuk membuat objek. Dengan demikian, design pattern mempermudah kolaborasi dan komunikasi antar tim pengembang.

Strategy Pattern merupakan salah satu pola desain perilaku yang dirancang untuk mendefinisikan serangkaian algoritma, mengenkapsulasi setiap algoritma, dan memungkinkan algoritma-algoritma tersebut untuk saling dipertukarkan. Pola ini memungkinkan algoritma untuk bervariasi. Dengan demikian, klien tidak perlu mengetahui detail implementasi dari algoritma yang digunakan, melainkan cukup berinteraksi melalui antarmuka umum yang disediakan oleh objek strategi.

Pola ini sangat berguna ketika terdapat kebutuhan untuk mendukung berbagai varian algoritma dalam menyelesaikan tugas yang sama. *Strategy Pattern* memindahkan setiap algoritma ke dalam kelas terpisah, yang disebut sebagai *concrete strategy*. Klien dapat memilih dan menentukan strategi yang sesuai ke dalam konteks pada waktu eksekusi, sehingga memberikan fleksibilitas yang tinggi dalam proses pengembangan perangkat lunak.

Manfaat utama dari *strategy pattern* adalah kemampuannya untuk menghilangkan kompleksitas yang diakibatkan oleh penggunaan pernyataan kondisional yang rumit dalam kode, serta kemudahan dalam menambahkan atau mengganti algoritma tanpa perlu memodifikasi kode klien atau konteks. Namun, penerapan pola ini juga memiliki kelemahan, seperti meningkatnya jumlah kelas dalam sistem dan potensi timbulnya *overhead* komunikasi antara konteks dan strategi. Oleh karena itu, penerapan *strategy pattern* sebaiknya dipertimbangkan dengan cermat, terutama dalam situasi di mana variasi algoritma memang diperlukan untuk memenuhi kebutuhan sistem.

Berikut merupakan struktur dari *strategy pattern* (Gambar 2.3).



Gambar 2.3: Struktur Strategy Pattern

2.2.1 Contoh Kode Program

Kode 2.1: IntegerSorter.java

```

1 abstract class IntegerSorter {
2     public abstract int[] sort(int[] arr);
3 }
```

Kode 2.2: ArraysSort.java

```

1 import java.util.*;
2
3 class ArraysSort extends IntegerSorter {
4     public int[] sort(int[] arr) {
5         Arrays.sort(arr);
6         return arr;
7     }
8 }
```

Kode 2.3: BubbleSort.java

```

1 class BubbleSort extends IntegerSorter {
2     public int[] sort(int[] arr) {
3         int n = arr.length;
4         for (int i = 0; i < n - 1; i++) {
5             for (int j = 0; j < n - i - 1; j++) {
6                 if (arr[j] > arr[j + 1]) {
7                     int temp = arr[j];
8                     arr[j] = arr[j + 1];
9                     arr[j + 1] = temp;
10                }
11            }
12        }
13        return arr;
14    }
15 }
```

Kode 2.4: InsertionSort.java

```

1 class InsertionSort extends IntegerSorter {
2     public int[] sort(int[] arr) {
3         int n = arr.length;
4         for (int i = 1; i < n; i++) {
5             int key = arr[i];
6             int j = i - 1;
7
8             while (j >= 0 && arr[j] > key) {
9                 arr[j + 1] = arr[j];
10                j = j - 1;
11            }
12            arr[j + 1] = key;
13        }
14        return arr;
15    }
16 }
```

Kode 2.5: Main.java

```

1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         int[] arr = { 6, 4, 21, 9, 14, 17, 3 };
6
7         IntegerSorter arraysSort = new ArraysSort();
8         System.out.println("Arrays.sort:" + Arrays.toString(
9             arraysSort.sort(arr.clone())));
10
11        IntegerSorter bubbleSort = new BubbleSort();
12        System.out.println("BubbleSort:" + Arrays.toString(
13            bubbleSort.sort(arr.clone())));
14
15        IntegerSorter insertionSort = new InsertionSort();
16        System.out.println("InsertionSort:" + Arrays.toString(
17            insertionSort.sort(arr.clone())));
18    }
19 }
```

2.3 MySQL [3]

MySQL merupakan sistem manajemen basis data relasional (*Relational Database Management System/RDBMS*) bersifat open source yang dikembangkan oleh *Oracle Corporation*. SQL, yang merupakan singkatan dari *Structured Query Language*, adalah bahasa pemrograman yang digunakan untuk mengambil, memperbarui, menghapus, serta memanipulasi data pada basis data relasional. Sebagai basis data relasional, MySQL menyimpan data dalam bentuk tabel yang terdiri atas baris dan kolom, yang disusun dalam suatu skema. Skema ini bertugas mendefinisikan bagaimana data diorganisasi dan disimpan, serta menjelaskan hubungan antara tabel-tabel yang ada di dalamnya. Dalam MySQL, terdapat berbagai sintaks yang digunakan untuk mendukung pengelolaan basis data. Sintaks-sintaks tersebut mencakup operasi penting, seperti pembuatan tabel, penyisipan data, pembaruan data, penghapusan data, hingga pengambilan data. Setiap sintaks dirancang untuk mempermudah pengguna dalam mengelola data secara efektif dan efisien sesuai kebutuhan sistem. Berikut merupakan sintaks-sintaks dasar yang umum digunakan dalam MySQL.

- ***CREATE DATABASE***

```
1 CREATE DATABASE database_name;
```

Sintaks tersebut digunakan untuk membuat database baru dalam MySQL. `database_name` diisi nama dari database baru yang akan dibuat.

- ***DROP DATABASE***

```
1 DROP DATABASE database_name;
```

Sintaks tersebut digunakan untuk menghapus database yang telah dibuat dalam MySQL. `database_name` diisi nama dari database yang akan dihapus.

- ***CREATE TABLE***

```

1 CREATE TABLE table_name (
2     column1 datatype,
3     column2 datatype,
4     column3 datatype,
5     ...
6 );
```

Sintaks tersebut digunakan untuk membuat atau memasukan tabel baru kedalam sebuah database. `table_name` diisi nama dari tabel yang akan dibuat, `column1`, `column2`, `column3`

dan seterusnya diisi dengan nama kolom didalam tabel yang akan dibuat, dan **datatype** diisi dengan tipe data dari kolom yang akan dibuat, seperti **varchar**, **integer**, **date**, dan lain-lain.

- **DROP TABLE**

```
1 DROP TABLE table_name;;
```

Sintaks tersebut digunakan untuk menghapus tabel yang telah dibuat dalam sebuah database. **table_name** diisi nama dari tabel yang akan dihapus.

- **SELECT**

```
1 SELECT column1, column2, ...
2 FROM table_name;
```

Sintaks tersebut digunakan untuk memilih atau mengambil data dari sebuah tabel dalam database. **column1**, **column2** dan seterusnya diisi dengan nama kolom dari sebuah tabel yang datanya akan diambil dan **table_name** diisi dengan nama tabel dimana kolom tersebut berada.

- **WHERE**

```
1 SELECT column1, column2, ...
2 FROM table_name
3 WHERE condition;
```

Sintaks tersebut digunakan untuk memilih atau mengambil data dari sebuah tabel dalam database dengan sebuah kondisi tertentu yang bertujuan untuk memfilter data yang akan diambil. **column1**, **column2** dan seterusnya diisi dengan nama kolom dari sebuah tabel yang datanya akan diambil, **table_name** diisi dengan nama tabel dimana kolom tersebut berada, dan **condition** diisi dengan kondisi dari data yang akan diambil atau filter seperti apa yang dingin dilakukan ketika mengambil data.

- **INSERT INTO**

```
1 INSERT INTO table_name (column1, column2, column3, ...)
2 VALUES (value1, value2, value3, ...);
```

Sintaks tersebut digunakan untuk memasukan atau menambahkan data baru kedalam kolom dari sebuah tabel yang telah ada. **table_name** diisi nama tabel yang akan ditambahkan data baru, **column1**, **column2**, **column3** dan seterusnya diisi dengan nama kolom yang akan ditambahkan data baru, dan **value** diisi dengan nilai atau **value** dari data baru yang akan ditambahkan.

- **DELETE**

```
1 DELETE FROM table_name
2 WHERE condition;
```

Sintaks tersebut digunakan untuk menghapus data baru kedalam kolom dari sebuah tabel yang telah ada. **table_name** diisi nama tabel yang akan ditambahkan data baru, **column1**, **column2**, **column3** dan seterusnya diisi dengan nama kolom yang akan ditambahkan data baru, dan **value1**, **value2**, **value3** dan seterusnya diisi dengan nilai atau **value** dari data baru yang akan ditambahkan.

2.3.1 LineString [3]

LineString adalah tipe data geometris dalam MySQL yang mewakili jalur atau lintasan yang terdiri dari satu atau lebih segmen garis yang terhubung. Tipe data ini digunakan dalam Sistem Informasi Geografis (GIS) untuk merepresentasikan lintasan seperti jalan, sungai, atau rute perjalanan. Setiap *LineString* terdiri dari urutan titik (point) yang memiliki koordinat (x, y) dan minimal memiliki dua titik untuk membentuk garis.

Untuk memanipulasi dan menganalisis *LineString*, MySQL menyediakan sejumlah fungsi bawaan. Sebelum fungsi tersebut digunakan, objek *LineString* biasanya dikonversi ke bentuk geometris menggunakan fungsi **ST_GeomFromText()**. Fungsi ini menerima teks representasi geometris, seperti *LineString(x1y1, x2y2, ...)* dan mengubahnya menjadi objek geometris yang dapat diproses oleh

fungsi GIS lainnya. Berikut adalah beberapa fungsi penting yang dapat digunakan untuk bekerja dengan *LineString*:

- ST_EndPoint(ls)

Mengembalikan titik akhir dari *LineString* ls. Contoh:

```
1 SET @ls = 'LineString(1_1,_2_2,_3_3)';
2 SELECT ST_AsText(ST_EndPoint(ST_GeomFromText(@ls)));
3 -- Hasil: 'POINT(3,3)'
```

- ST_IsClosed(ls)

Mengecek apakah *LineString* ls membentuk lintasan tertutup (titik awal dan akhir sama). Contoh:

```
1 SET @ls = 'LineString(1_1,_2_2,_3_3,_1_1)';
2 SELECT ST_IsClosed(ST_GeomFromText(@ls));
3 -- Hasil: 1 (TRUE)
```

- ST_Length(ls)

Menghitung panjang total *LineString* ls. Contoh:

```
1 SET @ls = 'LineString(1_1,_2_2,_3_3)';
2 SELECT ST_Length(ST_GeomFromText(@ls));
3 -- Hasil: 2.828427
```

- ST_NumPoints(ls)

Mengembalikan jumlah titik yang membentuk *LineString* ls. Contoh:

```
1 SET @ls = 'LineString(1_1,_2_2,_3_3)';
2 SELECT ST_NumPoints(ST_GeomFromText(@ls));
3 -- Hasil: 3
```

- ST_Point(ls, N)

Mengembalikan titik ke-N pada *LineString* ls. Contoh:

```
1 SET @ls = 'LineString(1_1,_2_2,_3_3)';
2 SELECT ST_AsText(ST_Point(ST_GeomFromText(@ls), 2));
3 -- Hasil: 'POINT(2,2)'
```

- ST_StartPoint(ls)

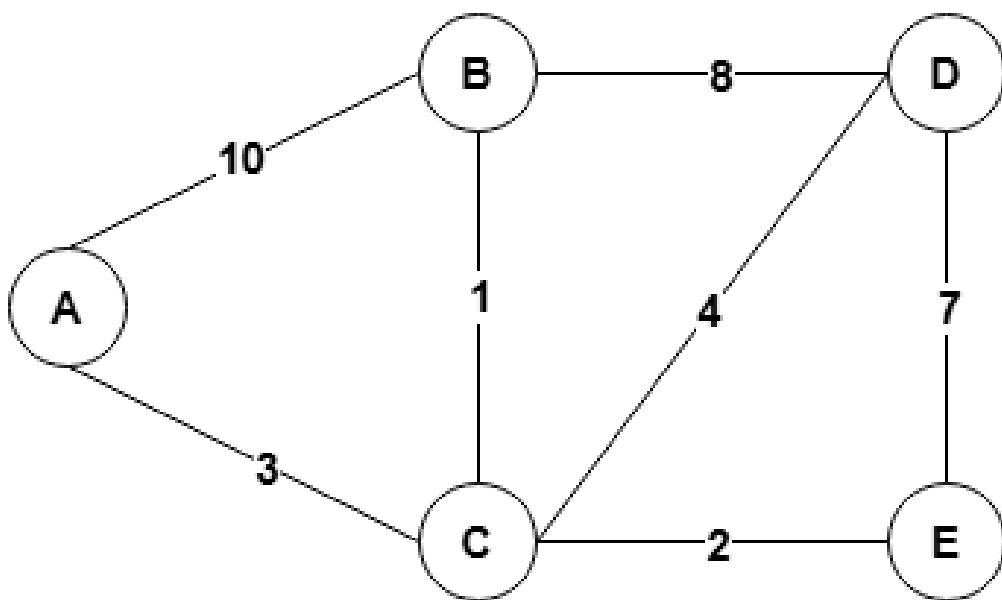
Mengembalikan titik awal dari *LineString* ls. Contoh:

```
1 SET @ls = 'LineString(1_1,_2_2,_3_3)';
2 SELECT ST_AsText(ST_StartPoint(ST_GeomFromText(@ls)));
3 -- Hasil: 'POINT(1,1)'
```

2.4 Graf [4]

Graf (G) adalah struktur yang terdiri dari simpul (*vertex*) dan sisi (*edge*), dimana sisi menghubungkan pasangan simpul (lihat Gambar 2.4). Sebuah graf direpresentasikan sebagai pasangan $G = (V, E)$, dengan V sebagai himpunan simpul dan E sebagai himpunan sisi. Sisi diwakili oleh pasangan simpul yang terhubung. Graf dapat bersifat terarah atau tidak terarah. Simpul-simpul dalam graf dapat memiliki derajat tertentu, yaitu jumlah sisi yang menghubunginya. Sebuah graf disebut terhubung jika terdapat jalur antara setiap pasangan simpul. Jalur ini adalah urutan simpul yang dihubungkan oleh sisi. Selain itu, siklus adalah jalur tertutup di mana simpul awal dan akhir adalah sama. Teori graf juga mencakup konsep seperti pohon (graf terhubung tanpa siklus), graf bipartit (simpul dibagi menjadi dua himpunan yang saling bebas sisi), dan subgraf (bagian dari graf yang tetap mempertahankan struktur graf).

Graf digunakan dalam berbagai hal, seperti jaringan komputer, rute transportasi, dan analisis hubungan sosial. Teori graf menyediakan dasar matematis untuk mempelajari struktur ini dan memberikan cara untuk memodelkan dan menyelesaikan masalah kompleks di berbagai bidang.

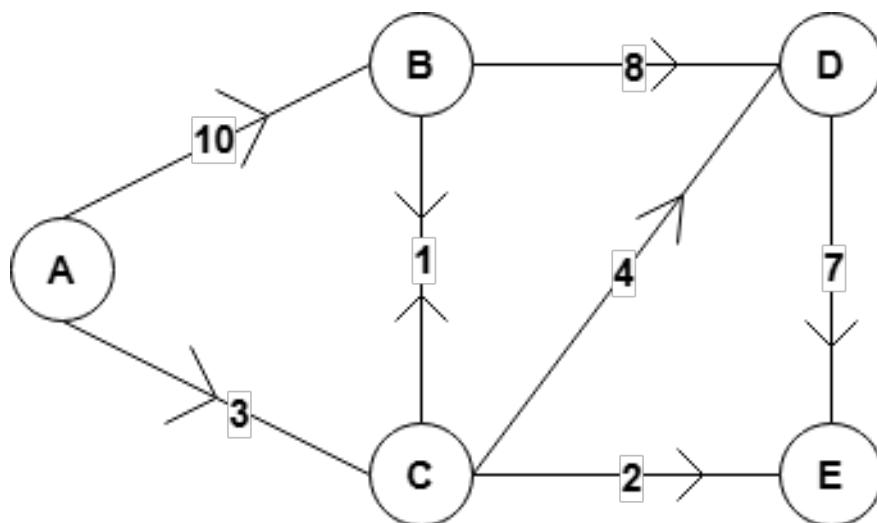


Gambar 2.4: Graf

2.4.1 Graf Berarah dan Graf Tidak Berarah [4]

Graf berarah (lihat Gambar 2.5) adalah graf di mana setiap sisi memiliki arah tertentu. Dengan kata lain, setiap sisi dalam graf ini diwakili oleh pasangan terurut dari simpul (*vertex*). Sebuah graf berarah didefinisikan sebagai pasangan $G = (V, E)$, di mana V adalah himpunan simpul dan E adalah himpunan sisi yang berbentuk pasangan terurut dari simpul. Setiap sisi e memiliki simpul awal $init(e)$ dan simpul terminal $ter(e)$ yang menunjukkan arah perjalanan dari satu simpul ke simpul lainnya.

Graf berarah dapat memiliki berbagai karakteristik khusus. Jika terdapat lebih dari satu sisi dengan arah yang sama antara dua simpul yang sama, sisi tersebut disebut sisi paralel. Jika sebuah sisi memiliki awal dan terminal yang sama, maka sisi tersebut disebut loop. Selain itu, jika suatu graf tidak berarah diberikan arah pada setiap sisinya, maka graf berarah yang dihasilkan disebut sebagai *oriented graph*.



Gambar 2.5: Graf Berarah

Graf tidak berarah (lihat Gambar 2.4) adalah jenis graf di mana setiap sisi hanya menghubungkan dua simpul tanpa adanya arah tertentu. Dalam graf ini, sisi diwakili sebagai pasangan tidak terurut $\{u, v\}$ yang berarti bahwa pergerakan dari simpul u ke v sama dengan pergerakan dari v ke u . Graf tidak berarah dapat direpresentasikan dalam bentuk matriks ketetanggaan (*adjacency matrix*), di mana matriks ini bersifat simetris.

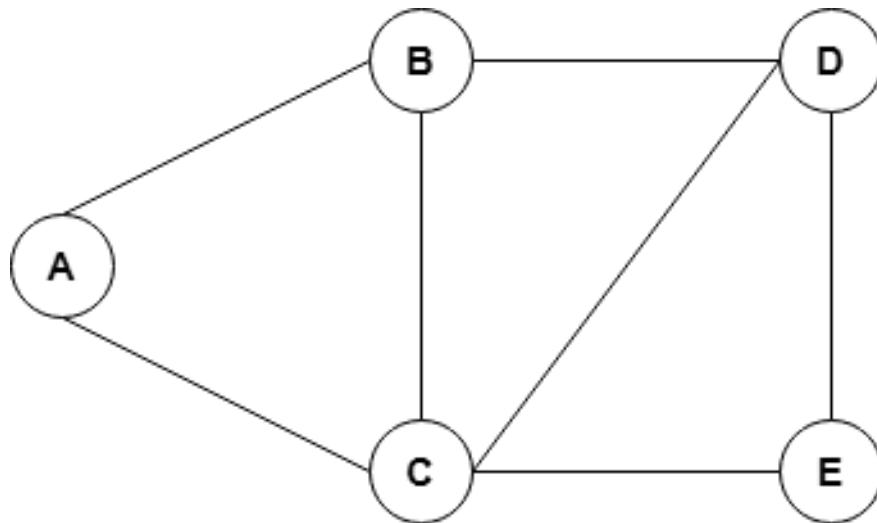
Salah satu sifat penting dari graf tidak berarah adalah konsep keterhubungan. Sebuah graf dikatakan terhubung jika terdapat jalur antara setiap pasangan simpulnya. Jika terdapat simpul yang tidak dapat dijangkau dari simpul lain, maka graf tersebut disebut tidak terhubung. Graf tidak berarah juga memiliki properti siklus, yaitu jalur tertutup di mana simpul awal dan simpul akhir adalah sama.

2.4.2 Graf Berbobot dan Graf Tidak Berbobot [4]

Graf berbobot (lihat Gambar 2.4) adalah graf di mana setiap sisi memiliki bobot atau nilai numerik yang menyatakan karakteristik tertentu seperti jarak, biaya, atau kapasitas. Graf berbobot direpresentasikan sebagai $G = (V, E, w)$, di mana $w : E \rightarrow R$ adalah fungsi yang memberikan bobot pada setiap sisi.

Graf berbobot sering digunakan dalam berbagai aplikasi seperti algoritma pencarian jalur terpendek, optimasi jaringan, dan model transportasi. Dalam beberapa kasus, graf berbobot dapat berarah atau tidak berarah, tergantung pada apakah bobot tersebut memiliki arah tertentu atau tidak.

Graf tidak berbobot (lihat Gambar 2.6) adalah graf di mana semua sisi dianggap memiliki bobot yang sama, biasanya bernilai satu atau dianggap tidak berbobot sama sekali. Graf jenis ini sering digunakan dalam analisis struktur jaringan, di mana hanya penting untuk mengetahui adanya hubungan antar simpul tanpa mempertimbangkan seberapa kuat atau penting hubungan tersebut. Dalam graf tak berbobot, algoritma seperti BFS (Breadth-First Search) dan DFS (Depth-First Search) sangat berguna untuk menemukan keterhubungan, jalur terpendek dalam jumlah langkah, serta untuk eksplorasi struktur graf lainnya.



Gambar 2.6: Graf Tidak Berbobot

2.5 Algoritma Shortest Path

Ada berbagai jenis algoritma *shortest path* yang dirancang untuk menemukan lintasan terpendek antara dua titik dalam sebuah graf. Algoritma ini memainkan peran penting dalam berbagai aplikasi, seperti sistem navigasi, perencanaan jaringan, analisis data geografis, dan pemecahan masalah rute optimal.

Setiap algoritma memiliki pendekatan, kelebihan, dan keterbatasan masing-masing, yang menjadikannya lebih sesuai untuk situasi tertentu. Sebagai contoh, algoritma seperti *Dijkstra* sangat cocok untuk graf dengan bobot positif, sementara *Floyd-Warshall* lebih sesuai untuk menemukan lintasan terpendek pada semua pasangan titik dalam graf kecil. Di sisi lain, algoritma *A** dirancang khusus untuk mempercepat pencarian lintasan dengan memanfaatkan heuristik. Berikut pembahasan secara detail tiga algoritma *shortest path* yang digunakan, yaitu *Dijkstra*, *Floyd-Warshall*, dan *A**.

2.5.1 Algoritma Dijkstra [5]

Algoritma Dijkstra merupakan sebuah algoritma untuk menyelesaikan masalah *single-source shortest path*, yaitu menemukan jalur terpendek dari satu titik asal ke semua titik lainnya dalam sebuah graf berarah dengan bobot tepi non-negatif. Algoritma ini menggunakan sebuah struktur *min-priority queue* yang menyimpan titik-titik dengan prioritas sesuai dengan perkiraan jarak terpendek dari titik asal. Algoritma ini memiliki kompleksitas waktu $O((V + E) \log V)$.

Algorithm 1 Dijkstra(G, w, s)

```

1: Initialize-Single-Source( $G, s$ )
2:  $S = \emptyset$ 
3:  $Q = G.V$ 
4: while  $Q \neq \emptyset$  do
5:    $u = \text{Extract-Min}(Q)$ 
6:    $S = S \cup \{u\}$ 
7:   for each vertex  $v \in G.\text{Adj}[u]$  do
8:     Relax( $u, v, w$ )
9:   end for
10: end while
```

Pseudocode 1 merupakan pseudocode dari algoritma Dijkstra. Pada pseudocode terdapat beberapa atribut diantarnya, yaitu G yang merepresentasikan graf, w merupakan bobot yang menyatakan jarak atau biaya antar dan s merepresentasikan simpul sumber yang merupakan titik awal pencarian. Selain itu, S merepresentasikan kumpulan simpul yang sudah diproses yang diawali diinisialisasi kosong, sedangkan Q merepresentasikan kumpulan simpul yang belum diproses, kemudian u merepresentasikan simpul yang sedang diproses dan v merepresentasikan simpul tetangga dari u . Algoritma Dijkstra dimulai dengan menginisialisasi perkiraan jarak terpendek dari titik asal s ke semua titik lain, kecuali s itu sendiri yang diinisialisasi dengan jarak 0 dan juga semua simpul dimasukkan ke dalam *min-priority queue* (Q), di mana prioritas ditentukan berdasarkan jarak terpendek yang diketahui. Selanjutnya, algoritma memproses simpul-simpul satu per satu dengan memilih simpul u dari Q yang memiliki jarak terpendek. Simpul tersebut kemudian ditambahkan ke dalam himpunan S .

Setelah simpul u diproses, algoritma akan memeriksa semua tetangga v dari u . Untuk setiap tetangga, algoritma melakukan proses *relaksasi*, yaitu membandingkan jarak saat ini ke v dengan jarak yang melewati u . Jika jalur melalui u memberikan jarak yang lebih pendek, jarak ke v diperbarui dengan jarak baru tersebut, dan simpul pendahulu v diatur menjadi u . Proses ini memastikan bahwa jalur terpendek ditemukan secara bertahap melalui iterasi. Algoritma akan terus berjalan hingga semua simpul telah diproses atau antrean Q kosong. Hasil akhir berupa jarak

terpendek dari simpul sumber s ke setiap simpul lain dalam graf.

Graf pada gambar 2.4 dapat diselesaikan menggunakan Dijkstra dengan langkah-langkah berikut.

1. Inisialisasi

- Atur jarak semua simpul ke ∞ (tak hingga), kecuali simpul A yang diset ke 0.
- Masukan semua simpul kedalam *Priority Queue*.
- PQ : $\{(A, 0), (B, \infty), (C, \infty), (D, \infty), (E, \infty)\}$

Simpul	Jarak Dari A	Jalur
A	0	A
B	∞	-
C	∞	-
D	∞	-
E	∞	-

2. Iterasi 1

- Ambil simpul A (0) dari PQ
- Periksa dan Perbarui jarak ke tetangga
 - B = $\min(\infty, 0 + 10) = 10$
 - C = $\min(\infty, 0 + 3) = 3$
- PQ : $\{(C, 3), (B, 10), (D, \infty), (E, \infty)\}$

Simpul	Jarak Dari A	Jalur
A	0	A
B	10	A - B
C	3	A - C
D	∞	-
E	∞	-

3. Iterasi 2

- Ambil simpul C (3) dari PQ
- Periksa dan Perbarui jarak ke tetangga
 - B = $\min(10, 3 + 1) = 4$
 - D = $\min(\infty, 3 + 4) = 7$
 - E = $\min(\infty, 3 + 2) = 5$
- PQ : $\{(B, 4), (E, 5), (D, 7), (B, 10)\}$

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

4. Iterasi 3

- Ambil simpul B (4) dari PQ
- Periksa dan Perbarui jarak ke tetangga
– $D = \min(7, 4 + 8) = 7$
- PQ : {(E, 5), (D, 7), (B, 10)}

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

5. Iterasi 4

- Ambil simpul E (5) dari PQ
- Periksa dan Perbarui jarak ke tetangga
– $D = \min(7, 5 + 7) = 7$
- PQ : {(D, 7)}

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

6. Iterasi 5

- Ambil simpul D (7) dari PQ
- Tidak ada tetangga atau PQ sudah kosong.

Simpul	Jarak Dari A	Jalur
A	0	A
B	4	A - C - B
C	3	A - C
D	7	A - C - D
E	5	A - C - E

2.5.2 Algoritma Floyd-Warshall [5]

Algoritma Floyd-Warshall merupakan sebuah algoritma untuk menyelesaikan masalah jalur terpendek untuk semua pasangan titik dalam graf berarah dengan menggunakan pendekatan pemrograman dinamis. Algoritma ini sangat berguna untuk graf yang memiliki bobot sisi negatif, selama tidak terdapat siklus dengan bobot negatif dalam graf tersebut. Pendekatan ini menghitung jalur terpendek antara semua pasangan titik dengan menggunakan tabel bobot antar titik dan mengulanginya secara bertahap untuk mencapai solusi optimal.

Algorithm 2 Floyd-Warshall(W)

```

1:  $n = W.rows$ 
2:  $D^{(0)} = W$ 
3: for  $k = 1$  to  $n$  do
4:   Let  $D^{(k)} = (d_{ij}^{(k)})$  be a new  $n \times n$  matrix
5:   for  $i = 1$  to  $n$  do
6:     for  $j = 1$  to  $n$  do
7:        $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ 
8:     end for
9:   end for
10: end for
11: return  $D^{(n)}$ 
```

Pseudocode 2 merupakan pseudocode dari algoritma Floyd-Warshall. Pada pseudocode terdapat beberapa atribut diantarnya, yaitu W yang merupakan sebuah matriks berbobot berukuran $n * n$ dan mewakili bobot dari setiap sisi pada graf, $D^{(0)}$ atau $D^{(k)}$ merupakan Matriks berukuran $n * n$ pada iterasi ke- k , n merepresentasikan banyaknya simpul dalam graf, diperoleh dari jumlah baris dari matriks W . Selain itu, terdapat $d_{ij}^{(k)}$ yang merupakan elemen dari matriks $D^{(k)}$ yang menunjukkan jarak terpendek antara simpul i dan j pada iterasi ke- k . Algoritma Floyd-Warshall dimulai dengan menginisialisasi n yang diinisialisasi dengan nilai baris pada matriks W dan juga $D^{(0)}$ yang diinisialisasi dengan matriks W .

Selama n iterasi, algoritma memperbarui matriks jarak terpendek dengan mempertimbangkan kemungkinan jalan melalui simpul antara $\{1, 2, \dots, k\}$. Pada setiap langkah, algoritma memeriksa apakah jarak dari i ke j dapat diperpendek dengan melalui simpul k , dibandingkan dengan jarak langsung antara i dan j . Proses ini menghasilkan solusi optimal, di mana $D^{(n)}$ mencakup semua jarak terpendek yang memungkinkan. Algoritma Floyd-Warshall memiliki kompleksitas waktu $O(n^3)$ karena terdiri dari tiga *looping* untuk semua titik dalam graf.

Graf pada gambar 2.4 dapat diselesaikan menggunakan Floyd-Warshall dengan langkah-langkah berikut.

1. Inisialisasi Matriks Jarak

- Setiap jarak awal diisi berdasarkan bobot sisi yang ada. Jika tidak ada sisi antara dua simpul, diisi dengan ∞ (tak hingga).

	A	B	C	D	E
A	0	10	3	∞	∞
B	∞	0	1	8	∞
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

2. Iterasi 1 dengan A sebagai perantara(k) / $k = A$

- Tidak ada perubahan karena A hanya memiliki jalur langsung ke B dan C dan tidak ada jalur baru yang lebih pendek yang bisa ditemukan.

	A	B	C	D	E
A	0	10	3	∞	∞
B	∞	0	1	8	∞
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

3. Iterasi 2 dengan B sebagai perantara(k) / k = B

- Tidak ada jalur lebih pendek yang ditemukan melalui B sehingga tidak ada perubahan.

	A	B	C	D	E
A	0	10	3	∞	∞
B	∞	0	1	8	∞
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

4. Iterasi 3 dengan C sebagai perantara (k) / k = C

- A - B dapat melalui C
 - A - C - B = $3 + 1 = 4$
 - Jarak langsung A - B adalah 10, jadi jarak A - B diperbarui dengan jarak A - C - B.
- A - D dapat melalui C
 - A - C - D = $3 + 4 = 7$
 - Jarak A - D diisi dengan jarak A - C - D.
- A - E dapat melalui C
 - A - C - E = $3 + 2 = 5$
 - Jarak A - E diisi dengan jarak A - C - E.
- B - D dapat melalui C
 - B - C - D = $1 + 4 = 5$
 - Jarak B - D diisi dengan jarak B - C - D.
- B - E dapat melalui C
 - B - C - E = $1 + 2 = 4$
 - Jarak B - E diisi dengan jarak B - C - E.

	A	B	C	D	E
A	0	4	3	7	5
B	∞	0	1	5	3
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

5. Iterasi 4 dengan D sebagai perantara (k) / k = D

- Tidak ada jalur lebih pendek yang ditemukan melalui D sehingga tidak ada perubahan.

	A	B	C	D	E
A	0	4	3	7	5
B	∞	0	1	5	3
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

6. Iterasi 5 dengan E sebagai perantara (k) / k = E

- Titik E sudah merupakan titik terakhir atau titik tujuan sehingga tidak ada perubahan.

	A	B	C	D	E
A	0	4	3	7	5
B	∞	0	1	5	3
C	∞	1	0	4	2
D	∞	∞	∞	0	7
E	∞	∞	∞	∞	0

2.5.3 Algoritma A* [6]

Algoritma A* adalah metode pencarian yang meminimalkan estimasi total biaya solusi dengan menggabungkan dua fungsi, yaitu $g(n)$ dan $h(n)$. Fungsi $g(n)$ menghitung biaya aktual dari titik awal hingga simpul n , sedangkan $h(n)$ memperkirakan biaya tersisa dari n ke tujuan. Kombinasi ini menghasilkan $f(n) = g(n) + h(n)$, yang memberikan perkiraan total biaya solusi jika rute melalui simpul n . Algoritma ini biasanya dipilih karena dapat mencapai solusi yang optimal dan lengkap, terutama jika fungsi heuristik $h(n)$ memenuhi kriteria tertentu.

Kondisi utama yang diperlukan agar algoritma A* memberikan solusi optimal adalah heuristik $h(n)$ yang bersifat *admissible*, yaitu tidak pernah melebih-lebihkan biaya ke tujuan, dan *consistent* atau *monotonic*, di mana nilai h tidak menurun di sepanjang jalur. Dengan adanya heuristik yang memenuhi syarat ini, algoritma A* dapat menghindari eksplorasi simpul-simpul yang tidak relevan, mengurangi waktu dan memori yang dibutuhkan.

Terdapat kendala utama dari algoritma A*, yaitu penggunaan memori yang besar karena algoritma ini perlu menyimpan semua simpul yang telah dihasilkan. Untuk mengatasi hal ini, terdapat varian A* seperti *Iterative-Deepening A** (IDA*) yang mengurangi kebutuhan memori tanpa mengorbankan optimalitas solusi, dengan biaya eksekusi yang sedikit lebih tinggi.

Graf pada gambar 2.4 dapat diselesaikan menggunakan A* dengan langkah-langkah berikut.

1. Inisialisasi

- Tentukan nilai heuristik setiap titik $h(n)$.
 - $h(A) = 7$
 - $h(B) = 6$
 - $h(C) = 2$
 - $h(D) = 4$
 - $h(E) = 0$
- Buat 2 buah list untuk menyimpan titik.
 - *Open List*, untuk menyimpan titik yang akan diperiksa.
 - *Closed List*, untuk menyimpan titik yang sudah diperiksa.
- Hitung total jarak untuk titik awal ($f(A)$).
 - $f(A) = 0 + 7 = 7$
- Update list.
 - *Open List*: {A}
 - *Closed List*: {}

2. Iterasi 1

- Pilih titik A, karena hanya ada titik A saja pada *Open List*
- Periksa titik-titik tetangganya.
 - $f(B) = 10 + 6 = 16$
 - $f(C) = 3 + 2 = 5$
- Update list.
 - *Open List*: {B, C}
 - *Closed List*: {A}

3. Iterasi 2

- Pilih titik C, karena memiliki total jarak terkecil pada *Open List*
- Periksa titik-titik tetangganya.
 - $f(B) = 4 + 6 = 10$
 - $f(D) = 7 + 4 = 11$
 - $f(E) = 5 + 0 = 5$
- Update list.
 - *Open List*: {B, D, E}
 - *Closed List*: {A, C}

4. Iterasi 3

- Pilih titik E, karena memiliki total jarak terkecil pada *Open List*
- Iterasi selesai karena E adalah titik tujuan.
- *Closed List*: {A, C, E}

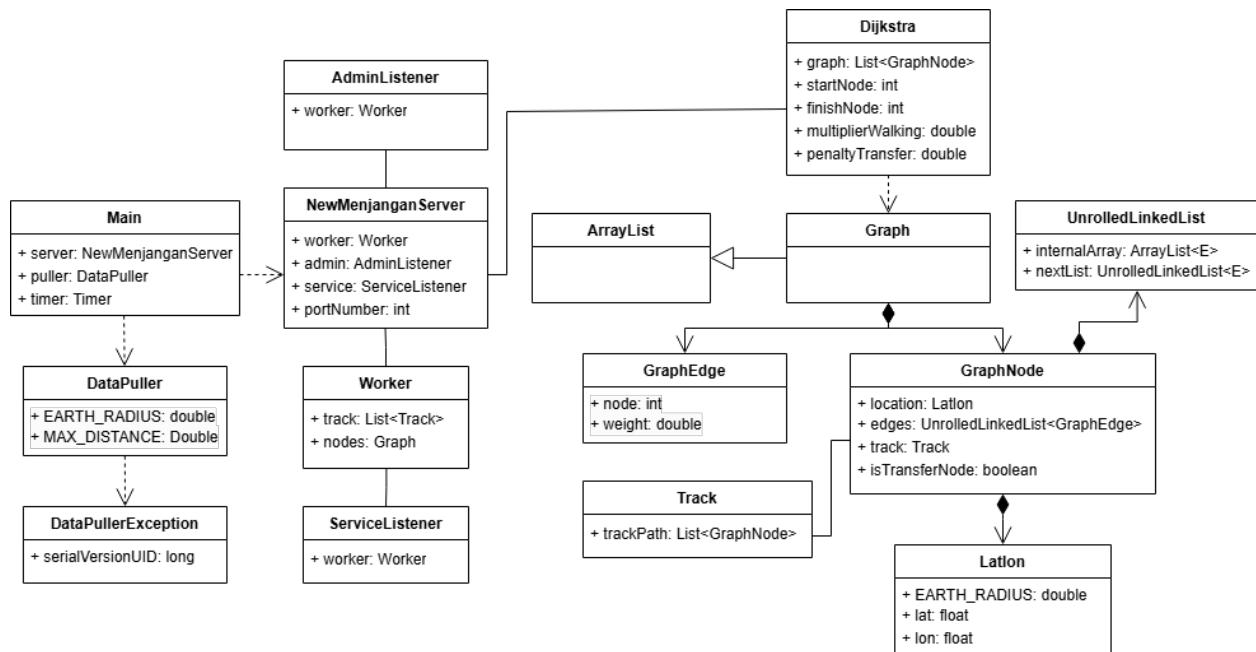
BAB 3

ANALISIS

Bab ini akan membahas analisis-analisis yang dilakukan dalam pengembangan perangkat lunak KIRI. Pada bab ini akan terdapat dua subbab utama, yang pertama, yaitu analisis sistem kini yang akan membahas kelas-kelas yang terdapat pada sistem backend KIRI bernama NewMenjangan. Kemudian yang kedua, yaitu analisis sistem usulan yang akan membahas usulan yang akan dilakukan dalam pengembangan KIRI, seperti implementasi *strategy pattern* serta implementasi algoritma Floyd-Warshall dan A-Star.

3.1 Analisis Sistem Kini

Analisis akan dilakukan terhadap sistem backend dari aplikasi KIRI yang bernama NewMenjangan. Sistem ini bertanggung jawab untuk menangani berbagai fungsi backend yang mendukung layanan utama KIRI, termasuk pengolahan data, komunikasi dengan komponen lain, dan pengelolaan algoritma terkait penelusuran rute. Gambar 3.1, merupakan struktur kelas saat ini dari aplikasi KIRI yang digambarkan dalam diagram kelas. Saat ini, algoritma yang diterapkan adalah algoritma Dijkstra, yang digunakan untuk menemukan jalur terpendek dalam graf berbobot. Analisis ini mencakup tinjauan menyeluruh terhadap struktur kelas NewMenjangan dan juga pemodelan graf pada aplikasi KIRI.



Gambar 3.1: Struktur Kelas NewMenjangan

3.1.1 Analisis Kelas

Pada bagian ini, akan dilakukan analisis terhadap struktur kelas yang membentuk sistem backend NewMenjangan. Analisis ini bertujuan untuk memahami peran dan hubungan antar kelas dalam sistem, termasuk bagaimana kelas-kelas tersebut berinteraksi untuk menangani pencarian rute menggunakan algoritma Dijkstra. Setiap kelas dalam diagram pada Gambar 3.1 akan dianalisis berdasarkan fungsinya dalam sistem.

Main.java

Kelas ini berfungsi sebagai pusat kendali dari backend KIRI. Melalui kelas ini, server bisa dijalankan, diperiksa statusnya, dimatikan, dan juga mengolah data. Pada kelas ini, terdapat 5 konstanta dan 5 atribut. Selain itu, terdapat *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**
 - TRACKS_CONF, MYSQL_PROPERTIES, dan MJNSERVE_PROPERTIES
Konstanta-konstanta tersebut digunakan untuk mengarahkan pada file konfigurasi yang diperlukan.
 - LOGGING_PROPERTIES dan NEWMJNSERVE_LOG
Konstanta-konstanta tersebut digunakan untuk mengatur lokasi konfigurasi logging dan file log.
- **Atribut**
 - server, puller, dan timer
Atribut-atribut tersebut digunakan untuk mengelola server, menarik data, dan menjalankan tugas terjadwal.
 - portNumber
Atribut ini berfungsi untuk menetapkan *port default* yang digunakan oleh server.
 - homeDirectory
Atribut ini digunakan untuk menjadi direktori utama yang diambil dari variabel lingkungan NEWMJNSERVE_HOME, yang diperlukan agar aplikasi berjalan.
- **Method**
 - main(String[] args)
Method ini dirancang untuk memproses argumen yang diterima guna memeriksa status server atau menghentikannya. Ketika argumen -c diberikan, fungsi sendCheckStatus akan dipanggil untuk memastikan bahwa server sedang berjalan. Sebaliknya, jika argumen -s diberikan, fungsi sendShutdown akan bertugas mematikan server. Sebelum proses lebih lanjut dilakukan, program memeriksa apakah variabel lingkungan NEWMJNSERVE_HOME telah disetel. Jika tidak, aplikasi akan segera dihentikan. Setelah inisialisasi konfigurasi *logging* selesai, fungsi pullData dijalankan untuk menarik data yang diperlukan. Server kemudian dimulai melalui pemanggilan metode server.start(), dan sebuah ShutdownHook disertakan untuk memastikan penghentian server dilakukan dengan aman.
 - sendCheckStatus(int portNumber) dan sendShutdown(int portNumber)
Keduanya menggunakan koneksi HTTP untuk mengirim permintaan ke server. *Method* sendCheckStatus berfungsi untuk mengecek status server, sementara *method* sendShutdown mengirim permintaan untuk mematikan server.
 - pullData()
Method ini bertugas untuk menarik data dari sumber SQL dan sumber eksternal lainnya. Jika terjadi perubahan pada file konfigurasi tracks.conf, data yang ada akan ditimpak dengan data baru yang diperbarui. Selain itu, proses pembaruan ini akan dicatat dalam log untuk pemantauan perubahan data yang terjadi. *Method* akan mengembalikan nilai true apabila penarikan data berhasil dan false apabila gagal.

- `fileEquals(File file1, File file2)`

Method ini dirancang untuk membandingkan dua file secara biner untuk menentukan kesamaan di antara keduanya. Jika kedua file memiliki panjang yang berbeda, maka file tersebut secara otomatis dianggap tidak identik. Selain itu, jika ditemukan perbedaan byte pada posisi tertentu selama proses pembandingan, posisi perbedaan tersebut akan dicatat dalam log. *Method* akan mengembalikan nilai `true` apabila file *identical* dan `false` apabila tidak.

AdminListener

Kelas ini berfungsi sebagai *handler* HTTP khusus yang menerima perintah-perintah administratif untuk mengelola server *backend* KIRI. Kelas ini memungkinkan aplikasi *backend* menerima dan menjalankan perintah administrasi dari localhost melalui HTTP. Pada kelas ini, terdapat 1 atribut diinisialisasi dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**

- `worker`

Variabel ini bertipe Worker yang merupakan sebuah kelas. `worker` ini diperlukan untuk menjalankan perintah-perintah tertentu, seperti `tracksinfo`.

- **Method**

- `handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)`

Method ini mengimplementasikan penanganan permintaan HTTP dengan memanfaatkan kelas `AbstractHandler` dari *library* Jetty. Ketika sebuah permintaan HTTP diterima, metode ini akan melakukan beberapa langkah. Pertama, sumber permintaan diperiksa untuk memastikan bahwa hanya permintaan dari localhost yang diterima. Selanjutnya, metode ini mengurai parameter query string dari URL untuk mengidentifikasi perintah yang diminta. Dengan pendekatan ini, setiap permintaan dapat diproses sesuai dengan parameter yang dikirimkan.

Method ini mendukung berbagai jenis perintah yang dapat diterima melalui permintaan HTTP. Salah satu perintah adalah `forceShutdown`, yang berfungsi untuk menghentikan server setelah jeda satu detik dengan menjalankan `System.exit(0)` dalam sebuah thread baru. Perintah lain, yaitu `tracksinfo`, akan mengembalikan informasi mengenai jalur jika worker telah diinisialisasi, menggunakan metode `worker.printTracksInfo()`. Selain itu, perintah ping akan mengembalikan string "pong" untuk memverifikasi bahwa server sedang aktif. Apabila perintah yang diterima tidak valid atau tidak dikenali, metode ini akan mengembalikan status dan pesan kesalahan yang sesuai.

Pengaturan status dan pesan respons dilakukan berdasarkan hasil dari setiap perintah yang diproses. Jika perintah berhasil dijalankan, status `HttpStatus.OK_200` akan dikembalikan. Jika permintaan berasal dari alamat selain localhost, status yang dikembalikan adalah `HttpStatus.FORBIDDEN_403`. Permintaan yang tidak mencantumkan perintah akan menerima status `HttpStatus.BAD_REQUEST_400`, sedangkan jika worker belum siap, status `HttpStatus.SERVICE_UNAVAILABLE_503` akan diberikan.

NewMenjanganServer

Kelas ini berfungsi untuk menginisialisasi dan menjalankan server HTTP yang mendengarkan permintaan pada backend KIRI. Server ini menggunakan *library* Jetty untuk menangani permintaan HTTP. Pada kelas ini, terdapat 1 konstanta dan 6 atribut. Selain itu, terdapat sebuah konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- `DEFAULT_PORT_NUMBER`

Merupakan nomor port *default* yang digunakan jika tidak ada port yang ditentukan.

- **Atribut**

- **worker**

Merupakan instance dari kelas Worker.

- **admin**

Merupakan instance dari kelas AdminListener.

- **service**

Merupakan instance dari kelas ServiceListener.

- **httpServer**

Merupakan server Jetty yang akan mendengarkan permintaan HTTP.

- **portNumber**

Berfungsi untuk menyimpan port yang digunakan server.

- **homeDirectory**

Berfungsi untuk menyimpan direktori *home* yang digunakan oleh server.

- **Konstruktor**

- **NewMenjanganServer(int portNumber, String homeDirectory)**

Bertujuan untuk menginisialisasi komponen-komponen utama server. Pada tahap awal, sebuah objek **worker** dibuat dengan menerima **homeDirectory** sebagai parameter, sehingga memungkinkan **worker** mengakses file yang dibutuhkan. Selanjutnya, objek **admin** dan **service** diinisialisasi untuk menangani permintaan. Selain itu, sebuah *instance* **httpServer** dibuat dan dikonfigurasi agar dapat mendengarkan pada *port* yang telah ditentukan dan juga **AbstractHandler** ditambahkan ke **httpServer** untuk mengarahkan permintaan HTTP ke *method* yang sesuai.

- **Method**

- **clone()**

Method ini bertujuan untuk membuat salinan baru dari objek **NewMenjanganServer** dengan mempertahankan nilai yang sama untuk variabel **portNumber** dan **homeDirectory**. Dalam kasus di mana proses *cloning* gagal, metode ini akan mencatat pesan kesalahan ke dalam *log* global untuk memastikan bahwa kegagalan tersebut tercatat.

- **start()** dan **stop()** Kedua metode ini berfungsi untuk mengelola server HTTP. Metode **start** digunakan untuk menjalankan server sehingga dapat mulai mendengarkan dan memproses permintaan yang masuk. Sebaliknya, metode **stop** bertugas menghentikan server HTTP, memastikan bahwa semua aktivitas server dihentikan dengan aman.

ServiceListener

Kelas ini bertanggung jawab untuk menangani permintaan layanan pada server KIRI. Class ini menerima permintaan HTTP untuk mencari rute dan transportasi terdekat berdasarkan parameter yang diberikan. Pada kelas ini, terdapat 6 konstanta dan 1 atribut. Selain itu, terdapat sebuah konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- **PARAMETER_START, PARAMETER_FINISH, PARAMETER_MAXIMUM_WALKING, PARAMETER_WALKING_MULTIPLIER,PARAMETER_PENALTY_TRANSFER, dan PARAMETER_TRACKTYPEID_BLACKLIST**

Semua konstanta tersebut digunakan untuk menentukan parameter permintaan.

- **Atribut**

- **worker**

Merupakan *instance* dari *class Worker*, yang bertanggung jawab untuk menemukan rute dan transportasi

- **Method**

- `handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)`

Method ini bertanggung jawab untuk menangani permintaan HTTP dan menghasilkan respons yang sesuai. Dalam prosesnya, variabel `query` digunakan untuk menyimpan string parameter permintaan, sementara `params` adalah objek Map yang berisi parameter permintaan yang telah diurai menggunakan `method parseQuery(query)`. Untuk menentukan hasil yang akan dikirimkan, *method* ini menggunakan variabel `responseText` untuk menyimpan teks respons dan `statusCode` untuk menyimpan status HTTP yang akan dikembalikan kepada klien.

- `parseQuery(String query)`

Method ini bertugas untuk memproses *string query* dan mengonversinya menjadi sebuah objek Map yang berisi pasangan kunci dan nilai. Proses dimulai dengan memecah *query* berdasarkan karakter & untuk mendapatkan setiap pasangan kunci dan nilai secara terpisah. Selanjutnya, setiap pasangan dipecah lebih lanjut berdasarkan karakter = untuk memisahkan kunci dari nilainya. Jika *query* yang diterima bernilai *null*, *method* ini akan melemparkan `NullPointerException` sebagai penanganan kesalahan.

Worker

Kelas ini bertanggung jawab untuk menangani permintaan routing (pencarian rute) menggunakan algoritma pencarian jalur terpendek. Fungsinya adalah untuk memproses permintaan rute berdasarkan data graf, dengan mempertimbangkan parameter jarak berjalan kaki, penalti transfer, dan lainnya. Pada kelas ini, terdapat 8 atribut. Selain itu, terdapat konstruktor dan method - method diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**

- `globalMaximumWalkingDistance`

Merepresentasikan jarak maksimal untuk berjalan kaki.

- `global_maximum_transfer_distance`

Merepresentasikan jarak maksimal untuk *transfer node*.

- `globalMultiplierWalking`

Berfungsi sebagai faktor pengali jarak berjalan kaki.

- `globalPenaltyTransfer`

Merepresentasikan nilai penalti untuk *transfer node*.

- `number0fRequests`

Merepresentasikan jumlah permintaan yang diproses.

- `totalProcessTime`

Merepresentasikan total waktu proses (dalam milidetik).

- `tracks`

Merepresentasikan daftar rute (jalur transportasi) yang tersedia.

- `nodes`

Representasi graf dari seluruh *node*.

- **Konstruktor**

- `public Worker(String homeDirectory)`

Konstruktor ini membaca file konfigurasi utama yang disebut `MJNSERVE_PROPERTIES` untuk memuat pengaturan yang dibutuhkan. Selanjutnya, data graf jalur diambil dari file konfigurasi tambahan, yaitu `TRACKS_CONF`, untuk membangun struktur jalur yang akan digunakan. Setelah itu, *method* `linkAngkots()` dijalankan untuk menghubungkan *node-node* angkot, memastikan keterhubungan jalur transportasi dalam graf. Terakhir, metode `cleanUpMemory()` dipanggil untuk membersihkan memori sementara, sehingga efisiensi dan stabilitas sistem tetap terjaga.

- **Method**

- `cleanUpMemory()`

Method ini berfungsi untuk membersihkan memori yang digunakan selama proses komputasi.

- `readConfiguration(String filename)`

Method ini berfungsi untuk membaca konfigurasi dari file properti dan menyimpan nilai ke dalam variabel global.

- `printTracksInfo()`

Method ini berfungsi untuk membuat ringkasan informasi tentang rute (*track*) dan *node* yang dimuat dalam aplikasi.

- `findRoute(LatLon start, LatLon finish, Double customMaximumWalkingDistance, Double customMultiplierWalking, Double customPenaltyTransfer, Set<String> trackTypeIdBlacklist)`

Method ini dirancang untuk membangun graf virtual sebagai bagian dari proses pencarian rute. Proses dimulai dengan menambahkan *node start* dan *end* ke dalam graf, yang mewakili titik awal dan tujuan perjalanan. Setelah itu, *node-node* dalam graf dihubungkan menggunakan jarak berjalan kaki untuk mencerminkan kemungkinan pergerakan pejalan kaki. Selanjutnya, algoritma Dijkstra dijalankan untuk menghitung dan menemukan rute terpendek antara *node start* dan *end*. Sebagai hasil akhirnya, langkah-langkah rute yang ditemukan disusun dalam format protokol Kalapa-Dago untuk memberikan panduan perjalanan yang terstruktur dan dapat diinterpretasikan dengan mudah.

- `resetStatistics()`

Method ini bertujuan untuk mereset statistik pemrosesan server. Dalam prosesnya, jumlah permintaan (`numberOfRequests`) diatur ulang menjadi nol untuk menghapus data historis mengenai jumlah permintaan yang telah diterima. Selain itu, waktu pemrosesan total (`totalProcessTime`) juga diatur ulang menjadi nol untuk menghapus catatan akumulasi waktu yang digunakan dalam memproses permintaan.

- `getNumberOfRequests()`

Method ini mengembalikan jumlah permintaan yang telah diproses sejak statistik terakhir diatur ulang.

- `getTotalProcessTime()`

Method ini mengembalikan total waktu pemrosesan semua permintaan dalam detik.

- `readGraph(String filename)`

Method ini dirancang untuk membangun graf dengan membaca data dari file yang berisi informasi lintasan, node, dan koneksi antar node. Proses dimulai dengan membaca file untuk mengambil detail lintasan, node, serta hubungan koneksi di antara keduanya. Graf kemudian dibuat untuk proses pencarian rute. *Method* ini mengembalikan nilai `true` apabila berhasil dan `false` apabila gagal.

- `linkAngkots()`

Method membangun *K-D Tree* yang digunakan untuk mempermudah pencarian node terdekat. Metode ini juga menghubungkan node transfer yang berada dalam batas jarak transfer maksimum, sehingga memastikan konektivitas antar node sesuai dengan aturan jarak yang telah ditentukan.

- `toString()`

Method ini mengembalikan representasi string dari semua *track* yang dimuat.

- `findNearbyTransports(LatLon start, Double customMaximumWalkingDistance)`

Method ini bertujuan untuk menemukan transportasi terdekat dari lokasi tertentu dengan menghitung jarak dari lokasi awal ke setiap *track* dalam graf. Setelah semua jarak dihitung, *method* ini akan menentukan lintasan dengan jarak minimum yang masih berada dalam batas yang dapat dijangkau dengan berjalan kaki. *Method* ini mengembalikan informasi dari transportasi yang ditemukan dalam bentuk *string*.

DataPuller

Kelas ini bertanggung jawab untuk mengambil data jalur dari basis data dan memprosesnya dalam bentuk yang diinginkan. Kelas ini menggunakan JDBC untuk koneksi ke basis data MySQL dan mengubah data jalur menjadi koordinat. Selain itu, kelas ini menambahkan titik-titik virtual untuk memenuhi jarak maksimum tertentu antar titik. Pada kelas ini, terdapat 2 konstanta serta *method-method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- `EARTH_RADIUS`

Konstanta tersebut merupakan radius Bumi dalam kilometer dan digunakan untuk menghitung jarak antar titik koordinat.

- `MAX_DISTANCE`

Konstanta tersebut merupakan jarak maksimum antar titik yang diizinkan, jika jarak antar dua titik melebihi nilai ini, titik-titik virtual akan ditambahkan di antaranya.

- **Method**

- `pull(File sqlPropertiesFile, PrintStream output)`

Berfungsi untuk mengambil data dari tabel tracks di basis data, kemudian menuliskan hasil format jalur dalam format yang ditentukan. *Method* ini memuat data dari file properti, terhubung ke basis data, dan melakukan query untuk mengambil data yang diperlukan. Hasil query diolah dan ditulis ke *output*.

- `lineStringToLngLatArray(String wktText)`

Mengubah data koordinat dalam format LINESTRING menjadi array `LngLatAlt`. Ini menghilangkan teks LINESTRING dan tanda kurung, kemudian memecah data menjadi objek koordinat `textttLngLatAlt`. *Method* ini mengembalikan array koordinat, yang berisi array objek `LngLatAlt` yang mewakili koordinat LineString.

- `computeDistance(LngLatAlt p1, LngLatAlt p2)`

Menghitung jarak antara dua titik koordinat. *Method* ini mempertimbangkan kelengkungan bumi dalam perhitungan jaraknya. *Method* ini mengembalikan jarak yang dihitung antara dua titik.

- `formatTrack`

Mengonversi informasi jalur yang diambil dari basis data ke format konfigurasi yang dibutuhkan. Metode ini mengatur titik transit, menambahkan titik virtual, dan menyusun informasi jalur dalam format konfigurasi yang diinginkan.

DataPuller.RouteResult

Merupakan sebuah *static nested class* yang menyimpan hasil akhir dalam format konfigurasi sebagai string `trackInConfFormat`, yang dapat diambil dengan *method* `getTrackInConfFormat()`.

DataPullerException

Kelas ini adalah kelas *custom exception* yang dibuat untuk menangani kesalahan khusus yang terjadi saat pemrosesan data dalam kelas DataPuller. Kelas ini memperluas `RuntimeException`, sehingga DataPullerException adalah *unchecked exception* dan tidak memerlukan penanganan eksplisit dengan blok *try-catch* di tempat pemanggilannya. Pada kelas ini, terdapat sebuah konstanta serta konstruktor yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- `serialVersionUID`

Menyimpan ID *serial*. ID ini memastikan data yang disimpan atau dikirimkan tetap cocok dengan versi kelas yang digunakan saat objek tersebut dibaca kembali. Hal ini penting, terutama jika kelas ini mengalami perubahan struktur, agar versi yang berbeda tetap dapat dikenali atau mencegah kesalahan jika struktur sudah tidak cocok.

- **Konstruktor**

- DataPullerException(String message)

Konstruktor ini menerima pesan kesalahan dalam bentuk String, yang kemudian diteruskan ke konstruktor *superclass RuntimeException* untuk disimpan dan nantinya dapat diambil dengan metode `getMessage()`. Pesan ini bertujuan untuk memberikan informasi yang lebih rinci tentang kesalahan yang terjadi.

LatLon

Kelas ini berfungsi untuk merepresentasikan posisi geografis dengan koordinat lintang (*latitude*) dan bujur (*longitude*). Pada kelas ini terdapat metode untuk menghitung jarak antara dua titik koordinat berdasarkan jarak permukaan bumi. Penggunaannya bisa ditemui pada sistem yang memerlukan perhitungan atau pengelolaan data geografis. Pada kelas ini, terdapat 1 konstanta dan 2 atribut. Selain itu, terdapat konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- `EARTH_RADIUS`

Menyimpan nilai jari-jari Bumi dalam satuan kilometer (6371.0 km). Konstanta ini digunakan dalam perhitungan jarak antara dua titik geografis.

- **Atribut**

- `lat`

Merepresentasikan nilai lintang suatu titik.

- `lon`

Merepresentasikan nilai bujur suatu titik.

- **Konstruktor**

- `LatLon(float lat, float lon)`

Konstruktor ini menerima dua parameter, `lat` (*latitude*) dan `lon` (*longitude*), yang disimpan langsung dalam atribut publik kelas.

- `LatLon(String latlon)`

Konstruktor ini menerima parameter tunggal berupa String dengan format *latitude,longitude*. String ini kemudian dipisah berdasarkan tanda koma (","), lalu nilai-nilai yang diperoleh diubah menjadi float dan disimpan dalam atribut `lat` dan `lon`.

- **Method**

- `toString()`

Method ini mengembalikan representasi string dari objek LatLon, berupa lat lon, yang menyajikan lintang dan bujur dalam format yang sederhana.

- `distanceTo(LatLon target)`

Method ini menghitung jarak antara objek LatLon saat ini dengan objek LatLon lain (target). *Method* ini mengembalikan jarak yang dihitung antara objek LatLon saat ini dan objek LatLon target dalam kilometer.

UnrolledLinkedList

Kelas ini merupakan struktur data khusus berbasis linked list yang berfungsi sebagai implementasi daftar berantai (linked list) yang memanfaatkan ArrayList sebagai penyimpanan internal. Pada kelas ini, terdapat 2 atribut. Selain itu, terdapat konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**

- `internalArray`

Menyimpan elemen-elemen di dalam setiap *node* `UnrolledLinkedList`.

- `nextList`

Referensi ke `UnrolledLinkedList` berikutnya (jika ada) untuk membentuk hubungan antara *node-node* `UnrolledLinkedList`.

- **Konstruktor**

- `UnrolledLinkedList()`

Inisialisasi `UnrolledLinkedList` dengan membuat `internalArray` kosong dan `nextList` sebagai `null`.

- **Method**

- `add(E e)`

Menambahkan elemen *e* ke `internalArray` pada `UnrolledLinkedList` saat ini.

- `iterator()`

Mengembalikan iterator (`FastLinkedListIterator`) yang dapat digunakan untuk menavigasi elemen-elemen di dalam `UnrolledLinkedList`.

- `addAll(UnrolledLinkedList<E> elements)`

Menambahkan semua elemen dari sebuah objek `UnrolledLinkedList<E>` lain (`elements`) ke dalam *list* yang sedang diproses (`this`).

- `size()`

Menghitung total elemen di seluruh `UnrolledLinkedList`, termasuk yang ada di `nextList` dan mengembalikan nilai total elemen tersebut.

- `cleanUpMemory()`

Mengurangi ukuran `internalArray` ke jumlah elemen yang bertujuan untuk mengurangi penggunaan memori.

UnrolledLinkedList.FastLinkedListIterator

Inner class ini merupakan iterator yang digunakan untuk menjelajahi atau mengambil elemen-elemen dalam `UnrolledLinkedList` satu per satu secara berurutan. Dalam *inner class* ini juga terdapat 3 atribut serta beberapa *method* yang diimplementasikan.

- **Atribut**

- `currentList`

Menunjuk pada `UnrolledLinkedList` saat ini yang sedang diiterasi.

- `currentIndex`

Posisi indeks di `internalArray` pada `currentList`.

- `globalIndex`

Posisi indeks global yang melacak elemen secara keseluruhan di seluruh `UnrolledLinkedList`.

- **Method**

- `hasNext()`

Mengecek apakah masih ada elemen berikutnya di dalam `internalArray` atau di `nextList` dan mengembalikan nilai `true` apabila masih ada dan `false` apabila tidak.

- `next()`

Mengembalikan elemen berikutnya dalam iterasi. Jika sudah mencapai akhir `internalArray`, beralih ke `nextList`.

- `hasPrevious()` dan `previous()`

Melempar `UnsupportedOperationException`.

- `nextIndex()` dan `previousIndex()`

Mengembalikan indeks global berikutnya atau sebelumnya.

- `remove()`

Melempar `UnsupportedOperationException`.

- `set(E e)`
Mengganti elemen di `currentIndex` dengan elemen baru `e`.
- `add(E e)`
Menambahkan elemen `e` ke dalam `UnrolledLinkedList`.

Track

Kelas Track merepresentasikan sebuah jalur transportasi umum, yaitu jalur angkot atau Transjakarta. Kelas ini menghubungkan node-node dalam sebuah graf menggunakan jalur yang spesifik untuk transportasi tersebut. Pada kelas ini, terdapat 4 atribut. Selain itu, terdapat konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**
 - `trackTypeId`
Menyimpan jenis jalur transportasi, seperti "angkot" atau "transjakarta".
 - `trackId`
Menyimpan ID spesifik dari jalur transportasi, seperti "kalapaledeng" atau "cicaheumciroyom".
 - `trackPath`
Menyimpan daftar node (`GraphNode`) yang membentuk jalur.
 - `penalty`
Menyimpan nilai penalti yang akan digunakan untuk memengaruhi biaya jalur saat algoritma pencarian jalur dijalankan.
- **Konstruktor**
 - `public Track(String fullyQualifiedTrackId)`
Menginisialisasi sebuah objek `Track` menggunakan ID jalur lengkap dalam format `trackTypeId.trackId`.
- **Method**
 - `getTrackId()`
Mengembalikan ID jalur (`trackId`).
 - `getTrackTypeId()`
Mengembalikan jenis jalur (`trackTypeId`).
 - `getPenalty()`
Mengembalikan nilai penalti.
 - `setPenalty(double penalty)`
Mengatur nilai penalti untuk suatu jalur.
 - `toString()`
Mengembalikan representasi string dari objek `Track`, meliputi jenis jalur, ID jalur, dan jumlah node.
 - `addNode(GraphNode node)`
Menambahkan sebuah node ke dalam jalur (`trackPath`).
 - `getNode(int idx)`
Mengambil dan mengembalikan node berdasarkan indeks dalam `trackPath`.
 - `getSize()`
Mengembalikan jumlah node dalam jalur.

GraphEdge

Kelas ini mewakili sebuah *edge* (sisi) dalam struktur data graf. *Edge* ini digunakan dalam konteks perhitungan jalur atau algoritma graf lainnya. Pada kelas ini, terdapat 2 atribut. Selain itu, terdapat konstruktor dan *method* - *method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**

- **node**

Menunjukkan simpul (*node*) yang dituju oleh *edge* ini. *Edge* ini menghubungkan dua *node* dengan arah tertentu dari satu *node* ke *node* yang lain, dan *node* menunjukkan simpul akhir yang menjadi tujuan.

- **weight**

Menunjukkan bobot dari *edge* ini. Dalam konteks jalur terpendek atau algoritma lainnya, bobot ini bisa berarti jarak, waktu tempuh, atau biaya yang diperlukan untuk bergerak dari *node* asal menuju *node* yang ditunjuk oleh *node*.

- **Konstruktor**

- **GraphEdge(int node, double weight)**

Konstruktor ini mengambil dua parameter, *node* dan *weight*, untuk menginisialisasi sebuah *edge*. Ini berarti bahwa setiap *edge* akan memiliki *node* tujuan dan bobotnya sendiri yang menunjukkan biaya atau jarak menuju *node* tersebut.

- **Method**

- **int getNode()**

Method ini mengembalikan nilai *node* yang dituju oleh *edge* ini yang bertujuan untuk mendapatkan informasi tentang *node* tujuan.

- **double getWeight()**

Method ini mengembalikan bobot dari *edge* saat ini, yang dapat digunakan dalam perhitungan atau penelusuran jalur dalam graf.

GraphNode

Kelas ini berfungsi untuk merepresentasikan sebuah *node* (simpul) dalam sebuah graf. *Node* ini digunakan untuk menyimpan informasi tentang lokasi geografis, koneksi ke *node* lain, dan atribut tertentu yang berhubungan dengan transportasi umum. Kelas ini dapat digunakan untuk membuat struktur graf yang akan mencerminkan rute dan jalur transportasi, sehingga memudahkan pemodelan dalam algoritma pencarian rute. Pada kelas ini, terdapat 4 atribut. Selain itu, terdapat konstruktor dan *method* - *method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Atribut**

- **location**

Atribut ini menyimpan lokasi *node* dalam bentuk koordinat lintang dan bujur. *location* merupakan objek dari *LatLon*, yang menyimpan posisi geografis dalam satuan derajat.

- **edges**

Atribut ini menyimpan daftar dari semua *edge* (sisi) atau jalur keluar dari node ini. Jalur keluar ini mengarah ke node lain, menciptakan koneksi dalam graf. *edges* diimplementasikan menggunakan *UnrolledLinkedList*.

- **track**

Atribut ini berfungsi sebagai referensi balik ke informasi rute (track) dari node. Dengan atribut ini, sistem dapat mengetahui rute atau *track* mana yang terkait dengan node tersebut.

- `isTransferNode`

Atribut ini menunjukkan apakah *node* tersebut adalah *node transfer*. Dalam konteks transportasi umum, sebuah *node transfer* memungkinkan pengguna untuk turun atau naik kendaraan umum dari node tersebut. Jika bernilai *true*, berarti *node* ini adalah *node transfer*.

- **Konstruktor**

- `GraphNode(LatLon location, Track track)`

Konstruktor ini membuat instance baru dari kelas `GraphNode` dengan lokasi (`LatLon`) dan referensi rute (`track`). Saat diinisialisasi, atribut `isTransferNode` diatur ke *false* secara *default*, dan `edges` diinisialisasi sebagai daftar kosong dari objek `GraphEdge`.

- **Method**

- `getEdges()`

Mengembalikan daftar `edges`, yaitu daftar sisi keluar dari *node* ini. Daftar ini dapat digunakan untuk mengetahui semua koneksi dari *node* ke *node* lain dalam graf.

- `push_back(int node, float weight)`

Method ini menambahkan sisi baru ke daftar `edges` dengan memasukkan informasi *node* tujuan dan bobotnya. Bobot (*weight*) mencerminkan jarak atau biaya perjalanan ke *node* lain.

- `link(GraphNode nextNode)`

Method ini menghubungkan *node* ini dengan *node* lain (`nextNode`) dengan menambahkan semua sisi (`edges`) dari *node* berikut ke daftar sisi (`edges`) *node* ini.

- `getLocation()`

Mengembalikan lokasi geografis dari *node* ini dalam bentuk objek `LatLon`

- `getTrack()`

Mengembalikan referensi rute (`track`) yang terkait dengan *node* ini. Ini memungkinkan akses ke informasi rute dari *node*.

- `toString()`

Mengembalikan representasi teks dari *node* ini, yang mencakup informasi lokasi dan status apakah *node* ini adalah *node transfer* atau bukan.

- `setTransferNode(boolean b)`

Mengatur apakah *node* ini merupakan *node transfer* berdasarkan parameter `b`. Jika `b` bernilai *true*, maka *node* akan dianggap sebagai *node transfer*.

- `isTransferNode()`

Mengembalikan nilai `boolean` yang menunjukkan apakah *node* ini adalah *node transfer* (*true*) atau bukan (*false*).

Graph

Kelas ini adalah kelas yang merepresentasikan sebuah graf, yaitu kumpulan dari *node-node* (`GraphNode`). Dengan metode `rangeSearch`, kelas ini dapat mendukung pencarian node dalam radius tertentu, yang berguna dalam pemodelan rute. Kelas ini tidak memiliki atribut selain dari `ArrayList` yang diwarisi, karena kelas ini mewarisi semua fungsi dari `ArrayList` dan berfungsi untuk menyimpan node-node dalam bentuk `GraphNode`. Terdapat 2 konstruktor dan *method - method* diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstruktor**

- `Graph()` Konstruktor ini membuat sebuah objek `Graph` tanpa menentukan kapasitas awal. Dengan kata lain, kapasitas akan ditentukan berdasarkan elemen yang ditambahkan.

- `Graph(int capacity)`

Konstruktor ini membuat objek `Graph` dengan kapasitas awal tertentu, sesuai dengan nilai `capacity` yang diberikan.

- **Method**

- `rangeSearch(LatLon center, double distance)`

Method ini berfungsi untuk mencari semua node yang berada dalam jangkauan atau radius tertentu dari suatu titik pusat. Parameter `center` adalah objek `LatLon` yang menyatakan titik pusat dari area pencarian, dan `distance` adalah radius pencarian dalam satuan kilometer.

Pada implementasi awal, `method` ini menggunakan perulangan sederhana melalui setiap `GraphNode` dalam graf (`for(GraphNode node : this)`). Untuk setiap `node` dalam graf, `method` ini menghitung jarak antara `center` dan lokasi `node` (`node.location`) dengan menggunakan metode `distanceTo` pada objek `LatLon`. Jika jarak antara `center` dan `node` tersebut lebih kecil atau sama dengan `distance`, maka `node` tersebut dianggap berada dalam jangkauan, dan dimasukkan ke dalam `list`. Metode ini mengembalikan `list`, yaitu daftar `GraphNode` yang berada dalam jangkauan dari `center`.

Dijkstra

Kelas ini adalah implementasi dari algoritma Dijkstra yang digunakan untuk mencari jarak terpendek antara dua titik dalam sebuah graf. Algoritma ini bekerja dengan mencari rute dengan bobot terendah atau rute dengan jarak minimum antara node awal dan node tujuan. Pada kelas ini, terdapat 1 konstanta dan 10 atribut. Selain itu, terdapat konstruktor dan `method` - `method` diimplementasikan yang memiliki penjelasan sebagai berikut:

- **Konstanta**

- `DIJKSTRA_NULLNODE`

Nilai konstan `-1` untuk menandakan node yang tidak valid.

- **Atribut**

- `graph`

Daftar `node` dalam graf yang merepresentasikan jalur.

- `startNode` dan `finishNode`

Menyimpan indeks `node` awal dan akhir dari pencarian.

- `nodeInfoLinks`

Array `NodeInfo` yang menyimpan informasi jarak terdekat untuk setiap node.

- `nodesMinHeap`

Array `NodeInfo` yang menyimpan node dalam urutan jarak terpendek untuk mendukung operasi heap dalam algoritma Dijkstra.

- `heapsize`

Ukuran heap, menandakan jumlah node yang ada dalam heap.

- `numOfNodes`

Jumlah total node dalam graf.

- `memorySize`

Ukuran memori yang digunakan.

- `multiplierWalking` dan `penaltyTransfer`

Faktor pengali dan penalti yang digunakan untuk menghitung bobot tambahan pada node yang terkait dengan jalur pejalan kaki atau transfer angkot.

- **Method**

- `runAlgorithm(Set<String> trackTypeIdBlacklist)`

Merupakan `method` utama untuk menjalankan algoritma Dijkstra yang dirancang untuk menemukan jalur terpendek dalam sebuah graf. Langkah pertama adalah menginisialisasi setiap objek `NodeInfo` dengan jarak awal bernilai `POSITIVE_INFINITY` dan menetapkan jarak awal `node` sumber (`startNode`) menjadi `0`. Setelah itu, struktur `nodesMinHeap` diatur ulang menggunakan metode `heapify` untuk mengurutkan `node` berdasarkan jarak terpendek. Proses pencarian dimulai dengan mengambil `node` saat ini (`currentNode`) dari heap, dan pencarian dihentikan jika `node` tersebut adalah `node` tujuan (`finishNode`).

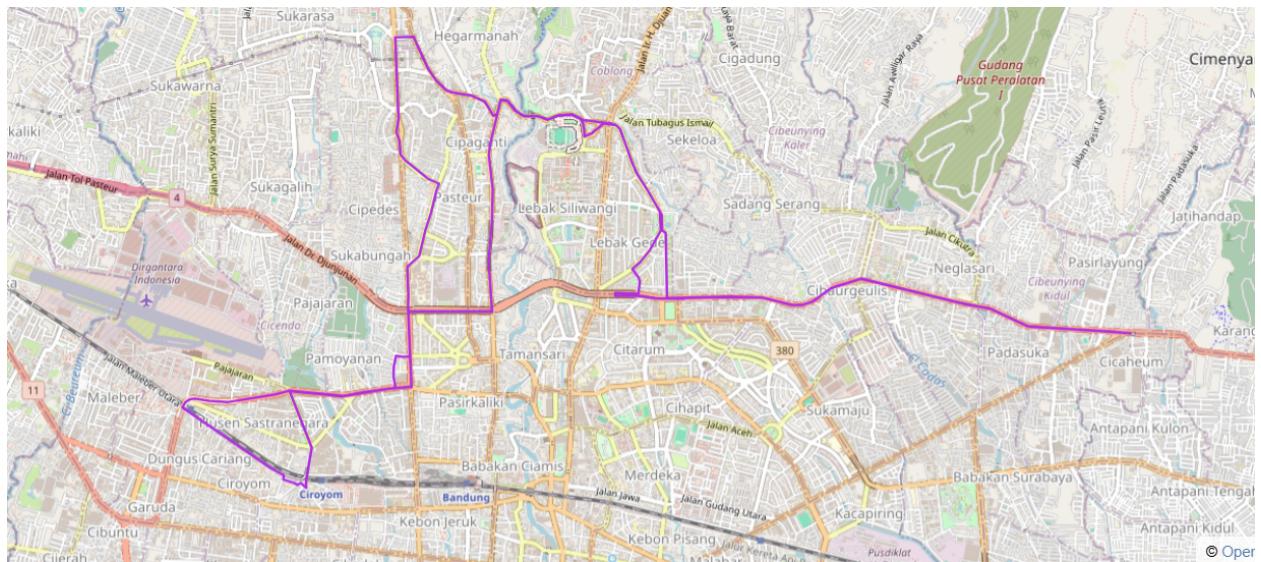
Selama iterasi, untuk setiap objek `GraphEdge` yang terhubung dengan `currentNode`, jarak ke `node` tujuan dihitung menggunakan *method* `calculateWeight`. Jika jarak baru yang dihitung lebih pendek daripada jarak yang tercatat sebelumnya, dan tidak termasuk dalam `trackTypeIdBlacklist`, maka jarak tersebut diperbarui. Selanjutnya, node dengan jarak yang lebih kecil dipindahkan ke posisi yang sesuai dalam heap menggunakan *method* `heapPercolateUp`. Setelah semua langkah selesai, *method* ini akan mengembalikan jarak terpendek ke `node` tujuan. Jika tidak ada jalur yang tersedia, nilai `POSITIVE_INFINITY` akan dikembalikan untuk menandakan kegagalan menemukan jalur.

- `getParent(int node)`
Mengembalikan *parent* (`node` asal) dari `node` yang dimaksud dalam rute terpendek.
- `getDistance(int node)`
Mengembalikan jarak dari `node` awal ke `node` yang diminta.
- `calculateWeight(NodeInfo currentNode, GraphEdge edge)`
Method ini bertugas menghitung bobot perjalanan dari `currentNode` ke `node` tujuan melalui sebuah `edge`. Perhitungan bobot dilakukan berdasarkan beberapa kondisi. Jika salah satu dari `node` tersebut merupakan jalur pejalan kaki, bobot dihitung dengan menerapkan nilai `multiplierWalking` untuk merepresentasikan jarak berjalan kaki. Jika node tersebut melibatkan *transfer* angkot, parameter `penaltyTransfer` akan ditambahkan ke bobot perjalanan. Namun, jika perjalanan tetap berada dalam angkot yang sama, bobot dihitung berdasarkan nilai bobot dari objek `GraphEdge` serta penalti yang berlaku pada jalur yang relevan. *Method* ini mengembalikan nilai Double yang mewakili bobot yang telah dihitung.
- `heapPercolateDown(int index)`
Menjaga agar heap tetap terurut setelah penghapusan node dengan jarak terpendek.
- `heapPercolateUp(int index)`
Memperbarui posisi node dalam heap setelah perubahan jarak.
- `heapDeleteMin()`
Menghapus node dengan jarak terpendek dari heap, dan menyesuaikan urutan heap. *Method* ini mengembalikan variabel `ret`, yang berisi objek `NodeInfo`.
- `getString(int node)`
Mengembalikan string dari `NodeInfo`.

Dijkstra.NodeInfo

Merupakan sebuah *static nested class* yang menyimpan data untuk setiap node, seperti `baseIndex`, `heapIndex`, `distance`, dan `parent`. Selain itu, diimplementasikan juga method `toString()` yang mengembalikan nilai string dari data-data tersebut yang telah diformat.

3.1.2 Pemodelan Graf



Gambar 3.2: Rute Ciroyom – Cicaheum

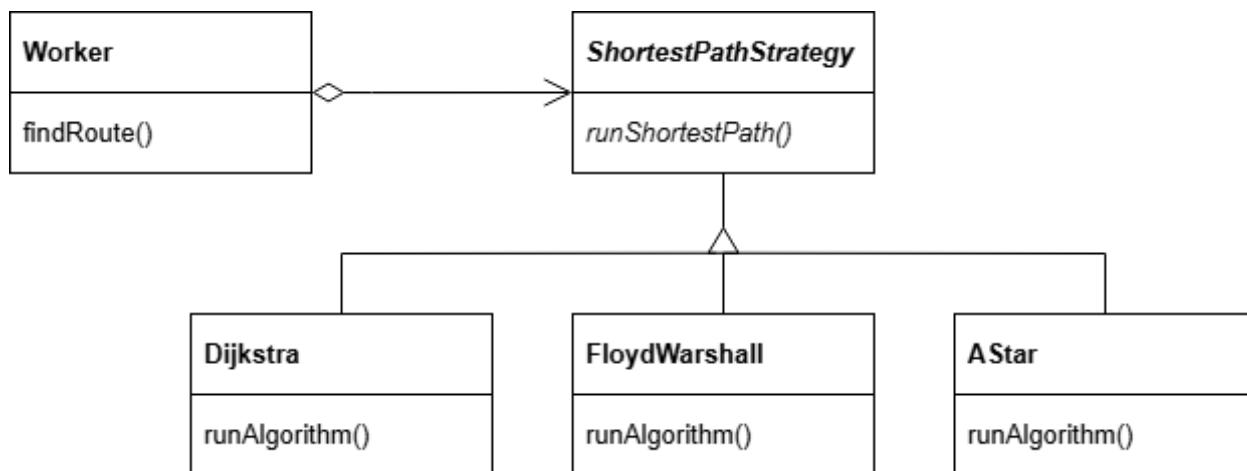
Pada bagian ini, akan dilakukan analisis mengenai bagaimana rute-rute yang terdapat dalam tabel "tracks" pada database dapat dimodelkan menjadi sebuah graf oleh NewMenjangan. Gambar 3.2 merupakan visualisasi GIS (*Geographic Information System*) dari rute angkot cicaheum-ciroyom. Pada saat NewMenjangan dijalankan data dari tabel "tracks" akan diambil menggunakan method `pull()` pada kelas DataPuller 3.1.1 dalam format LineString 2.3.1. Kemudian format data diubah menjadi bentuk array `LngLatAlt` (*longitude* dan *latitude*) menggunakan method `lineStringToLngLatArray` yang juga terdapat pada kelas DataPuller 3.1.1. Dari Titik-titik yang terdapat pada array `LngLatAlt` akan dimodelkan menjadi graf dengan menjadikan setiap titik menjadi sebuah node dan akan dihubungkan dengan edge antara nodenya, selain itu juga apabila ada dua titik dari rute angkot berbeda atau disebut juga *transferNode* yang jaraknya dibawah dari konstanta yang telah ditentukan, maka akan dibuatkan juga sebuah edge untuk menghubungkannya. Pemodelan graf tersebut menggunakan method-method pada kelas Worker 3.1.1 yang juga memanfaatkan method-method pada kelas Graph 3.1.1, GraphNode 3.1.1, LatLon 3.1.1, dan Track 3.1.1.

3.2 Analisis Sistem Usulan

Analisis ini dilakukan terhadap sistem usulan yang bertujuan untuk meningkatkan fleksibilitas dan efisiensi dalam proses pencarian rute terpendek pada perangkat lunak KIRI. Sistem yang diusulkan mengadopsi strategy pattern sebagai pola desain untuk memisahkan logika implementasi algoritma shortest path, sehingga memungkinkan penggantian atau penambahan algoritma dengan lebih mudah. Selain algoritma Dijkstra yang telah diterapkan pada sistem saat ini, sistem usulan juga mengimplementasikan dua algoritma tambahan, yaitu Floyd-Warshall dan A*. Penambahan kedua algoritma ini bertujuan untuk memperluas cakupan kebutuhan kasus penggunaan yang beragam, di mana Floyd-Warshall cocok untuk penghitungan semua pasangan titik, sedangkan A* menawarkan efisiensi untuk pencarian rute dengan heuristik tertentu. Dengan demikian, sistem usulan diharapkan dapat mengatasi keterbatasan sistem yang ada, yang saat ini hanya menggunakan satu algoritma dan belum mengadopsi pola desain modular seperti strategy pattern.

3.2.1 Implementasi Strategy Pattern

Implementasi strategy pattern pada NewMenjangan bertujuan untuk memberikan KIRI kebebasan untuk memilih dan menentukan strategi yang akan dipilih, dalam kasus ini adalah algoritma *shortest path*. Selain itu, juga memberikan fleksibilitas yang tinggi serta memberikan kemudahan dalam menambahkan atau mengganti algoritma ketika proses pengembangan perangkat lunak, seperti yang telah dijelaskan pada subbab 2.2.



Gambar 3.3: Struktur Strategy Pattern

Gambar 3.3 menunjukkan struktur strategy pattern yang dirancang untuk implementasi yang akan dilakukan dalam tugas akhir ini. Gambar 3.3 merupakan lanjutan dari gambar 2.3 yang hanya menggambarkan struktur strategy pattern secara umum. Gambar 3.3 menjelaskan bagaimana strategy pattern diimplementasikan dengan algoritma Dijkstra, A*, dan Floyd-Warshall sebagai *concrete strategy*. Kemudian, kelas ShortestPathAlgorithm sebagai *Strategy* dan kelas Worker sebagai *Context*.

3.2.2 Implementasi Algoritma A* dan Algoritma Floyd-Warshall

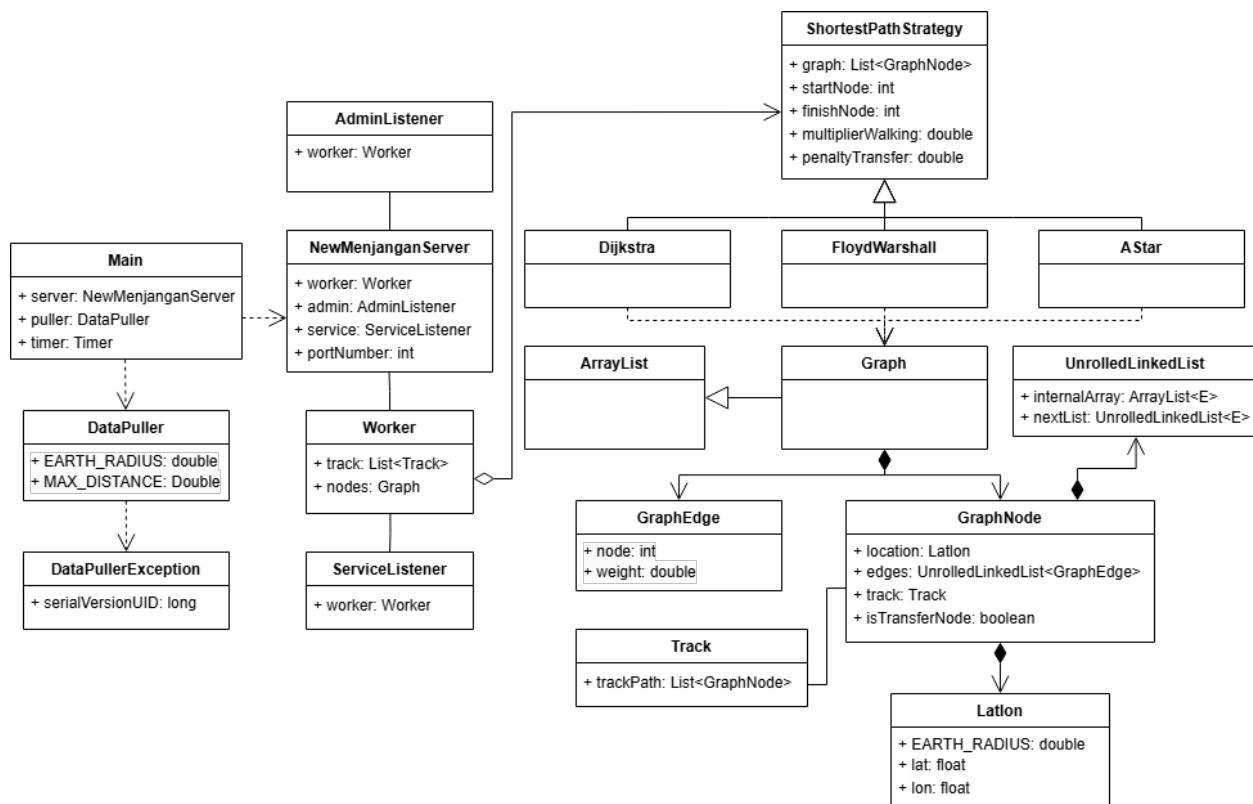
Sistem usulan dirancang untuk meningkatkan performa dan fleksibilitas dalam penentuan rute terpendek dengan mengimplementasikan algoritma A* dan Floyd-Warshall. Pada sistem saat ini, hanya algoritma Dijkstra yang telah diimplementasikan. Algoritma A* dan Floyd-Warshall juga sama seperti algoritma Dijkstra yang termasuk algoritma *shortest path* yang digunakan untuk mencari rute terdekat, seperti yang sudah dijelaskan pada subbab 2.5.3 dan 2.5.2. Perubahan akan dilakukan pada NewMenjangan untuk mengimplementasikan algoritma A* dan Floyd-Warshall, yaitu dengan menambahkan 2 *class* java baru dengan nama Dijkstra.java dan AStar.java.

BAB 4

PERANCANGAN

Bab ini akan membahas mengenai perancangan perangkat lunak berdasarkan hasil analisis pada Bab 3. Bab ini mencakup perancangan kelas, perancangan antar muka, dan perancangan protokol.

4.1 Perancangan Kelas



Gambar 4.1: Perancangan Kelas

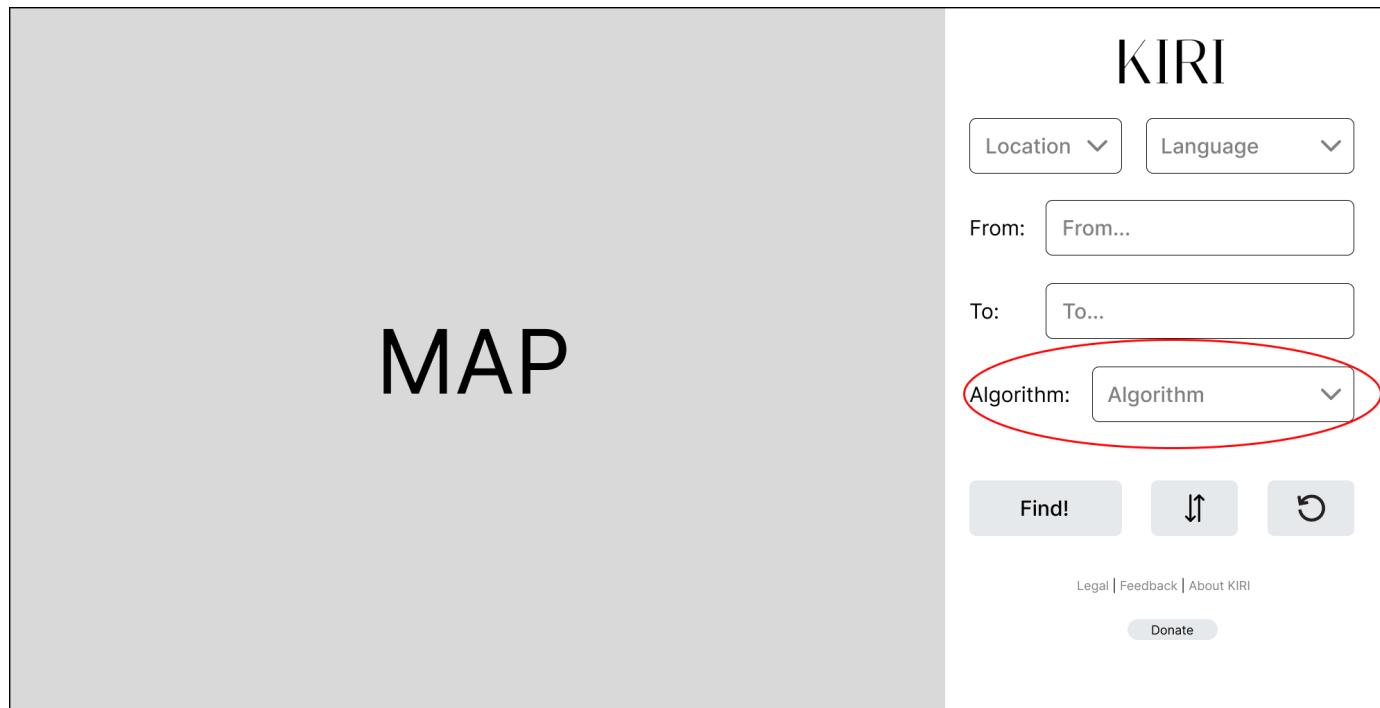
Perancangan kelas dilakukan untuk merepresentasikan struktur perangkat lunak setelah dilakukan pengembangan terhadap peran KIRI. Perubahan ini difokuskan pada penerapan Strategy Pattern guna meningkatkan fleksibilitas sistem dalam menggunakan berbagai algoritma pencarian jalur terpendek. Gambar 4.1 menunjukkan struktur kelas setelah pengembangan dilakukan.

Sebelumnya, algoritma shortest path diimplementasikan secara langsung dalam sistem menggunakan algoritma Dijkstra. Oleh karena itu, dilakukan perubahan desain dengan menerapkan Strategy Pattern dengan dibuatnya kelas abstrak **ShortestPathStrategy** yang menjadi dasar dari semua algoritma shortest path. Kelas-kelas seperti **Dijkstra**, **FloydWarshall**, dan **AStar** kemudian diimplementasikan sebagai turunan dari **ShortestPathStrategy**. Dengan struktur ini, ketiga algoritma tersebut dapat

saling dipertukarkan tanpa memengaruhi bagian lain dari sistem, karena semuanya mengikuti superclass-nya, yaitu ShortestPathStrategy. Strategi ini memudahkan sistem dalam memilih dan menggunakan algoritma pencarian jalur terpendek yang sesuai dengan kebutuhan tertentu. Selain itu, Struktur kelas seperti GraphNode, GraphEdge, dan Graph tetap dipertahankan, namun kini berfungsi mendukung berbagai algoritma.

Kelas ShortestPathStrategy memiliki atribut-atribut yang sebelumnya terdapat pada Dijkstra, yaitu graph, startNode, finishNode, multiplierWalking, dan penaltyTransfer. Atribut-atribut tersebut akan diturunkan dan digunakan oleh seluruh algoritma shortest path. Selain itu, kelas ShortestPathStrategy juga memiliki method-method yang di-override dari kelas-kelas algoritma shortest path. Dengan perancangan ini, perangkat lunak KIRI kini memiliki arsitektur yang lebih modular dan akan lebih mudah apabila akan dikembangkan lebih lanjut, baik dari sisi algoritma maupun fungsionalitas lain.

4.2 Perancangan Antarmuka



Gambar 4.2: Rancangan Antarmuka

Dilakukan sedikit perubahan pada halaman antarmuka dari perangkat lunak KIRI, yaitu dengan menambahkan sebuah *dropdown* baru untuk memilih algoritma yang akan digunakan, seperti pada gambar 4.2. Pada *dropdown* tersebut akan terdiri dari 3 buah algoritma *shortest path* yang sudah diimplementasikan sebelumnya, yaitu Dijkstra, Floyd-Warshall, dan A-Star.

4.3 Perancangan Protokol

Dilakukan sedikit modifikasi pada protokol API untuk memberikan fleksibilitas pengguna dalam memilih algoritma jalur terpendek yang akan digunakan, seperti Dijkstra, A*, dan Floyd-Warshall. Modifikasi dilakukan pada bagian *front-end* (Tirtayasa) dan juga *back-end* (NewMenjangan). Perubahan ini hanya menambah satu parameter baru tanpa mengubah struktur data utama sehingga modifikasi dapat dilakukan tanpa mempengaruhi fungsi sistem lainnya secara signifikan.

Tirtayasa

Pada bagian ini akan dilakukan modifikasi pada protokol API, yaitu akan dilakukan modifikasi pada kode-kode program yang berkaitan dalam pengiriman data ke bagian *back-end*. Berikut merupakan kode-kode program yang akan modifikasi.

- **main.js**

Pada kode ini modifikasi yang akan dilakukan adalah penambahan parameter baru pada *function checkCoordinatesThenRoute* ketika akan memanfaatkan *function findroute* akan ditambahkan sebuah parameter baru yang akan mengirimkan algoritma yang dipilih.

- **protocol.js**

Pada kode ini modifikasi yang akan dilakukan adalah penambahan parameter baru pada saat mendefinisikan *function findroute* yaitu parameter untuk algoritma yang dipilih.

- **Api.php**

Pada kode ini modifikasi yang akan dilakukan adalah penambahan parameter baru pada *function _findroute* yaitu parameter untuk algoritma yang dipilih.

NewMenjangan

Pada bagian ini akan dilakukan modifikasi pada kelas yang akan menerima parameter-parameter yang dikirimkan dari bagian Tirtayasa. Berikut merupakan kelas yang akan modifikasi.

- **ServiceListener.java**

Pada kelas ini modifikasi yang akan dilakukan adalah penambahan parameter baru, yaitu parameter untuk algoritma yang dipilih.

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini akan membahas hasil implementasi dalam pengembangan perangkat lunak KIRI berdasarkan rancangan yang telah dibuat pada Bab 4. Selain itu, akan dibahas juga mengenai pengujian dari perangkat lunak KIRI yang sudah dikembangkan.

5.1 Implementasi

Pada bagian ini akan dijelaskan mengenai lingkungan implementasi perangkat lunak yang telah dikembangkan beserta hasilnya. Bagian ini terbagi menjadi empat subbagian, yaitu lingkungan perangkat keras, lingkungan perangkat keras, hasil implementasi antarmuka, dan hasil implementasi antarmuka. Implementasi perangkat lunak ini didasarkan pada perancangan yang telah dijelaskan pada Bab 4.

5.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam mengimplementasi rancangan perangkat lunak memiliki spesifikasi sebagai berikut:

1. Laptop: ASUS TUF Gaming F15
2. Processor: Intel(R) Core(TM) i7-12700H (20 CPUs), 2.3GHz
3. Random Access Memory (RAM): 16GB
4. Solid State Drive (SSD): 512GB

5.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam mengimplementasi rancangan perangkat lunak memiliki spesifikasi sebagai berikut:

1. Operating System: Windows 11 Home Single Language
2. Bahasa Pemrograman:
 - PHP
 - Java
 - Javascript
3. Framework: CodeIgniter 3.1.13
4. Code Editor: Visual Studio Code 1.99.3
5. Perangkat Lunak Pendukung:
 - XAMPP version 3.3.0
 - phpMyAdmin 5.2.1
 - Apache Ant 1.10.15
 - PHP 8.2.12
 - Java 19.0.2

5.1.3 Penjelasan Kode Program

Pada bagian ini akan dijelaskan kode-kode program atau kelas-kelas yang telah diimplementasikan untuk mendukung pengembangan perangkat lunak KIRI. Kode-kode program tersebut mencakup kode program yang baru saja ditambahkan serta kode program yang sudah ada sejak awal dan dilakukan modifikasi. Berikut merupakan penjelasannya.

Dijkstra.java

Kelas ini berfungsi untuk mengimplementasikan algoritma Dijkstra yang digunakan untuk mencari jalur terpendek dari satu titik ke titik lainnya dalam graf berbobot. Secara keseluruhan kelas Dijkstra masih sama seperti yang dijelaskan di [3.1.1](#).

Pada kelas Dijkstra ini, hanya dilakukan sedikit perubahan dari implementasi aslinya (lihat Kode [A.1](#)). Perubahan yang dilakukan hanya menambahkan pewarisan (extends ShortestPathStrategy) agar kelas Dijkstra dapat menjadi turunan dari kelas abstrak ShortestPathStrategy, yang berfungsi sebagai kerangka umum untuk berbagai algoritma pencarian jalur.

Kode 5.1: Dijkstra.java

```
1 public class Dijkstra extends ShortestPathStrategy
```

Selain itu, beberapa *method* di dalam kelas ini diberi anotasi `@Override` untuk menandakan bahwa metode tersebut mengimplementasikan metode dari kelas induknya.

Kode 5.2: Dijkstra.java

```
1 @Override
2 public double runAlgorithm
3
4 @Override
5 public int getParent
6
7 @Override
8 public double getDistance
```

Perubahan lainnya, yaitu memindahkan atribut-atribut utama seperti `graph`, `startNode`, `finishNode`, `multiplierWalking`, dan `penaltyTransfer` ke kelas induknya sehingga pemanggilan atribut menggunakan `super`.

Kode 5.3: Dijkstra.java

```
1 super(graph, startNode, finishNode, multiplierWalking, penaltyTransfer);
```

AStar.java

Kelas ini adalah implementasi dari algoritma A-Star, yang digunakan untuk mencari jarak terpendek antara dua titik dalam sebuah graf dengan mempertimbangkan heuristic (estimasi jarak ke tujuan). Algoritma ini bekerja dengan mencari rute optimal berdasarkan kombinasi antara jarak yang telah ditempuh (g) dan estimasi jarak yang tersisa ($f = g + h$). Kode program dari AStar.java bisa dilihat pada lampiran [A.2](#) dan berikut merupakan penjelasan dari kode tersebut.

- **Atribut**

- **graph**

Daftar node dalam graf yang merepresentasikan jaringan jalur. Atribut ini merupakan turunan dari kelas induk / *superclass*

- **startNode** dan **finishNode**

Indeks node awal dan akhir dari pencarian. Atribut ini merupakan turunan dari kelas induk / *superclass*

- **nodeInfoLinks**

Array `NodeInfo` yang menyimpan informasi seperti jarak yang telah ditempuh (`g`), estimasi total biaya (`f`), dan node asal (`parent`) untuk setiap node.

- **openSet**

Priority queue yang berisi `NodeInfo` dan mengurutkan node berdasarkan nilai estimasi total biaya (`f`).

- **multiplierWalking** dan **penaltyTransfer**

Faktor pengali dan penalti yang digunakan untuk menghitung bobot tambahan pada node yang terkait dengan jalur pejalan kaki atau perpindahan angkot. Atribut ini merupakan turunan dari kelas induk / *superclass*

- **Method**

- **runAlgorithm(Set<String> trackTypeIdBlacklist)**

Method utama untuk menjalankan algoritma. Langkah pertama adalah mengatur jarak dari node awal (`g`) menjadi 0 dan menghitung estimasi awal (`f`) menggunakan fungsi heuristic. Node awal kemudian dimasukkan ke dalam `openSet`. Selama `openSet` tidak kosong, node dengan nilai `f` terendah diambil sebagai `current`. Jika node ini adalah `finishNode`, maka jarak terpendek (`g`) dikembalikan. Untuk setiap tetangga dari `current`, dihitung biaya baru menuju node tersebut menggunakan `calculateWeight`. Jika biaya baru lebih kecil daripada nilai `g` yang telah tersimpan, nilai-nilai dalam `NodeInfo` diperbarui dan node dimasukkan ke `openSet`. Jika tidak ditemukan jalur ke `finishNode`, nilai `Double.POSITIVE_INFINITY` dikembalikan. Method ini diberi anotasi `@Override` karena method ini mengimplementasikan method dari kelas induk / *superclass*.

- **getParent(int node)**

Mengembalikan parent (node asal) dari node yang dimaksud dalam rute terpendek. Method ini diberi anotasi `@Override` karena method ini mengimplementasikan method dari kelas induk / *superclass*.

- **getDistance(int node)**

Mengembalikan jarak dari node awal ke node yang diminta (nilai `g` dari node tersebut). Method ini diberi anotasi `@Override` karena method ini mengimplementasikan method dari kelas induk / *superclass*.

- **heuristic(int node)**

Fungsi `heuristic` yang mengembalikan estimasi jarak antara node saat ini dan node tujuan menggunakan metode `distanceTo` antar koordinat `LatLon`.

- **calculateWeight(NodeInfo currentNode, GraphEdge edge)**

Method ini menghitung bobot perjalanan dari `currentNode` ke node tujuan melalui sebuah edge. Jika salah satu track merupakan jalur pejalan kaki, maka bobot dihitung berdasarkan `multiplierWalking`. Jika perpindahan kendaraan terjadi, maka bobot ditambah `penaltyTransfer`. Jika tetap berada dalam kendaraan yang sama, bobot dikalikan dengan penalti jalur tersebut. Method ini mengembalikan nilai `double` sebagai bobot perjalanan.

AStar.NodeInfo

Merupakan *static nested class* yang menyimpan informasi-informasi berikut ini:

- **index**
Indeks node
- **g**
Jarak dari node awal ke node ini.
- **f**
Estimasi total jarak ($g + \text{heuristic}$).
- **parent**
Node asal yang menuju ke node ini.

Kelas ini juga mengimplementasikan `Comparable<NodeInfo>` agar dapat digunakan dalam mengurutkan `PriorityQueue` berdasarkan nilai f .

FloydWarshall.java

Kelas ini merupakan implementasi dari algoritma Floyd-Warshall, yaitu algoritma pemrograman dinamis yang digunakan untuk mencari jarak terpendek antar semua pasangan node dalam graf. Algoritma ini sangat efisien untuk graf yang padat dan tidak bergantung pada struktur heap atau antrian prioritas, melainkan menggunakan matriks jarak. Kode program dari `FloydWarshall.java` bisa dilihat pada lampiran [A.3](#) dan berikut merupakan penjelasan dari kode tersebut.

- **Atribut**
 - **graph**
Daftar node dalam graf yang merepresentasikan jaringan jalur. Atribut ini merupakan turunan dari kelas induk / *superclass*.
 - **startNode** dan **finishNode**
Menyimpan indeks node awal dan akhir untuk pencarian rute spesifik. Atribut ini merupakan turunan dari kelas induk / *superclass*
 - **numOfNodes**
Menyimpan jumlah total node dalam graf.
 - **dist**
Matriks dua dimensi `double[][]` yang menyimpan jarak terpendek antar setiap pasangan node.
 - **parent**
Matriks dua dimensi `int[][]` yang menyimpan node asal dari jalur terpendek antara setiap pasangan node.
 - **multiplierWalking** dan **penaltyTransfer**
Digunakan untuk menghitung bobot perjalanan tambahan yang melibatkan jalan kaki atau perpindahan antar angkot. Atribut ini merupakan turunan dari kelas induk / *superclass*
- **Method**
 - **runAlgorithm(Set<String> trackTypeIdBlacklist)**
Method utama untuk menjalankan algoritma. Pertama-tama, semua nilai dalam matriks `dist` diinisialisasi dengan jarak langsung antar node berdasarkan edge yang tersedia, dan mempertimbangkan blacklist dari `trackTypeIdBlacklist`. Nilai `parent[i][j]` diatur sesuai asal hubungan. Kemudian, algoritma berjalan dalam tiga *looping* (k, i, j) untuk mengevaluasi apakah melalui node k bisa memberikan jalur lebih pendek antara i dan j . Jika ya, nilai `dist[i][j]` dan `parent[i][j]` diperbarui. Method ini jarak terpendek dari `startNode` ke `finishNode`. Method ini diberi anotasi `@Override` karena method ini mengimplementasikan method dari kelas induk / *superclass*.

- `calculateWeight(int fromIndex, GraphEdge edge)`
Menghitung bobot perjalanan antara dua node berdasarkan jenis track. Jika salah satu track adalah pejalan kaki, bobot dikalikan dengan `multiplierWalking`. Jika terdapat perpindahan antar kendaraan (track berbeda), penalti `penaltyTransfer` ditambahkan ke bobot. Jika berada dalam kendaraan yang sama, bobot dikalikan penalti jalur.
- `getParent(int node)`
Mengembalikan parent (node asal) dari node yang dimaksud dalam rute terpendek dari `startNode`. Method ini diberi anotasi `@Override` karena method ini mengimplementasikan method dari kelas induk / *superclass*.
- `getDistance(int node)`
Mengembalikan jarak terpendek dari `startNode` ke node yang diminta berdasarkan matriks `dist`. Method ini diberi anotasi `@Override` karena method ini mengimplementasikan method dari kelas induk / *superclass*.

ShortestPathStrategy.java

Kelas ini adalah kelas abstrak yang digunakan sebagai kelas induk untuk berbagai implementasi algoritma *shortest path*, yaitu Dijkstra, A-Star, dan Floyd-Warshall. Kelas ini mengimplementasikan pola desain Strategy Pattern, di mana berbagai algoritma dapat diimplementasikan sebagai subclass dari kelas ini. Tujuannya adalah agar algoritma dapat digunakan secara fleksibel. Kelas ini juga memiliki atribut-atribut yang diturunkan ke subclassnya dan juga method-method yang diimplementasikan pada subclassnya. Kode program kelas ini bisa dilihat pada lampiran [A.4](#).

Worker.java

Kelas ini bertanggung jawab untuk menangani permintaan routing (pencarian rute) menggunakan algoritma pencarian jalur terpendek. Secara keseluruhan kelas Worker masih sama seperti yang dijelaskan di [3.1.1](#).

Perubahan yang dilakukan pada Worker hanya sedikit dari implementasi aslinya (lihat Kode [A.5](#)). Salah satu perubahan yang dilakukan adalah pada method `findRoute` ketika melakukan pemanggilan method pada kelas algoritma *shortest path*.

Kode 5.4: Worker.java

```

1 ShorestPathStrategy strategy;
2
3 if ("floydwarshall".equals(algorithm)) {
4     strategy = new FloydWarshall(vNodes, startNode, endNode,
5         customMultiplierWalking, customPenaltyTransfer);
6 } else if ("astar".equals(algorithm)) {
7     strategy = new AStar(vNodes, startNode, endNode,
8         customMultiplierWalking, customPenaltyTransfer);
9 } else {
10     strategy = new Dijkstra(vNodes, startNode, endNode,
11         customMultiplierWalking, customPenaltyTransfer);
12 }
13
14 strategy.runAlgorithm(trackTypeIdBlacklist);

```

Pada saat ini kelas yang dipanggil merupakan kelas induk, yaitu kelas `ShorestPathStrategy` yang pada sebelumnya langsung memanggil kelas Dijkstra. Selain itu ada kondisi untuk mendukung pemilihan algoritma yang akan digunakan dalam perhitungan. Selain itu ditambahkan juga baris kode untuk mencatat waktu proses dari perhitungan.

ServiceListener.java

Kelas ini bertanggung jawab untuk menangani permintaan layanan pada server KIRI. Kelas ini menerima permintaan HTTP untuk mencari rute dan transportasi terdekat berdasarkan parameter yang diberikan seperti yang sudah dijelaskan di [3.1.1](#). Pada kelas ini perubahan yang dilakukan hanya menambah satu parameter saja, yaitu parameter PARAMETER_ALGO yang diperlukan untuk pemilihan algoritma yang akan digunakan dalam perhitungan. Kode program kelas ini bisa dilihat pada lampiran [A.6](#).

main.php

Kelas ini bertanggung jawab sebagai antarmuka dari perangkat lunak KIRI. Perubahan yang dilakukan pada kelas ini yaitu dengan menambahkan baris kode untuk sebuah *dropdown* baru untuk memilih algoritma. Berikut merupakan baris kode yang ditambahkan.

Kode 5.5: main.php

```

1 <div class="row up-0">
2   <div class="col-lg-12">
3     <select id="selectAlgo" class="form-control hidden"></select>
4   </div>
5 </div>
6 <div class="row up-1">
7   <div class="col-3">
8     <span for="algoselect" class="align-middle">Algorithm:</span>
9   </div>
10  <div class="col-9">
11    <select id="algoselect" class="form-control">
12      <option value="dijkstra">Dijkstra</option>
13      <option value="floydwarshall">Floyd-Warshall</option>
14      <option value="astar">A*</option>
15    </select>
16  </div>
17 </div>
```

Untuk keseluruhan kode program dari kelas main.php bisa dilihat pada lampiran [A.7](#).

main.js

main.js merupakan sebuah file skrip JavaScript yang menangani berbagai interaksi antara pengguna dan aplikasi di sisi frontend, seperti mengambil input dari pengguna, menanggapi klik tombol, serta menampilkan hasil pencarian atau rute yang diterima dari server. Perubahan yang dilakukan pada skrip ini terdapat pada *function checkCoordinatesThenRoute* yang dimana ditambahkan satu parameter baru yaitu parameter untuk memilih algoritma yang diambil dari antarmuka. Kode program dari skrip main.js dapat dilihat pada lampiran [A.8](#)

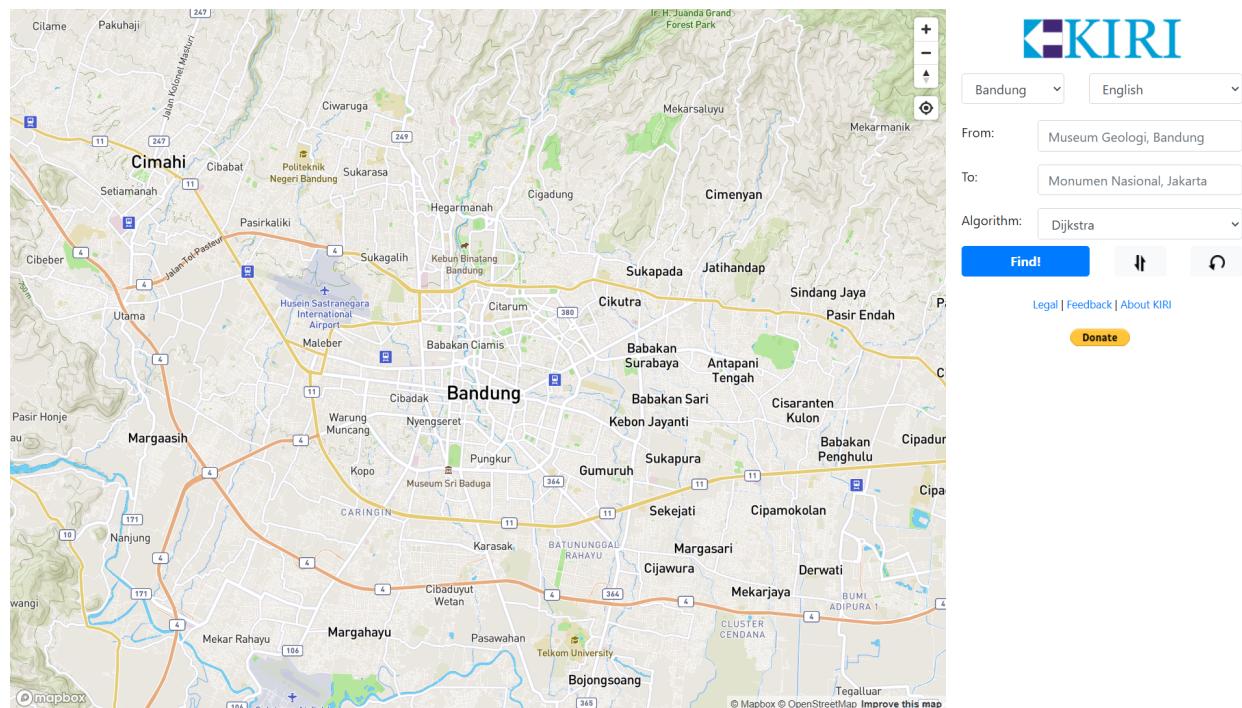
protocol.js

Kelas ini merupakan sebuah kelas JavaScript yang bertugas untuk mengirimkan permintaan data dari sisi pengguna (*frontend*) ke server (*backend*) melalui metode AJAX. Kelas ini memfasilitasi komunikasi antara antarmuka pengguna dengan API. Perubahan dilakukan pada saat mendefinisikan fungsi bernama *findroute* yaitu menambahkan sebuah parameter baru yaitu parameter *algo*. Kode program dari kelas ini dapat dilihat pada lampiran [A.9](#)

Api.php

Kelas ini adalah sebuah kelas di sisi server (*backend*), yang berperan penting dalam memproses dan menangani setiap permintaan yang dikirimkan oleh pengguna melalui antarmuka aplikasi. Kelas ini akan menerima parameter permintaan, mengeksekusi logika tertentu berdasarkan jenis permintaan tersebut, dan mengembalikan hasil dalam bentuk data JSON. Perubahan yang dilakukan yaitu penambahan parameter baru pada *function _findroute* dengan menambahkan parameter *algo*. Kode program dari kelas ini dapat dilihat pada lampiran A.10

5.1.4 Implementasi Antarmuka



Gambar 5.1: Hasil Implementasi Antarmuka

Hasil implementasi perubahan antarmuka dari perangkat lunak KIRI dapat dilihat pada Gambar 5.1. Pengguna dapat memilih salah satu dari tiga algoritma *shortest path* yang ada, yaitu Dijkstra, A-Star, dan Floyd-Warshall pada dropdown "Algorithm". Sistem akan melakukan pencarian jalur terpendek menggunakan algoritma yang dipilih pengguna.

5.2 Pengujian Fungsional dan Eksperimental

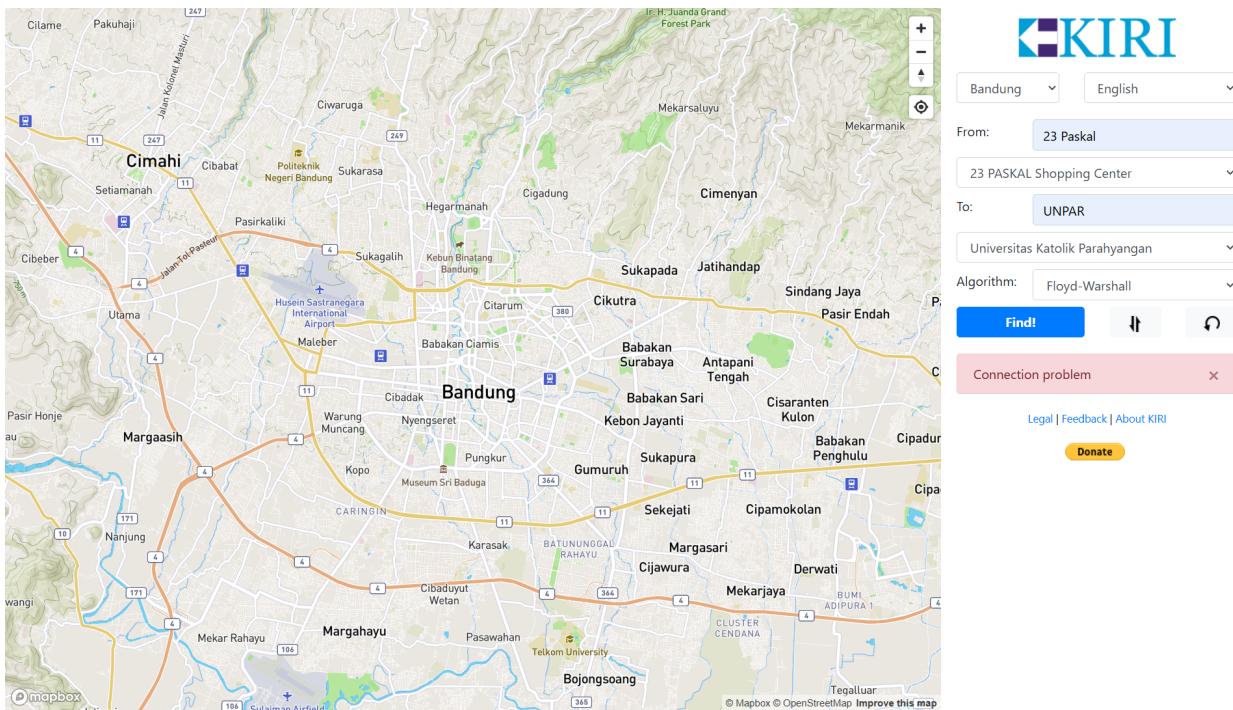
Pada bagian ini akan dijelaskan mengenai hasil pengujian dari perangkat lunak KIRI. Pengujian akan dibagi menjadi dua, yaitu pengujian fungsional dan pengujian eksperimental.

5.2.1 Pengujian Fungsional

Pengujian dilakukan dengan tujuan untuk memastikan setiap fungsi pada perangkat lunak KIRI dapat bekerja dengan baik. Pengujian ini dilakukan pada fitur dari perangkat lunak KIRI, dimulai dari, memasukan titik awal dan titik tujuan, dan memilih algoritma yang akan digunakan untuk melakukan pencarian yang merupakan sebuah fitur yang baru diimplementasikan.

Sebelum dilakukan pengujian tersebut, akan dilakukan terlebih dahulu pengujian untuk algoritma FloydWarshall saja. Pengujian dilakukan bertujuan untuk mencari jumlah maksimal dari data

"tracks" yang dapat digunakan karena pada saat ini apabila seluruh data "tracks" digunakan maka akan terjadi "java.lang.OutOfMemoryError" yang disebabkan jumlah data "tracks" yang berisikan jalur-jalur transportasi umum dalam bentuk graf terlalu banyak. Jumlah record yang terdapat pada data "tracks" adalah 140 dan pada saat pengujian akan dikurangi 10 record setiap kali pengujinya hingga algoritma Floyd-Warshall dapat berjalan dengan baik. Pengujian ini dilakukan dengan menggunakan Paskal 23 sebagai titik awal dan UNPAR sebagai titik tujuan.



Gambar 5.2: Kondisi Saat Algoritma Floyd-Warshall Tidak Berhasil Dijalankan

```
java.lang.OutOfMemoryError: Java heap space
    at travel.kiri.backend.algorithm.FloydWarshall.<init>(Unknown Source)
    at travel.kiri.backend.Worker.findRoute(Unknown Source)
    at travel.kiri.backend.ServiceListener.handle(Unknown Source)
    at travel.kiri.backend.NewMenjanganServer$1.handle(Unknown Source)
    at org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:132)
    at org.eclipse.jetty.server.Server.handle(Server.java:531)
    at org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:352)
    at org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:260)
    at org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:281)
    at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:102)
    at org.eclipse.jetty.io.ChannelEndPoint$2.run(ChannelEndPoint.java:118)
    at org.eclipse.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:762)
    at org.eclipse.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:680)
    at java.base/java.lang.Thread.run(Thread.java:1589)
```

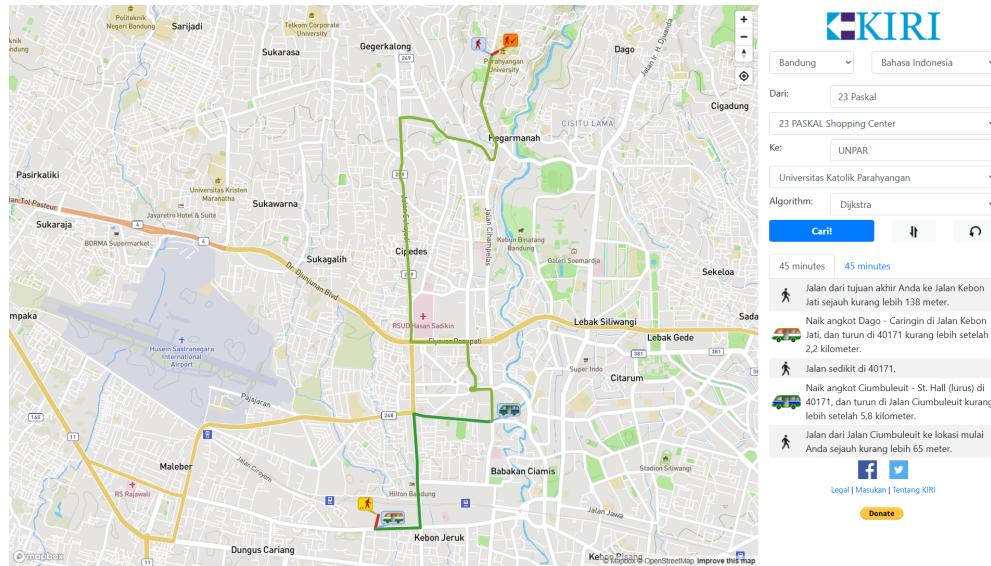
Gambar 5.3: Error Yang Menyebabkan Algoritma Floyd-Warshall Tidak Berhasil Dijalankan

Setelah dilakukan beberapa pengujian, jumlah data "tracks" yang dapat digunakan adalah lima record saja. Karena ketika jumlah data lebih dari lima record, pencarian jalur terpendek menggunakan algoritma Floyd-Warshall tidak berhasil dilakukan (lihat Gambar 5.2) yang disebabkan program mencoba menggunakan lebih banyak memori dari yang tersedia di heap *Java Virtual Machine* (JVM) (lihat Gambar 5.3). Oleh karena itu, pada pengujian ini, jumlah data "tracks" yang akan digunakan adalah lima record, dengan data yang digunakan, yaitu Ciroyom - Antapani, Ciroyom - Ciburiel, Ciumbuleuit - St. Hall (belok), Ciumbuleuit - St. Hall (lurus), dan Dago - Caringin.

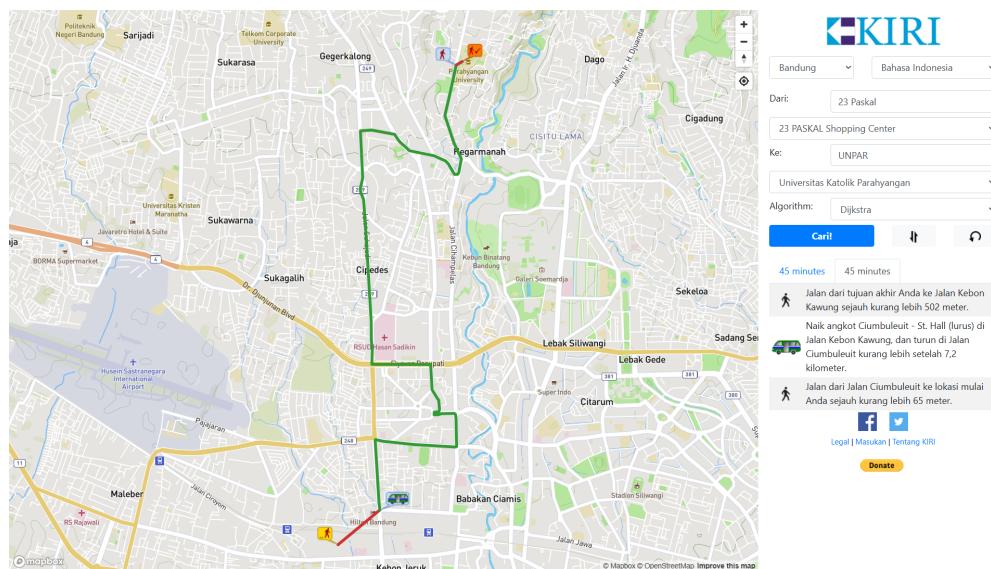
Pengujian 1: Algoritma Dijkstra

Hasil yang diharapkan dari pengujian menggunakan algoritma dijkstra ini adalah perangkat lunak KIRI dapat berjalan dengan tidak adanya error serta dapat ditemukannya jalur yang optimal. Pada pengujian ini menggunakan data sebagai berikut:

- Titik awal: "23 Paskal"
- Titik akhir: "UNPAR"
- Algoritma: "Dijkstra"



Gambar 5.4: Hasil Pengujian Algoritma Dijkstra Jalur 1



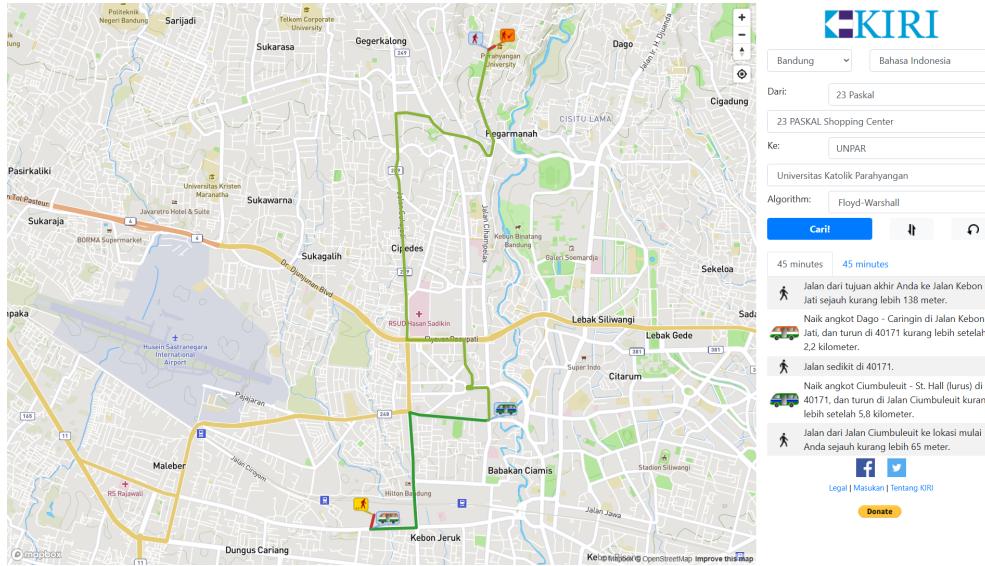
Gambar 5.5: Hasil Pengujian Algoritma Dijkstra Jalur 2

Setelah semua data yang perlukan diisi dan dilakukan pencarian, perangkat lunak KIRI berjalan dengan baik ketika melakukan pencarian menggunakan algoritma Dijkstra, serta dapat menemukan dua jalur yang optimal dari Paskal 23 menuju UNPAR (lihat Gambar 5.4 dan 5.5).

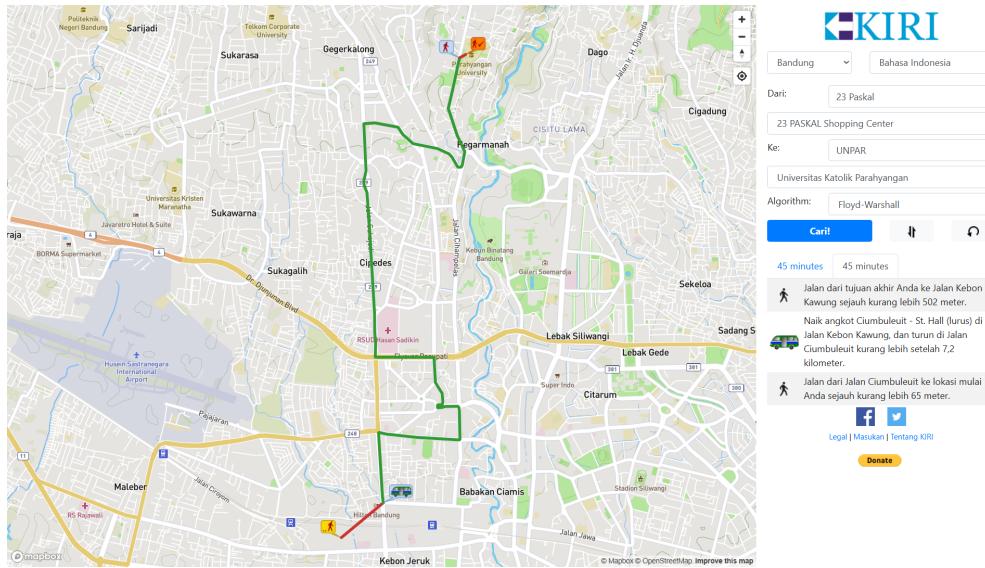
Pengujian 2: Algoritma Floyd-Warshall

Hasil yang diharapkan dari pengujian menggunakan algoritma Floyd-Warshall ini adalah perangkat lunak KIRI dapat berjalan dengan tidak adanya error serta dapat ditemukannya jalur yang optimal. Pada pengujian ini menggunakan data sebagai berikut:

- Titik awal: "23 Paskal"
- Titik akhir: "UNPAR"
- Algoritma: "Floyd-Warshall"



Gambar 5.6: Hasil Pengujian Algoritma Floyd-Warshall Jalur 1



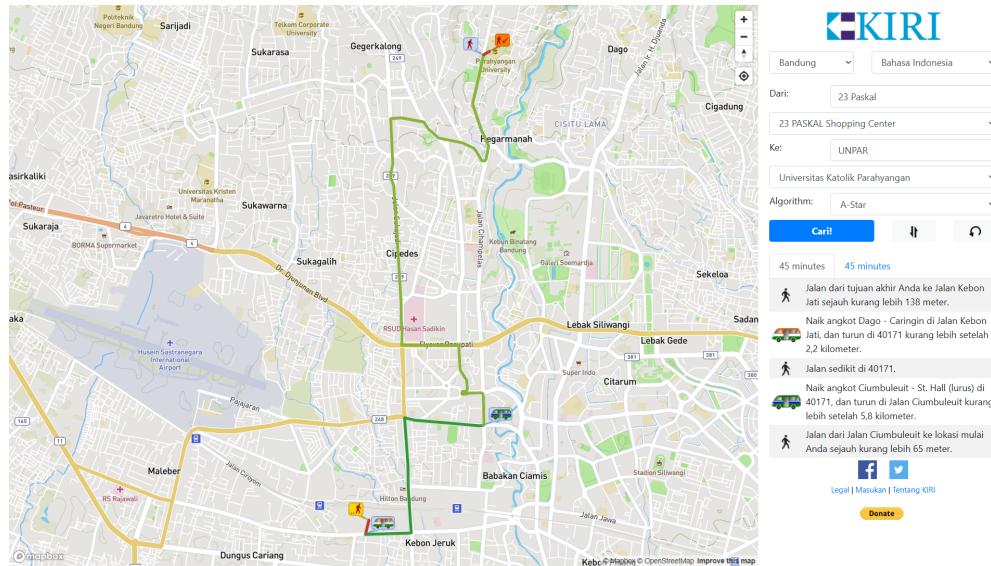
Gambar 5.7: Hasil Pengujian Algoritma Floyd-Warshall Jalur 2

Setelah semua data yang perlukan diisi dan dilakukan pencarian, perangkat lunak KIRI berjalan dengan baik ketika melakukan pencarian menggunakan algoritma Floyd-Warshall tidak terjadi error atau gagal melakukan pencarian seperti pada gambar 5.2 dan gambar 5.3, serta dapat menemukan dua jalur yang optimal dari Paskal 23 menuju UNPAR (lihat Gambar 5.6 dan 5.7).

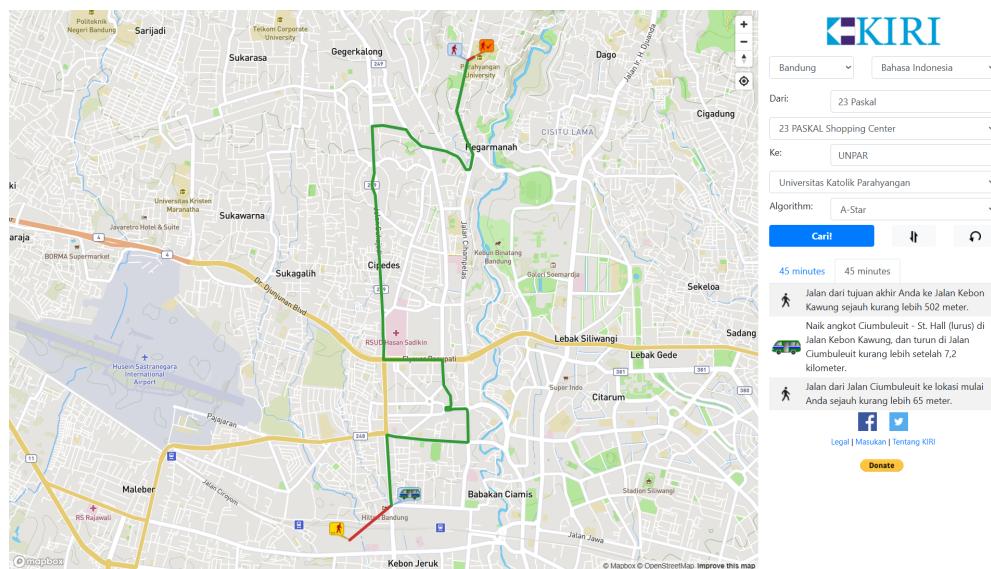
Pengujian 3: Algoritma A-Star

Hasil yang diharapkan dari pengujian menggunakan algoritma A-Star ini adalah perangkat lunak KIRI dapat berjalan dengan tidak adanya error serta dapat ditemukannya jalur yang optimal. Pada pengujian ini menggunakan data sebagai berikut:

- Titik awal: "23 Paskal"
- Titik akhir: "UNPAR"
- Algoritma: "A-Star"



Gambar 5.8: Hasil Pengujian Algoritma A-Star Jalur 1



Gambar 5.9: Hasil Pengujian Algoritma A-Star Jalur 2

Setelah semua data yang perlukan diisi dan dilakukan pencarian, perangkat lunak KIRI berjalan dengan baik ketika melakukan pencarian menggunakan algoritma A-Star, serta dapat menemukan dua jalur yang optimal dari Paskal 23 menuju UNPAR (lihat Gambar 5.8 dan 5.9).

5.2.2 Pengujian Eksperimental

Pengujian akan dilakukan untuk mengukur waktu eksekusi untuk setiap algoritma yang sudah diimplementasikan dalam melakukan pencarian atau perhitungan jalur terpendek dari titik awal hingga titik tujuan. Sebelum dilakukan pengujian, berikut disajikan kompleksitas waktu dari masing-masing algoritma untuk dijadikan sebagai acuan.

- Algoritma Dijkstra: $O((V + E) \log V)$.
- Algoritma A-Star: $O((V + E) \log V)$.
- Algoritma Floyd-Warshall: $O(V^3)$.

Pengujian ini akan dibagi menjadi dua bagian, bagian pertama hanya akan menggunakan lima record dari data "tracks" saja dan seluruh algoritma akan diuji, sedangkan pada bagian kedua akan digunakan seluruh record pada data "tracks" dan hanya algoritma Dijkstra dan A-Star saja yang akan diuji. Selain itu, untuk setiap algoritmanya akan diuji sebanyak tiga kali dan juga untuk setiap bagiannya akan terdapat tiga kasus yang berbeda. Rangkuman kasus-kasus dari pengujian yang akan dilakukan dapat dilihat pada tabel 5.1.

Pada pengujian ini, pengukuran waktu menggunakan `System.nanoTime()` sebelum proses pencarian jalur dimulai dan diakhiri dengan mencatat waktu akhir setelah seluruh proses pencarian dan pembentukan hasil selesai. Selisih antara waktu akhir dan waktu awal dihitung dan dikonversi dari nanodetik ke milidetik dengan membaginya sebesar 1.000.000. Pengukuran ini dilakukan pada kelas `Worker.java` yang kodennya bisa dilihat pada A.5.

Tabel 5.1: Rangkuman Kasus Pengujian

Pengujian	Titik Awal	Titik Akhir	Tracks
1.1	23 Paskal	UNPAR	Ciroyom - Antapani Ciroyom - Ciburial Ciumbuleuit - St. Hall (belok) Ciumbuleuit - St. Hall (lurus) Dago - Caringin
1.2	Gedung Sate	Trans Studio Bandung	Cicadas - Elang (Kalapa - Cicadas) Cicadas - Elang (Kalapa - Elang) Dago - Caringin Dago - Riung Bandung Kalapa - Ledeng
1.3	Alun-Alun Bandung	Festival Citylink	Cijerah - Ciwastra Kalapa - Buah Batu Kalapa - Cibaduyut Kalapa - Karang Setra Kalapa - Ledeng
2.1	23 Paskal	UNPAR	Seluruh tracks digunakan
2.2	Gedung Sate	Trans Studio Bandung	Seluruh tracks digunakan
2.3	Alun-Alun Bandung	Festival Citylink	Seluruh tracks digunakan

Pengujian 1.1

Pada pengujian ini titik awal yang digunakan, yaitu "23 Paskal" dan titik akhir, yaitu "UNPAR". Selain itu, pada pengujian ini data "tracks" yang digunakan, yaitu Ciroyom - Antapani, Ciroyom - Ciburial, Ciumbuleuit - St. Hall (belok), Ciumbuleuit - St. Hall (lurus), dan Dago - Caringin.

Tabel 5.2: Hasil Pengujian 1.1

Algoritma	Pengujian 1	Pengujian 2	Pengujian 3	Average
Dijkstra	3.43 ms	4.29 ms	4.23 ms	3.98 ms
A-Star	4.09 ms	3.96 ms	4.37 ms	4.14 ms
Floyd-Warshall	26469.95 ms	25765.38 ms	27804.94 ms	26680.09 ms

Pengujian 1.2

Pada pengujian ini titik awal yang digunakan, yaitu "Gedung Sate" dan titik akhir, yaitu "Trans Studio Bandung". Selain itu, pada pengujian ini data "tracks" yang digunakan, yaitu Cicadas - Elang (Kalapa - Cicadas), Cicadas - Elang (Kalapa - Elang), Dago - Caringin, Dago - Riung Bandung, dan Kalapa - Ledeng.

Tabel 5.3: Hasil Pengujian 1.2

Algoritma	Pengujian 1	Pengujian 2	Pengujian 3	Average
Dijkstra	3.17 ms	3.66 ms	3.22 ms	3.35 ms
A-Star	3.30 ms	3.19 ms	3.49 ms	3.32 ms
Floyd-Warshall	31212.38 ms	30494.62 ms	30710.07 ms	30804.69 ms

Pengujian 1.3

Pada pengujian ini titik awal yang digunakan, yaitu "Alun-Alun Bandung" dan titik akhir, yaitu "Festival Citylink". Selain itu, pada pengujian ini data "tracks" yang digunakan, yaitu Cijerah - Ciwastra, Kalapa - Buah Batu, Kalapa - Cibaduyut, Kalapa - Karang Setra, dan Kalapa - Ledeng.

Tabel 5.4: Hasil Pengujian 1.3

Algoritma	Pengujian 1	Pengujian 2	Pengujian 3	Average
Dijkstra	2.31 ms	2.18 ms	1.96 ms	2.15 ms
A-Star	2.42 ms	2.29 ms	2.26 ms	2.32 ms
Floyd-Warshall	8799.05 ms	6577.51 ms	8568.48 ms	7981.68 ms

Pengujian 2.1

Pada pengujian ini titik awal yang digunakan, yaitu "23 Paskal" dan titik akhir, yaitu "UNPAR". Selain itu, pada pengujian ini seluruh data "tracks" digunakan.

Tabel 5.5: Hasil Pengujian 2.1

Algoritma	Pengujian 1	Pengujian 2	Pengujian 3	Average
Dijkstra	44.45 ms	46.98 ms	46.22 ms	45.88 ms
A-Star	26.65 ms	30.02 ms	27.30 ms	27.99 ms

Pengujian 2.2

Pada pengujian ini titik awal yang digunakan, yaitu "Gedung Sate" dan titik akhir, yaitu "Trans Studio Bandung". Selain itu, pada pengujian ini seluruh data "tracks" digunakan.

Tabel 5.6: Hasil Pengujian 2.2

Algoritma	Pengujian 1	Pengujian 2	Pengujian 3	Average
Dijkstra	83.23 ms	75.47 ms	80.42 ms	79.70 ms
A-Star	28.45 ms	26.56 ms	27.61 ms	27.54 ms

Pengujian 2.3

Pada pengujian ini titik awal yang digunakan, yaitu "Alun-Alun Bandung" dan titik akhir, yaitu "Festival Citylink". Selain itu, pada pengujian ini seluruh data "tracks" digunakan.

Tabel 5.7: Hasil Pengujian 2.3

Algoritma	Pengujian 1	Pengujian 2	Pengujian 3	Average
Dijkstra	52.30 ms	51.21 ms	49.21 ms	50.90 ms
A-Star	34.76 ms	31.64 ms	33.47 ms	33.29 ms

Kesimpulan Pengujian

Berdasarkan hasil pengujian 1 yang bisa dilihat pada tabel 5.2, 5.3, dan 5.4, algoritma Dijkstra dan A-Star memiliki waktu eksekusi yang relatif sama. Akan tetapi, algoritma Floyd-Warshall memiliki waktu eksekusi yang cukup besar dibanding kedua algoritma lainnya, hal tersebut disebabkan karena algoritma Floyd-Warshall memiliki kompleksitas waktu yang sangat besar dan jauh lebih besar dari pada dua algoritma lainnya, serta Floyd-Warshall juga memanfaatkan matriks sehingga simpul yang diproses pun menjadi lebih banyak. Sedangkan, pada pengujian 2 algoritma A-Star memiliki waktu eksekusi yang lebih cepat dari algoritma Dijkstra berdasarkan seluruh pengujian dilakukan yang hasilnya bisa dilihat pada tabel 5.5, 5.6, dan 5.7. Meskipun kompleksitas waktu algoritma A-Star dan Dijkstra sama, tetapi algoritma A-Star bisa lebih cepat dibanding algoritma Dijkstra karena algoritma A-Star menggunakan fungsi heuristik. Dengan adanya heuristik ini, algoritma A-Star tidak perlu memproses semua simpul, melainkan hanya simpul-simpul yang dianggap lebih potensial berdasarkan nilai heuristicnya. Hal ini membuat jumlah simpul yang diproses menjadi lebih sedikit, sehingga berdampak pada waktu eksekusi yang lebih cepat.

BAB 6

KESIMPULAN DAN SARAN

Bab ini akan membahas kesimpulan dari hasil pengembangan perangkat lunak KIRI. Selain itu, akan disampaikan juga saran untuk pengembangan lanjutan yang dapat dilakukan.

6.1 Kesimpulan

Berdasarkan hasil penelitian dan implementasi serta pengujian yang telah dilakukan dalam pengembangan perangkat lunak KIRI, maka dapat disimpulkan hal-hal berikut.

1. Pada Tugas Akhir ini, perubahan arsitektur kelas pada NewMenjangan untuk menerapkan Strategy Pattern berhasil dilakukan. Penerapan dilakukan dengan membentuk sebuah *superclass* bernama `ShortestPathStrategy` sebagai antarmuka strategi. Kelas ini berfungsi sebagai abstraksi algoritma pencarian jalur, sehingga memungkinkan berbagai algoritma dapat diimplementasikan secara fleksibel sebagai turunan dari kelas-kelas algoritma pencarian jalur.
2. Implementasi algoritma A-Star dan Floyd-Warshall sebagai *concrete strategy* telah berhasil dilakukan. Pada NewMenjangan dibuat dua kelas java baru, yaitu `AStar.java` dan `FloydWarshall.java` untuk mengimplementasikan kedua algoritma tersebut. Kedua kelas tersebut juga menjadi kelas turunan dari kelas `ShortestPathStrategy` dan dapat dipertukarkan sesuai kebutuhan sistem. Selain itu, dari hasil pengujian menunjukkan bahwa kedua algoritma tersebut dapat berjalan dengan baik dan dapat menghasilkan jalur yang optimal setelah menerima masukan-masukan yang dibutuhkan.

6.2 Saran

Berdasarkan hasil penelitian dan implementasi serta pengujian yang telah dilakukan dalam pengembangan perangkat lunak KIRI, berikut ini adalah beberapa saran yang dapat disampaikan.

1. Menerapkan algoritma *shortest path* lainnya, selain Dijkstra, A-Star, dan Floyd-Warshall, guna memperluas cakupan analisis performa dan potensi peningkatan efisiensi dalam pencarian jalur terpendek, seperti algoritma Bellman-Ford, algoritma Johnson's, dan lain-lain.
2. Melakukan eksplorasi untuk pendekatan berbasis geometri sebagai alternatif pengganti representasi node, dengan tujuan meningkatkan akurasi dalam merepresentasikan posisi dan bentuk jalur secara geografis dan fleksibilitas dalam pemodelan jaringan jalur angkutan umum.

DAFTAR REFERENSI

- [1] Nugroho, P. A. dan Natali, V. (2017) Open sourcing proprietary application case study: Kiri website. *IJNMT (International Journal of New Media Technology)*, 4, 82–86.
- [2] Gamma, E., Helm, R., Johnson, R., dan Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
- [3] Version 8.4 (2024) *MySQL 8.4 Reference Manual - Including MySQL NDB Cluster 8.4*. Oracle Corporation. Austin, USA.
- [4] Diestel, R. (2017) *Graph Theory*, 5th edition. Springer, Berlin, Heidelberg.
- [5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., dan Stein, C. (2009) *Introduction to Algorithms*, 3rd edition. The MIT Press, Cambridge, MA.
- [6] Russell, S. dan Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall, Upper Saddle River, NJ.
- [7] Pressman, R. S. dan Maxim, B. R. (2019) *Software Engineering: A Practitioner's Approach*, ninth edition. McGraw-Hill Education, New York, NY, USA.

LAMPIRAN A

KODE PROGRAM

Kode A.1: Dijkstra.java

```
1 package travel.kiri.backend.algorithm;
2
3 import java.util.*;
4
5 /**
6  * Merepresentasikan sebuah penghitung jarak terdekat dengan Dijkstra.
7  * Referensi dari dijsktra.cc
8  *
9  * @author PascalAlfadian
10 *
11 */
12 public class Dijkstra extends ShorestPathStrategy {
13
14     NodeInfo[] nodeInfoLinks;
15     NodeInfo[] nodesMinHeap;
16     int heapsize;
17     int numOfNodes;
18     int memorySize;
19
20     /**
21      * Class constructor. All allocations should go here.
22      *
23      * @param graph      the list of nodes, specifying the graph
24      * @param startNode  reference to the starting node
25      * @param finishNode reference to the finish node
26      */
27     public Dijkstra(Graph graph, int startNode, int finishNode, double multiplierWalking, double penaltyTransfer) {
28         super(graph, startNode, finishNode, multiplierWalking, penaltyTransfer);
29         this.numOfNodes = graph.size();
30         this.nodesMinHeap = new NodeInfo[numOfNodes];
31         this.nodeInfoLinks = new NodeInfo[numOfNodes];
32
33         for (int i = 0; i < numOfNodes; i++) {
34             nodeInfoLinks[i] = new NodeInfo();
35         }
36     }
37
38     /**
39      * Run the Dijkstra algorithm, based on the input. After the function
40      * is run, {@link Dijkstra#getDistance(GraphNode)} will retrieve distance of
41      * each node from start
42      * node and {@link Dijkstra#getParent(GraphNode)} will retrieve the parent of
43      * each node.
44      * Complexity:  $O(|E| + |V| \log |V|)$ 
45      *
46      * @param trackTypeIdBlacklist Set of track type ids blacklisted on navigation,
47      * or null to allow everything
48      * @return the distance from source, or {@link Double#POSITIVE_INFINITY} if no
49      * path was found.
50      */
51     @Override
52     public double runAlgorithm(Set<String> trackTypeIdBlacklist) {
53
54         heapsize = 0;
55         for (int i = 0; i < numOfNodes; i++) {
56             NodeInfo ni = nodeInfoLinks[i];
57             ni.baseIndex = i;
58             ni.heapIndex = i;
59             ni.distance = Double.POSITIVE_INFINITY;
60             ni.parent = NULL_NODE;
61             nodesMinHeap[heapsize++] = ni;
62         }
63         nodeInfoLinks[startNode].distance = 0;
64         // heapify!
65
66         for (int i = heapsize / 2 - 1; i >= 0; i--) {
67             heapPercolateDown(i);
68         }
69
70         NodeInfo currentNode;
71
72         do {
73             currentNode = heapDeleteMin();
74             if (currentNode == null || currentNode.baseIndex == finishNode) {
75                 break;
```

```

76    }
77
78    UnrolledLinkedList<GraphEdge> edges = graph.get(currentNode.baseIndex).edges;
79
80    for (GraphEdge edge : edges) {
81        double weight = calculateWeight(currentNode, edge);
82        if (currentNode.distance + weight < nodeInfoLinks[edge.node].distance
83            && (trackTypeIdBlacklist == null || graph.get(edge.node).track == null
84                || !trackTypeIdBlacklist.contains(graph.get(edge.node).track.trackTypeId))) {
85            nodeInfoLinks[edge.node].distance = currentNode.distance + weight;
86            nodeInfoLinks[edge.node].parent = currentNode.baseIndex;
87            heapPercolateUp(nodeInfoLinks[edge.node].heapIndex);
88        }
89    }
90
91    } while (currentNode.distance != Double.POSITIVE_INFINITY);
92
93    return nodeInfoLinks[finishNode].distance;
94}
95
96 /**
97 * Retrieves the parent of a particular node.
98 *
99 * @param node the node to check.
100 * @return the parent of requested node, or null if it has no parent.
101 */
102 @Override
103 public int getParent(int node) {
104     return nodeInfoLinks[node].parent;
105 }
106
107 /**
108 * Retrieves the distance of a particular node.
109 *
110 * @param node the node to check.
111 * @return the distance of this node from the starting node.
112 */
113 @Override
114 public double getDistance(int node) {
115     return nodeInfoLinks[node].distance;
116 }
117
118 public double calculateWeight(NodeInfo currentNode, GraphEdge edge) {
119     Track track1 = graph.get(currentNode.baseIndex).getTrack();
120     Track track2 = graph.get(nodeInfoLinks[edge.node].baseIndex).getTrack();
121
122     // WALK from start / to finish
123     if (track1 == null || track2 == null) {
124         return multiplierWalking * edge.weight;
125     }
126     // PINDAH ANGKOT
127     else if (track1 != track2) {
128         return multiplierWalking * (penaltyTransfer + edge.weight);
129     }
130     // MASIH DI ANGKOT
131     else if (track1.getTrackId().equals(track2.getTrackId())) {
132         return edge.weight * track1.penalty;
133     }
134
135     return Double.POSITIVE_INFINITY;
136 }
137
138 void heapPercolateDown(int index) {
139     int minIndex;
140     boolean ok = true;
141
142     while (ok) {
143         minIndex = index;
144         // kalau ada anak kiri DAN kalau distance anak kiri lebih kecil
145         if ((index * 2 + 1 < heapsize) && nodesMinHeap[index * 2 + 1].distance < nodesMinHeap[minIndex].distance) {
146             minIndex = index * 2 + 1;
147         }
148
149         // kalau ada anak kanan DAN kalau distance anak kanan lebih kecil
150         if ((index * 2 + 2 < heapsize) && nodesMinHeap[index * 2 + 2].distance < nodesMinHeap[minIndex].distance) {
151             minIndex = index * 2 + 2;
152         }
153
154         // kalau memang bukan yang paling kecil maka swap
155         if (ok = (minIndex != index)) {
156             // swap!
157             NodeInfo temp = nodesMinHeap[index];
158             nodesMinHeap[index] = nodesMinHeap[minIndex];
159             nodesMinHeap[minIndex] = temp;
160
161             // swap the heap index
162             int tmp = nodesMinHeap[index].heapIndex;
163             nodesMinHeap[index].heapIndex = nodesMinHeap[minIndex].heapIndex;
164             nodesMinHeap[minIndex].heapIndex = tmp;
165
166             index = minIndex;
167         }
168     }
169 }
170
171 void heapPercolateUp(int index) {
172     // selama belum yang paling atas DAN yang di atasnya lebih besar
173     while ((index - 1) / 2 >= 0 && nodesMinHeap[(index - 1) / 2].distance > nodesMinHeap[index].distance) {
174

```

```

175        // swap!
176        NodeInfo temp = nodesMinHeap[index];
177        nodesMinHeap[index] = nodesMinHeap[(index - 1) / 2];
178        nodesMinHeap[(index - 1) / 2] = temp;
179
180        // swap the heap index
181        int tmp = nodesMinHeap[index].heapIndex;
182        nodesMinHeap[index].heapIndex = nodesMinHeap[(index - 1) / 2].heapIndex;
183        nodesMinHeap[(index - 1) / 2].heapIndex = tmp;
184
185        index = (index - 1) / 2;
186    }
187}
188
189NodeInfo heapDeleteMin() {
190    if (heapsize == 0) {
191        return null;
192    }
193
194    NodeInfo ret = nodesMinHeap[0];
195
196    nodesMinHeap[0] = nodesMinHeap[heapsize - 1];
197    heapsize--;
198    heapPercolateDown(0);
199
200    return ret;
201}
202
203String getString(int node) {
204    return nodeInfoLinks[node].toString();
205}
206
207private static class NodeInfo {
208    int baseIndex;
209    int heapIndex;
210    double distance;
211    int parent;
212
213    public NodeInfo() {
214    }
215
216    public String toString() {
217        String t = "";
218
219        t += "point=" + baseIndex + " distance=" + distance + " parent=" + parent;
220
221        return t;
222    }
223}
224}
225}
226}

```

Kode A.2: AStar.java

```

1 package travel.kiri.backend.algorithm;
2
3 import java.util.*;
4
5 public class AStar extends ShorestPathStrategy {
6
7     private NodeInfo[] nodeInfoLinks;
8     private PriorityQueue<NodeInfo> openSet;
9
10    public AStar(Graph graph, int startNode, int finishNode, double multiplierWalking, double penaltyTransfer) {
11        super(graph, startNode, finishNode, multiplierWalking, penaltyTransfer);
12        int numNodes = graph.size();
13        this.nodeInfoLinks = new NodeInfo[numNodes];
14        this.openSet = new PriorityQueue<>();
15
16        for (int i = 0; i < numNodes; i++) {
17            nodeInfoLinks[i] = new NodeInfo(i);
18        }
19    }
20
21    @Override
22    public double runAlgorithm(Set<String> trackTypeIdBlacklist) {
23        nodeInfoLinks[startNode].g = 0;
24        nodeInfoLinks[startNode].f = heuristic(startNode);
25        openSet.add(nodeInfoLinks[startNode]);
26
27        while (!openSet.isEmpty()) {
28            NodeInfo current = openSet.poll();
29
30            if (current.index == finishNode) {
31                return nodeInfoLinks[finishNode].g;
32            }
33
34            for (GraphEdge edge : graph.get(current.index).getEdges()) {
35                int neighbor = edge.node;
36                if (trackTypeIdBlacklist != null
37                    && trackTypeIdBlacklist.contains(graph.get(neighbor).getTrack().trackTypeId)) {
38                    continue;
39                }
40
41                double newCost = current.g + calculateWeight(current, edge);
42
43                if (newCost < nodeInfoLinks[neighbor].g) {

```

```

44         nodeInfoLinks[neighbor].g = newCost;
45         nodeInfoLinks[neighbor].f = newCost + heuristic(neighbor);
46         nodeInfoLinks[neighbor].parent = current.index;
47         openSet.add(nodeInfoLinks[neighbor]);
48     }
49 }
50 }
51
52     return Double.POSITIVE_INFINITY;
53 }
54
55 @Override
56 public int getParent(int node) {
57     return nodeInfoLinks[node].parent;
58 }
59
60 @Override
61 public double getDistance(int node) {
62     return nodeInfoLinks[node].g;
63 }
64
65 private double heuristic(int node) {
66     LatLon currentPos = graph.get(node).getLocation();
67     LatLon goalPos = graph.get(finishNode).getLocation();
68     return currentPos.distanceTo(goalPos);
69 }
70
71 private double calculateWeight(NodeInfo currentNode, GraphEdge edge) {
72     Track track1 = graph.get(currentNode.index).getTrack();
73     Track track2 = graph.get(edge.node).getTrack();
74
75     if (track1 == null || track2 == null) {
76         return multiplierWalking * edge.weight;
77     } else if (track1 != track2) {
78         return multiplierWalking * (penaltyTransfer + edge.weight);
79     } else {
80         return edge.weight * track1.penalty;
81     }
82 }
83
84 private static class NodeInfo implements Comparable<NodeInfo> {
85     int index;
86     double g = Double.POSITIVE_INFINITY;
87     double f = Double.POSITIVE_INFINITY;
88     int parent = NULL_NODE;
89
90     NodeInfo(int index) {
91         this.index = index;
92     }
93
94     @Override
95     public int compareTo(NodeInfo other) {
96         return Double.compare(this.f, other.f);
97     }
98 }
99 }
```

Kode A.3: FloydWarshall.java

```
1 package travel.kiri.backend.algorithm;
2
3 import java.util.*;
4
5 public class FloydWarshall extends ShorestPathStrategy {
6
7     private final int numOfNodes;
8     private double[][] dist;
9     private int[][] parent;;
10
11    public FloydWarshall(Graph graph, int startNode, int finishNode, double multiplierWalking, double penaltyTransfer) {
12        super(graph, startNode, finishNode, multiplierWalking, penaltyTransfer);
13        this.numOfNodes = graph.size();
14        System.out.println(numOfNodes);
15        dist = new double[numOfNodes][numOfNodes];
16        parent = new int[numOfNodes][numOfNodes];
17    }
18
19    @Override
20    public double runAlgorithm(Set<String> trackTypeIdBlacklist) {
21        for (int i = 0; i < numOfNodes; i++) {
22            for (int j = 0; j < numOfNodes; j++) {
23                if (i == j) {
24                    dist[i][j] = 0;
25                    parent[i][j] = -1;
26                } else {
27                    dist[i][j] = Double.POSITIVE_INFINITY;
28                    parent[i][j] = -1;
29                }
30            }
31
32            for (GraphEdge edge : graph.get(i).edges) {
33                if (trackTypeIdBlacklist == null || graph.get(edge.node).track == null
34                    || !trackTypeIdBlacklist.contains(graph.get(edge.node).track.trackTypeId)) {
35
36                    double weight = calculateWeight(i, edge);
37                    dist[i][edge.node] = weight;
38                    parent[i][edge.node] = i;
39                }
40            }
41        }
42    }
43}
```

```

40        }
41    }
42
43    for (int k = 0; k < numNodes; k++) {
44        for (int i = 0; i < numNodes; i++) {
45            for (int j = 0; j < numNodes; j++) {
46                if (dist[i][k] + dist[k][j] < dist[i][j]) {
47                    dist[i][j] = dist[i][k] + dist[k][j];
48                    parent[i][j] = parent[k][j];
49                }
50            }
51        }
52    }
53
54    return dist[startNode][finishNode];
55}
56
57 private double calculateWeight(int fromIndex, GraphEdge edge) {
58     Track trackFrom = graph.get(fromIndex).getTrack();
59     Track trackTo = graph.get(edge.node).getTrack();
60
61     if (trackFrom == null || trackTo == null) {
62         return multiplierWalking * edge.weight;
63     } else if (!trackFrom.getTrackId().equals(trackTo.getTrackId())) {
64         return multiplierWalking * (penaltyTransfer + edge.weight);
65     } else {
66         return edge.weight * trackFrom.penalty;
67     }
68 }
69
70 @Override
71 public int getParent(int node) {
72     return parent[startNode][node];
73 }
74
75 @Override
76 public double getDistance(int node) {
77     return dist[startNode][node];
78 }
79 }

```

Kode A.4: ShorestPathStrategy.java

```

1 package travel.kiri.backend.algorithm;
2
3 import java.util.*;
4
5 public abstract class ShorestPathStrategy {
6
7     protected List<GraphNode> graph;
8     protected int startNode;
9     protected int finishNode;
10    protected double multiplierWalking;
11    protected double penaltyTransfer;
12    public static final int NULL_NODE = -1;
13
14    public ShorestPathStrategy(List<GraphNode> graph, int startNode, int finishNode, double multiplierWalking,
15                               double penaltyTransfer) {
16        this.graph = graph;
17        this.startNode = startNode;
18        this.finishNode = finishNode;
19        this.multiplierWalking = multiplierWalking;
20        this.penaltyTransfer = penaltyTransfer;
21    }
22
23    public abstract double runAlgorithm(Set<String> trackTypeIdBlacklist);
24
25    public abstract int getParent(int node);
26
27    public abstract double getDistance(int node);
28 }

```

Kode A.5: Worker.java

```

1 package travel.kiri.backend;
2
3 import java.io.File;
4 import java.io.FileInputStream;
5 import java.io.FileNotFoundException;
6 import java.io.IOException;
7 import java.util.ArrayList;
8 import java.util.List;
9 import java.util.Locale;
10 import java.util.Properties;
11 import java.util.Scanner;
12 import java.util.Set;
13
14 import travel.kiri.backend.algorithm.ShorestPathStrategy;
15 import travel.kiri.backend.algorithm.Astar;
16 import travel.kiri.backend.algorithm.Dijkstra;
17 import travel.kiri.backend.algorithm.Floydwarshall;
18 import travel.kiri.backend.algorithm.Graph;
19 import travel.kiri.backend.algorithm.GraphNode;
20 import travel.kiri.backend.algorithm.LatLon;
21 import travel.kiri.backend.algorithm.Track;

```

```

22| import edu.wlu.cs.levy.CG.KDTree;
23| import edu.wlu.cs.levy.CG.KeyDuplicateException;
24| import edu.wlu.cs.levy.CG.KeySizeException;
25|
26| /**
27| * The class responsible for processing routing requests.
28| *
29| * @author PascalAlfadian
30| *
31| */
32| public class Worker {
33|
34|     public Double globalMaximumWalkingDistance;
35|     public Double global_maximum_transfer_distance;
36|     public Double globalMultiplierWalking;
37|     public Double globalPenaltyTransfer;
38|
39|     public int numberOfRequests;
40|     // in millis, needs to be converted to seconds later
41|     public long totalProcessTime;
42|
43|     List<Track> tracks;
44|     Graph nodes;
45|
46|     public Worker(String homeDirectory) throws FileNotFoundException, IOException {
47|         tracks = new ArrayList<Track>();
48|         nodes = new Graph();
49|         readConfiguration(homeDirectory + "/" + Main.MJNSERVE_PROPERTIES);
50|         Main.globalLogger.info("Configuration_were_read_successfully");
51|         readGraph(homeDirectory + "/" + Main.TRACKS_CONF);
52|         Main.globalLogger.info("Tracks_were_read_successfully");
53|         linkAngkots();
54|         cleanUpMemory();
55|         Main.globalLogger.info("Tracks_were_linked_successfully");
56|     }
57|
58| /**
59| * Cleans up memory used during precomputation.
60| */
61| private void cleanUpMemory() {
62|     for (GraphNode node : nodes) {
63|         node.getEdges().cleanUpMemory();
64|     }
65|     System.gc();
66| }
67|
68| private void readConfiguration(String filename) throws FileNotFoundException, IOException {
69|     Properties properties = new Properties();
70|     properties.load(new FileInputStream(filename));
71|     globalMaximumWalkingDistance = Double.parseDouble(properties.getProperty("maximum_walking_distance"));
72|     global_maximum_transfer_distance = Double.parseDouble(properties.getProperty("maximum_transfer_distance"));
73|     globalMultiplierWalking = Double.parseDouble(properties.getProperty("multiplier_walking"));
74|     globalPenaltyTransfer = Double.parseDouble(properties.getProperty("penalty_transfer"));
75| }
76|
77| /**
78| * Show the summary of internal track information to the log file. Useful
79| * for debugging.
80| *
81| * @return
82| */
83| String printTracksInfo() {
84|     StringBuilder sb = new StringBuilder();
85|
86|     sb.append("Loaded_tracks_information\n");
87|     sb.append(tracks.size() + "_tracks.\n");
88|     sb.append(nodes.size() + "_nodes.\n");
89|     int transferNodes = 0;
90|     int edgesCount = 0;
91|     for (GraphNode node : nodes) {
92|         if (node.isTransferNode()) {
93|             transferNodes++;
94|         }
95|         edgesCount += node.getEdges().size();
96|     }
97|     sb.append(transferNodes + "_transfer_nodes.\n");
98|     sb.append(edgesCount + "_edges.\n");
99|
100|    sb.append("Maximum_walking_=_" + globalMaximumWalkingDistance + "\n");
101|    sb.append("Maximum_transfer_=_" + global_maximum_transfer_distance + "\n");
102|    sb.append("Walking_multiplier_=_" + globalMultiplierWalking + "\n");
103|    sb.append("Transfer_penalty_=_" + globalPenaltyTransfer + "\n");
104|    sb.append("Log_Level_=_" + Main.globalLogger.getLevel() + "\n");
105|
106|    return sb.toString();
107| }
108|
109| /**
110| * Compute the shortest path between the start point and the end point.
111| *
112| * @param start
113| *          the start location to route
114| * @param finish
115| *          the finish location to route
116| * @param customMaximumWalkingDistance
117| *          the custom maximum walking distance, or
118| *          null to use default
119| * @param customMultiplierWalking
120| *          the custom walking multiplier, or null to

```

```

121     * @param customPenaltyTransfer      use default
122     *                                         the custom penalty transfer, or null to
123     *                                         use default
124     * @param trackTypeIdBlacklist       Set of disallowed track type ids, or null
125     *                                         to use all
126     * @return string containing the steps, as specified in Kalapa-Dago
127     *                                         protocol.
128     */
129
130    public String findRoute(LatLon start, LatLon finish,
131                           Double customMaximumWalkingDistance,
132                           Double customMultiplierWalking, Double customPenaltyTransfer,
133                           Set<String> trackTypeIdBlacklist, String algorithm) {
134
135        double startTime, endTime;
136
137        Main.globalLogger.fine("Worker started for " + start + " to " + finish);
138
139        // thread and stuff
140        startTime = System.nanoTime();
141
142        // setting the parameters
143        if (customMaximumWalkingDistance == null || customMaximumWalkingDistance == -1) {
144            customMaximumWalkingDistance = globalMaximumWalkingDistance;
145        }
146        if (customMultiplierWalking == null || customMultiplierWalking == -1) {
147            customMultiplierWalking = globalMultiplierWalking;
148        }
149        if (customPenaltyTransfer == null || customPenaltyTransfer == -1) {
150            customPenaltyTransfer = globalPenaltyTransfer;
151        }
152
153        // create virtual graph
154
155        int vNodesSize = nodes.size() + 2;
156        int startNode = nodes.size();
157        int endNode = nodes.size() + 1;
158
159        Graph vNodes = new Graph(vNodesSize);
160        GraphNode realNode;
161        for (int i = 0; i < nodes.size(); i++) {
162            realNode = nodes.get(i);
163            vNodes.add(new GraphNode(realNode.getLocation(), realNode.getTrack()));
164            vNodes.get(i).link(realNode);
165        }
166
167        // add start and finish node
168        vNodes.add(new GraphNode(start, null));
169        vNodes.add(new GraphNode(finish, null));
170
171        // Link startNode to other nodes by walking
172        for (int i = 0; i < nodes.size(); i++) {
173            double distance = start.distanceTo(nodes.get(i).getLocation());
174            if (distance <= customMaximumWalkingDistance && nodes.get(i).isTransferNode()) {
175                vNodes.get(startNode).push_back(i, (float) distance);
176            }
177        }
178
179        // Link endNode to other nodes by walking
180        for (int i = 0; i < nodes.size(); i++) {
181            double distance = finish.distanceTo(nodes.get(i).getLocation());
182            if (distance <= customMaximumWalkingDistance && nodes.get(i).isTransferNode()) {
183                vNodes.get(i).push_back(endNode, (float) distance);
184            }
185        }
186
187        {
188            double distance = start.distanceTo(finish);
189            if (distance <= customMaximumWalkingDistance) {
190                vNodes.get(startNode).push_back(endNode, (float) (customMultiplierWalking * distance));
191                vNodes.get(endNode).push_back(startNode, (float) (customMultiplierWalking * distance));
192            }
193        }
194
195        ShorestPathStrategy strategy;
196
197        if ("floydwarshall".equals(algorithm)) {
198            strategy = new FloydWarshall(vNodes, startNode, endNode, customMultiplierWalking, customPenaltyTransfer);
199        } else if ("astar".equals(algorithm)) {
200            strategy = new AStar(vNodes, startNode, endNode, customMultiplierWalking, customPenaltyTransfer);
201        } else {
202            strategy = new Dijkstra(vNodes, startNode, endNode, customMultiplierWalking, customPenaltyTransfer);
203        }
204
205        strategy.runAlgorithm(trackTypeIdBlacklist);
206
207        // traversing
208        int currentNode = endNode;
209
210        int lastNode, angkotLength = 0;
211        double distance = 0;
212        StringBuilder line = new StringBuilder();
213        List<String> steps = new ArrayList<String>();
214
215        while (strategy.getParent(currentNode) != strategy.NULL_NODE) {
216            lastNode = currentNode;
217            currentNode = strategy.getParent(currentNode);
218
219        }

```

```

220|
221|     if (lastNode == endNode || currentNode == startNode
222|         || !nodes.get(currentNode).getTrack().equals(nodes.get(lastNode).getTrack())) {
223|         if (angkotLength > 0) {
224|             Track t = nodes.get(lastNode).getTrack();
225|             line.insert(0, "/");
226|             line.insert(0, t.getTrackId());
227|             line.insert(0, "/");
228|             line.insert(0, t.getTrackTypeId());
229|
230|             line.append(String.format(Locale.US, "%.3f", distance));
231|             line.append("/");
232|             // places line.append(b)
233|             line.append("\n");
234|
235|             steps.add(line.toString());
236|
237|             distance = (strategy.getDistance(lastNode) - strategy.getDistance(currentNode))
238|                         / customMultiplierWalking;
239|             if (!(lastNode == endNode || currentNode == startNode)) {
240|                 distance -= customPenaltyTransfer;
241|             }
242|
243|             line = new StringBuilder("walk/walk/");
244|             if (currentNode == startNode) {
245|                 line.append("start_");
246|             } else {
247|                 LatLon location = nodes.get(currentNode).getLocation();
248|                 line.append(String.format(Locale.US, "%.5f,%.5f",
249|                                         location.lat, location.lon));
250|             }
251|
252|             if (lastNode == endNode) {
253|                 line.append("finish");
254|             } else {
255|                 LatLon location = nodes.get(lastNode).getLocation();
256|                 line.append(String.format(Locale.US, "%.5f,%.5f",
257|                                         location.lat, location.lon));
258|             }
259|
260|             line.append(String.format(Locale.US, "/%.3f/\n", distance));
261|
262|             steps.add(line.toString());
263|
264|             if (currentNode != startNode) {
265|                 LatLon location = nodes.get(currentNode).getLocation();
266|                 line = new StringBuilder(String.format(Locale.US,
267|                                         "%.5f,%.5f/", location.lat, location.lon));
268|                 distance = 0;
269|                 angkotLength = 1;
270|             }
271|         } else {
272|             // angkot!!
273|             distance += (strategy.getDistance(lastNode) - strategy.getDistance(currentNode))
274|                         / nodes.get(currentNode).getTrack().getPenalty();
275|
276|             LatLon location = nodes.get(currentNode).getLocation();
277|             line.insert(0, String.format(Locale.US, "%.5f,%.5f",
278|                                         location.lat, location.lon));
279|             angkotLength++;
280|         }
281|
282|         StringBuilder retval = new StringBuilder();
283|         if (steps.size() == 0) {
284|             retval.append("none\n");
285|         } else {
286|             for (int i = steps.size() - 1; i >= 0; i--) {
287|                 retval.append(steps.get(i));
288|             }
289|         }
290|
291|         endTime = System.nanoTime();
292|
293|         // logs
294|         double diff = (endTime - startTime) / 1000000.0;
295|         numberOfRequests++;
296|         totalProcessTime += diff;
297|
298|         Main.globalLogger.fine("Worker_ended,_elapsed:_ " + diff + "_milliseconds");
299|
300|         System.out.println("Running_Time_" + algorithm + "_=" + String.format("%.2f", diff) + "_ms");
301|
302|         return retval.toString();
303|
304|     }
305|
306| /**
307|  * Reset the statistic values.
308|  */
309| void resetStatistics() {
310|     numberOfRequests = 0;
311|     totalProcessTime = 0;
312| }
313|
314| /**
315|  * Return the number of requests since last reset.
316|  */
317| @return the number of requests
318| */

```

```

319     int getNumberOfRequests() {
320         return numberOfRequests;
321     }
322
323     /**
324      * Return the total processing time for all requests.
325      *
326      * @return the total process time.
327      */
328     double getTotalProcessTime() {
329         return totalProcessTime / 1000.0;
330     }
331
332     public boolean readGraph(String filename) {
333         try {
334             Scanner linescan = new Scanner(new File(filename));
335             Scanner scan;
336             String line, word;
337             while (linescan.hasNextLine()) {
338                 line = linescan.nextLine();
339                 // kalau ada karakter dan bukan comment
340                 if (line.length() > 0 && line.charAt(0) != '#') {
341                     scan = new Scanner(line);
342
343                     // trackid
344                     word = scan.next();
345                     Track track = new Track(word);
346
347                     // penalty
348                     word = scan.next();
349                     track.setPenalty(Double.parseDouble(word));
350
351                     // number of nodes
352                     int numNodes = scan.nextInt();
353
354                     // foreach nodes
355                     for (int i = 0; i < numNodes; i++) {
356                         int nodeIndex = nodes.size();
357                         // parse lat and lon
358                         float lat = Float.parseFloat(scan.next());
359                         float lon = Float.parseFloat(scan.next());
360                         GraphNode node = new GraphNode(new LatLon(lat, lon),
361                             track);
362                         track.addNode(node);
363                         nodes.add(node);
364
365                         // connect it with the previous node
366                         if (i > 0) {
367                             double distance = node.getLocation().distanceTo(
368                                 track.getNode(i - 1).getLocation());
369                             nodes.get(nodeIndex - 1).push_back(nodeIndex,
370                                 (float) distance);
371                         }
372                     }
373
374                     // loop
375                     int loop = scan.nextInt();
376
377                     // if loop
378                     if (loop > 0) {
379                         // connect last node to the first
380                        GraphNode first = track.getNode(0);
381                        GraphNode last = track.getNode(numNodes - 1);
382
383                         int firstIndex = nodes.size() - track.getSize();
384                         int lastIndex = nodes.size() - 1;
385
386                         double distance = first.getLocation().distanceTo(
387                             last.getLocation());
388                         // pushback edge
389                         nodes.get(lastIndex).push_back(firstIndex, (float) distance);
390                     }
391
392                     // transferNodes
393                     word = scan.next();
394                     String[] tnodes = word.split(",");
395                     // nodes
396                     int start, finish;
397                     for (int i = 0; i < tnodes.length; i++) {
398                         // if a-
399                         if (tnodes[i].indexOf("-") != -1) {
400                             String[] sf = tnodes[i].split("-");
401                             start = Integer.parseInt(sf[0]);
402                             finish = Integer.parseInt(sf[1]);
403                         } else {
404                             start = Integer.parseInt(tnodes[i]);
405                             finish = start;
406                         }
407
408                         for (int j = start; j <= finish; j++) {
409                             track.getNode(j).setTransferNode(true);
410                         }
411                     }
412
413                     tracks.add(track);
414
415                 }
416             }
417             linescan.close();

```

```

418 } catch (FileNotFoundException e) {
419     e.printStackTrace();
420     return false;
421 }
422
423     return true;
424
425 }
426
427 void linkAngkots() {
428     KDTree<GraphNodeContainer> kd = new KDTree<GraphNodeContainer>(2);
429     float minLat = Float.POSITIVE_INFINITY;
430     float minLon = Float.POSITIVE_INFINITY;
431     float maxLat = Float.NEGATIVE_INFINITY;
432     float maxLon = Float.NEGATIVE_INFINITY;
433     for (int i = 0; i < nodes.size(); i++) {
434         GraphNode n = nodes.get(i);
435         double[] key = { n.getLocation().lat, n.getLocation().lon };
436
437         if (minLat > n.getLocation().lat)
438             minLat = n.getLocation().lat;
439         if (maxLat < n.getLocation().lat)
440             maxLat = n.getLocation().lat;
441
442         if (minLon > n.getLocation().lon)
443             minLon = n.getLocation().lon;
444         if (maxLon < n.getLocation().lon)
445             maxLon = n.getLocation().lon;
446     }
447
448     boolean ok = true;
449     while (ok) {
450         try {
451             kd.insert(key, new GraphNodeContainer(n, i));
452             ok = false;
453         } catch (KeySizeException e) {
454             e.printStackTrace();
455         } catch (KeyDuplicateException e) {
456             key[0] += 0.00001;
457             key[1] += 0.00001;
458         }
459     }
460 }
461
462 LatLon minLoc = new LatLon(minLat, minLon);
463
464 double latPerKm = (maxLat - minLat) / minLoc.distanceTo(new LatLon(maxLat, minLon));
465 double lonPerKm = (maxLon - minLon) / minLoc.distanceTo(new LatLon(minLat, maxLon));
466 double threshold = 2;
467 for (int i = 0; i < nodes.size(); i++) {
468     GraphNode n = nodes.get(i);
469     double[] lowk = { n.getLocation().lat - (threshold * latPerKm * global_maximum_transfer_distance),
470                     n.getLocation().lon - (threshold * lonPerKm * global_maximum_transfer_distance) };
471     double[] uppk = { n.getLocation().lat + (threshold * latPerKm * global_maximum_transfer_distance),
472                     n.getLocation().lon + (threshold * lonPerKm * global_maximum_transfer_distance) };
473     List<GraphNodeContainer> nearby = null;
474     try {
475         nearby = kd.range(lowk, uppk);
476     } catch (KeySizeException e) {
477         e.printStackTrace();
478     }
479
480     for (GraphNodeContainer near : nearby) {
481         // if not in same track and both are transferNode
482         if (!(!nodes.get(i).getTrack().equals(near.gn.getTrack())))
483             && nodes.get(i).isTransferNode()
484             && near.gn.isTransferNode() {
485             double distance = nodes.get(i).getLocation()
486                 .distanceTo(near.gn.getLocation());
487             if (distance < global_maximum_transfer_distance) {
488                 nodes.get(i).push_back(near.index, (float) distance);
489                 near.gn.push_back(i, (float) distance);
490             }
491         }
492     }
493 }
494
495 }
496
497 public String toString() {
498     String t = "";
499     for (Track tr : tracks) {
500         t += tr + "\n";
501     }
502     return t;
503 }
504
505 private static class GraphNodeContainer {
506     public GraphNode gn;
507     public int index;
508
509     public GraphNodeContainer(GraphNode gn, int index) {
510         this.gn = gn;
511         this.index = index;
512     }
513
514 }
515
516 public String findNearbyTransports(LatLon start, Double customMaximumWalkingDistance) {
517     if (customMaximumWalkingDistance == null || customMaximumWalkingDistance == -1) {

```

```

517     customMaximumWalkingDistance = globalMaximumWalkingDistance;
518 }
519
520 StringBuilder res = new StringBuilder();
521
522 // for each tracks, check minimum distance
523 for (int idx = 0; idx < tracks.size(); idx++) {
524     Track t = tracks.get(idx);
525     int tSize = t.getSize();
526     double min = Double.POSITIVE_INFINITY;
527     for (int i = 0; i < tSize; i++) {
528         double dist = start.distanceTo(t.getNode(i).getLocation());
529         if (dist < min) {
530             min = dist;
531         }
532     }
533
534     if (min <= customMaximumWalkingDistance) {
535         res.append(t.getTrackTypeId());
536         res.append("/");
537         res.append(t.getTrackId());
538         res.append("/");
539         res.append(String.format(Locale.US, "%.3f", min));
540         res.append("\n");
541     }
542 }
543
544 return res.toString();
545 }
546
547 }

```

Kode A.6: ServiceListener.java

```

1 package travel.kiri.backend;
2
3 import java.io.IOException;
4 import java.io.UnsupportedEncodingException;
5 import java.net.URLDecoder;
6 import java.util.HashMap;
7 import java.util.HashSet;
8 import java.util.Map;
9 import java.util.Set;
10
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 import org.eclipse.jetty.http.HttpStatus;
15 import org.eclipse.jetty.server.Request;
16 import org.eclipse.jetty.server.handler.AbstractHandler;
17
18 import travel.kiri.backend.algorithm.LatLon;
19
20 public class ServiceListener extends AbstractHandler {
21
22     public static final String PARAMETER_START = "start";
23     public static final String PARAMETER_FINISH = "finish";
24     public static final String PARAMETER_MAXIMUM_WALKING = "mw";
25     public static final String PARAMETER_WALKING_MULTIPLIER = "wm";
26     public static final String PARAMETER_PENALTY_TRANSFER = "pt";
27     public static final String PARAMETER_TRACKTYPEID_BLACKLIST = "ttib";
28     public static final String PARAMETER_ALGO = "algo";
29
30     Worker worker;
31
32     public ServiceListener(Worker w) {
33         this.worker = w;
34     }
35
36     @Override
37     public void handle(String target, Request baseRequest,
38             HttpServletRequest request, HttpServletResponse response) throws IOException {
39         String query = request.getQueryString();
40         Map<String, String> params = new HashMap<String, String>();
41         params = parseQuery(query);
42
43         String responseText = "Internal_error:_not_updated";
44         int responseCode = HttpStatus.INTERNAL_SERVER_ERROR_500;
45
46         try {
47             LatLon start = null;
48             try {
49                 start = new LatLon(params.get(PARAMETER_START));
50             } catch (NullPointerException e) {
51             }
52
53             LatLon finish = null;
54             try {
55                 finish = new LatLon(params.get(PARAMETER_FINISH));
56             } catch (NullPointerException e) {
57             }
58
59             String maximumWalking = params.get(PARAMETER_MAXIMUM_WALKING);
60             String walkingMultiplier = params.get(PARAMETER_WALKING_MULTIPLIER);
61             String penaltyTransfer = params.get(PARAMETER_PENALTY_TRANSFER);
62             String trackTypeIdBlacklistString = params.get(PARAMETER_TRACKTYPEID_BLACKLIST);
63             String algorithm = params.get(PARAMETER_ALGO);
64

```

```

65     Set<String> trackTypeIdBlacklist = null;
66     if (trackTypeIdBlacklistString != null) {
67         String[] trackTypeIdsArray = trackTypeIdBlacklistString.split(",");
68         trackTypeIdBlacklist = new HashSet<String>();
69         for (String member : trackTypeIdsArray) {
70             trackTypeIdBlacklist.add(member);
71         }
72     }
73
74     if (start != null && finish != null) {
75         // findRoute
76         responseText = worker.findRoute(
77             start,
78             finish,
79             maximumWalking == null ? null : new Double(maximumWalking),
80             walkingMultiplier == null ? null : new Double(walkingMultiplier),
81             penaltyTransfer == null ? null : new Double(penaltyTransfer),
82             trackTypeIdBlacklist,
83             algorithm);
84         responseCode = HttpStatus.OK_200;
85     } else if (start != null && finish == null) {
86         // findNearby
87         responseText = worker.findNearbyTransports(
88             start,
89             maximumWalking == null ? null : new Double(maximumWalking));
90         responseCode = HttpStatus.OK_200;
91     } else {
92         responseText = "Please_provide_start_and_finish_location";
93         responseCode = HttpStatus.BAD_REQUEST_400;
94     }
95
96 } catch (Exception e) {
97     e.printStackTrace();
98     responseText = e.toString();
99     responseCode = HttpStatus.INTERNAL_SERVER_ERROR_500;
100 }
101
102 response.setStatus(responseCode);
103 baseRequest.setHandled(true);
104 response.getWriter().println(responseText);
105 }
106
107 private Map<String, String> parseQuery(String query)
108     throws UnsupportedEncodingException, NullPointerException {
109     Map<String, String> params = new HashMap<String, String>();
110     if (query != null) {
111         String pairs[] = query.split("&\"");
112         for (String pair : pairs) {
113             String param[] = pair.split("=\"");
114
115             String key = null;
116             String value = null;
117             if (param.length > 0) {
118                 key = URLDecoder.decode(param[0],
119                     System.getProperty("file.encoding"));
120             }
121             if (param.length > 1) {
122                 value = URLDecoder.decode(param[1],
123                     System.getProperty("file.encoding"));
124             }
125
126             params.put(key, value);
127         }
128     } else {
129         throw new NullPointerException("query_is_null");
130     }
131     return params;
132 }
133 }
```

Kode A.7: main.php

```

1 <?php
2 defined('BASEPATH') or exit('No direct script access allowed');
3 ?>
4 <!DOCTYPE html>
5 <html class="no-js" lang="en">
6
7 <head>
8     <meta charset="utf-8" />
9     <!-- Global site tag (gtag.js) - Google Analytics -->
10    <script async src="https://www.googletagmanager.com/gtag/js?id=G-QXRGWXE3RZ"></script>
11    <script>
12        window.dataLayer = window.dataLayer || [];
13
14        function gtag() {
15            dataLayer.push(arguments);
16        }
17        gtag('js', new Date());
18
19        gtag('config', 'G-QXRGWXE3RZ');
20    </script>
21 <title>KIRI</title>
22 <meta name="viewport" content="width=device-width,_initial-scale=1.0" />
23 <meta name="description" content="=$this-&gt;lang-&gt;line('meta-description')? />
24 <meta name="author" content="Project_Kiri_(KIRI)" />
25 <link rel="stylesheet" href="/ext/bootstrap/css/bootstrap.min.css" />
26 <link href="https://api.mapbox.com/mapbox-gl-js/v1.10.0/mapbox-gl.css" rel="stylesheet" />
```

```

27    <link rel="stylesheet" href="/styleIndex.css" />
28    <link rel="icon" href="/images/favicon.ico" type="image/x-icon">
29 </head>
30
31 <body>
32   <div class="container-fluid">
33     <div class="row_order-3">
34       <div id="controlpanel" class="col-lg-3 col-md-6 order-md-9">
35         <div class="col">
36           
37         </div>
38
39       <div class="row_p-1_pb-3">
40         <div class="col-5">
41           <select id="regionselect" class="form-control">
42             <?php foreach ($regions as $key => $value): ?>
43               <option value=<?= $key ?> <?= ($region == $key ? 'selected' : '') ?>><?= $value['name'] ?><
44                 option>
45             <?php endforeach; ?>
46           </select>
47         </div>
48         <div class="col-7">
49           <select id="localeselect" class="form-control">
50             <?php foreach ($languages as $key => $value): ?>
51               <option value=<?= $key ?> <?= ($locale == $key ? 'selected' : '') ?>><?= $value['name'] ?><
52                 option>
53             <?php endforeach; ?>
54           </select>
55         </div>
56       <div class="row_p-1">
57         <div class="col-3">
58           <span for="startInput" class="align-middle"><?= $this->lang->line('From') ?>:</span>
59         </div>
60         <div class="col-9">
61           <input type="text" id="startInput" class="form-control" value="" placeholder=<?= $this->lang->line(
62             'placeholder-from') ?>">
63         </div>
64       <div class="row_p-1">
65         <div class="col-lg-12">
66           <select id="startSelect" class="form-control_hidden"></select>
67         </div>
68       <div class="row_p-1">
69         <div class="col-3">
70           <span for="finishInput" class="align-middle"><?= $this->lang->line('To') ?>:</span>
71         </div>
72         <div class="col-9">
73           <input type="text" id="finishInput" class="form-control" value="" placeholder=<?= $this->lang->line(
74             'placeholder-to') ?>">
75         </div>
76       <div class="row_p-1">
77         <div class="col-lg-12">
78           <select id="finishSelect" class="form-control_hidden"></select>
79         </div>
80       <div class="row_p-0">
81         <div class="col-lg-12">
82           <select id="selectAlgo" class="form-control_hidden"></select>
83         </div>
84       <div class="row_p-1">
85         <div class="col-3">
86           <span for="algoselect" class="align-middle">Algorithm:</span>
87         </div>
88         <div class="col-9">
89           <select id="algoselect" class="form-control">
90             <option value="dijkstra">Dijkstra</option>
91             <option value="floydwarshall">Floyd-Warshall</option>
92             <option value="astar">A*</option>
93           </select>
94         </div>
95       <div class="row_p-1_pb-3">
96         <div class="btn-group_fullwidth" role="group">
97           <div class="col-sm-6">
98             <a href="#" class="btn btn-primary_btn-block" id="findbutton"><strong><?= $this->lang->line('Find')
99               ?></strong></a>
100            </div>
101            <div class="col-sm-3">
102              <a href="#" class="btn btn-light_btn-block" id="swapbutton"></a>
103            </div>
104            <div class="col-sm-3">
105              <a href="#" class="btn btn-light_btn-block" id="resetbutton"></a>
107            </div>
108          </div>
109        </div>
110      </div>
111      <div class="row_p-1">
112        <div class="col-12" id="routingresults">
113          <div id="results-section-container"></div>
114        </div>
115      </div>
116      <div class="row">
117        <div class="col-12">
118          <footer>
119        </div>

```

```

120    <a href=<?=_$this->lang->line('url-legal')_?>><?= $this->lang->line('Legal') ?></a> |  

121    <a href=<?=_$this->lang->line('url-feedback')_?>><?= $this->lang->line('Feedback') ?></a> |  

122    <a href=<?=_$this->lang->line('url-about')_?>><?= $this->lang->line('About_KIRI') ?></a><br /><br />  

123    <form action="https://www.paypal.com/cgi-bin/webscr" method="post" target="_top">  

124        <input type="hidden" name="cmd" value="s-xclick">  

125        <input type="hidden" name="hosted_button_id" value="WKWS26A57WHJG">  

126        <input type="image" src="https://www.paypalobjects.com/en_US/i/btn(btn_donate_SM.gif" border="0"  

127            name="submit" alt="PayPal - The safer, easier way to pay online!">  

128          

129    </form>  

130    </footer>  

131    &nbsp;  

132    </div>  

133    <div id="map" class="col-md-6 col-lg-9"></div>  

134  </div>  

135  </div>  

136  </div>  

137  <script src="/ext/jquery/jquery.min.js"></script>  

138  <script src="/ext/bootstrap/js/bootstrap.min.js"></script>  

139  <script src="https://api.mapbox.com/mapbox-gl-js/v1.10.0/mapbox-gl.js"></script>  

140  <script>  

141      var region = '<?=_$region_?>';  

142      var input_text = <?= json_encode($inputText) ?>;  

143      var coordinates = <?= json_encode($inputCoordinate) ?>;  

144  </script>  

145  <script src="/mainpage/js/protocol.js"></script>  

146  <script src="/mainpage/js/main.js?locale=<?=_$locale_?>"></script>  

147  <?php if ($newhome_popup): ?>  

148      <div class="modal_fade" id="popupModal" tabindex="-1" role="dialog" aria-labelledby="modalLabel" aria-hidden="true">  

149          <div class="modal_dialog">  

150              <div class="modal_content">  

151                  <div class="modal_header">  

152                      <h5 class="modal_title" id="modalodalLabel">KIRI has a new home</h5>  

153                      <button type="button" class="close" data-dismiss="modal" aria-label="Close">  

154                          <span aria-hidden="true">&times;</span>  

155                      </button>  

156                  </div>  

157                  <div class="modal_body">  

158                      <?= $this->lang->line('newhome-message') ?>  

159                  </div>  

160                  <div class="modal_footer">  

161                      <button type="button" class="btn btn-primary" data-dismiss="modal">OK</button>  

162                  </div>  

163          </div>  

164      </div>  

165      <script>  

166          $(document).ready(function() {  

167              $('#popupModal').modal();  

168          });
169      </script>
170  <?php endif; ?>
171 </body>
172 </html>

```

Kode A.8: main.js

```

1  <?php  

2  defined('BASEPATH') OR exit('No direct script access allowed');  

3  ?>  

4  var regions = <?=json_encode($this->config->item('regions'))?>;  

5  mapboxgl.accessToken = <?=json_encode($this->config->item('mapbox-token'))?>;  

6  

7  const trackColors = ['#339933', '#8B33B', '#267373'];  

8  const walkColor = '#CC3333';  

9  

10 var map_component_ids = [];  

11  

12 $(document).ready(function () {  

13     var protocol = new CicahumLedengProtocol(<?=json_encode($this->config->item('cicaheumledeng-key'))?>, function (message) {  

14         clearSecondaryAlerts();  

15         showAlert('<?=$this->lang->line("Connection_problem")?>', 'alert');  

16     });  

17  

18     var map = new mapboxgl.Map({  

19         container: 'map', // container id  

20         style: 'mapbox://styles/mapbox/outdoors-v11', // stylesheet location  

21         center: [regions[region].lon, regions[region].lat], // starting position [lng, lat]  

22         zoom: regions[region].zoom // starting zoom  

23     });  

24     map.addControl(new mapboxgl.NavigationControl());  

25     var resultVectorSource = {  

26         'type': 'FeatureCollection',  

27         'features': [  

28             {}  

29         ]};  

30  

31     // Start geolocation tracking routine  

32     var geolocation = new mapboxgl.GeolocateControl({  

33         positionOptions: {  

34             enableHighAccuracy: true,  

35             timeout: 1000  

36         },  

37         trackUserLocation: true
38

```

```

39    });
40    map.addControl(geolocation);
41    // End geolocation tracking routine
42
43    var markers = { start: null, finish: null };
44    var routingResultMarkers = [];
45
46    // Preload start and finish marker image
47    var startMarkerElement = document.createElement('img');
48    startMarkerElement.setAttribute('src', '../../../../../images/start.png');
49    startMarkerElement.setAttribute('alt', 'start_marker');
50    var finishMarkerElement = document.createElement('img');
51    finishMarkerElement.setAttribute('src', '../../../../../images/finish.png');
52    finishMarkerElement.setAttribute('alt', 'finish_marker');
53    var walkMarkerElement = document.createElement('img');
54    walkMarkerElement.setAttribute('src', '../../../../../images/means/walk/balloon/walk.png');
55    walkMarkerElement.setAttribute('alt', 'walk_marker');
56
57    var focused = false;
58    $.each(['start', 'finish'], function (sfIndex, sfValue) {
59      var placeInput = $('#' + sfValue + 'Input');
60      var placeSelect = $('#' + sfValue + 'Select');
61
62      if (input_text[sfValue] != null) {
63        placeInput.val(input_text[sfValue]);
64        if (coordinates[sfValue] != null) {
65          placeInput.prop('disabled', true);
66          var lonlat = stringToLonLat(coordinates[sfValue]);
67          mapCenter = lonlat;
68        }
69      } else if (focused === false) {
70        placeInput.focus();
71        focused = true;
72      }
73      $('#' + sfValue + 'Select').addClass('hidden');
74
75      placeInput.change(function () {
76        coordinates[sfValue] = null;
77        if (markers[sfValue] != null) {
78          markers[sfValue].remove();
79          markers[sfValue] = null;
80        }
81      });
82      placeSelect.change(function () {
83        clearAlerts();
84        showAlert('' + '<?=>lang->line("Please wait")?>...', 'secondary');
85        coordinates[sfValue] = $(this).val();
86        checkCoordinatesThenRoute(coordinates);
87      });
88    });
89
90    // Event handlers
91    var localeSelect = $('#localeselect');
92    localeSelect.change(function () {
93      // IE fix: when window.location.origin is not available
94      if (!window.location.origin) {
95        window.location.origin = window.location.protocol + "//" + window.location.hostname + (window.location.port ? ":" + window.location.port : '');
96      }
97      window.location.replace(window.location.origin + "?locale=" + localeSelect.val());
98    });
99    var regionSelect = $('#regionselect');
100   regionSelect.change(function () {
101     updateRegion(regionSelect.val(), true);
102     coordinates['start'] = null;
103     coordinates['finish'] = null;
104   });
105   $('#findbutton').click(findRouteClicked);
106   $('input').keyup(function (e) {
107     if (e.keyCode === 13) {
108       findRouteClicked();
109     }
110   });
111   $('#resetbutton').click(resetScreen);
112   $('#swapbutton').click(swapInput);
113
114   // Map click event
115   map.on('click', function (event) {
116     if ($('#startInput').val() === '') {
117       markers['start'] = new mapboxgl.Marker({
118         element: startMarkerElement,
119         anchor: 'bottom-right',
120       });
121       markers['start'].setLngLat([event.lngLat['lng'], event.lngLat['lat']]);
122       markers['start'].addTo(map);
123       $('#startInput').val(latLngToString(event.lngLat));
124     } else if ($('#finishInput').val() === '') {
125       markers['finish'] = new mapboxgl.Marker({
126         element: finishMarkerElement,
127         anchor: 'bottom-left',
128       });
129       markers['finish'].setLngLat([event.lngLat['lng'], event.lngLat['lat']]);
130       markers['finish'].addTo(map);
131       $('#finishInput').val(latLngToString(event.lngLat));
132     }
133   });
134
135   // Lastly, execute search if both start and finish are ready
136

```

```

137 if ($('#startInput').val() != '' && $('#finishInput').val() != '') {
138   findRouteClicked();
139 }
140
141 /**
142 * Check if coordinates are complete. If yes, then start routing.
143 * @param coordinates the coordinates to check.
144 */
145 function checkCoordinatesThenRoute(coordinates) {
146   if (coordinates['start'] != null && coordinates['finish'] != null) {
147
148     protocol.findRoute(
149       coordinates['start'],
150       coordinates['finish'],
151       $('#algoselect').val(),
152       '<?=$locale?>',
153       function (results) {
154         if (results.status === 'ok') {
155           showRoutingResults(results);
156         } else {
157           clearSecondaryAlerts();
158           showAlert('<?=$this->lang->line("Connection_problem")?>', 'alert');
159         }
160       });
161   }
162 }
163
164
165 function clearRoutingResultsOnMap() {
166   updateRegion(region, false);
167   // Remove layers in backward manner, because layer is dependant on source
168   // but source was created first
169   for (let i = map_component_ids.length; i >= 0; i--) {
170     if (map.getLayer(map_component_ids[i])) {
171       map.removeLayer(map_component_ids[i]);
172     }
173     if (map.getSource(map_component_ids[i])) {
174       map.removeSource(map_component_ids[i]);
175     }
176   }
177   // Remove markers
178   for (let i = 0; i < routingResultMarkers.length; i++) {
179     routingResultMarkers[i].remove();
180   }
181   routingResultMarkers = [];
182 }
183
184 function clearRoutingResultsOnTable() {
185   $('.nav').remove();
186   $('.tab-content').remove();
187 }
188
189 function clearAlerts() {
190   $('.alert').remove();
191 }
192
193 function clearSecondaryAlerts() {
194   $('.alert.alert-secondary').fadeOut();
195 }
196
197 function clearStartFinishMarker() {
198   if (markers['start'] != null) {
199     markers['start'].remove();
200     markers['start'] = null;
201   }
202   if (markers['finish'] != null) {
203     markers['finish'].remove();
204     markers['finish'] = null;
205   }
206 }
207
208 /**
209 * A function that will be called when find route button is clicked
210 * (or triggered by another means)
211 */
212 function findRouteClicked() {
213   // Validate
214   var cancel = false;
215   $.each(['start', 'finish'], function (sfIndex, sfValue) {
216     if ($('#' + sfValue + 'Input').val() === '') {
217       cancel = true;
218       return;
219     }
220   });
221   if (cancel) {
222     showAlert('<?=$this->lang->line("Fill_both")?>', 'alert');
223     return;
224   }
225
226   clearAlerts();
227   clearRoutingResultsOnTable();
228   showAlert(' ' + '<?=$this->lang->line("Please_wait")?>...', 'secondary');
229
230   var completedLatLon = 0;
231   $.each(['start', 'finish'], function (sfIndex, sfValue) {
232     var placeInput = $('#' + sfValue + 'Input');
233     var placeSelect = $('#' + sfValue + 'Select');
234
235

```

```

236        if (isLatLang(placeInput.val())) {
237            coordinates[sfValue] = placeInput.val();
238            completedLatLon++;
239        } else {
240            if (coordinates[sfValue] == null) {
241                // Coordinates not yet ready, we do a search place
242                if (coordinates[sfValue] == null) {
243                    // Coordinates not yet ready, we do a search place
244                    protocol.searchPlace(
245                        placeInput.val(),
246                        region,
247                        function (result) {
248                            placeSelect.empty();
249                            placeSelect.addClass('hidden');
250                            if (result.status != 'error') {
251                                if (result.searchresult.length > 0) {
252                                    $().each(result.searchresult, function (index, value) {
253                                        var placeSelect = $('#' + sfValue + 'Select');
254                                        placeSelect
255                                            .append('<option>' + value['location'] + '</option>');
256                                            .attr('value', value['location']);
257                                            .text(value['placename']));
258                                        placeSelect.removeClass('hidden');
259                                    });
260                                    coordinates[sfValue] = result.searchresult[0]['location'];
261                                    checkCoordinatesThenRoute(coordinates);
262                                } else {
263                                    clearSecondaryAlerts();
264                                    clearRoutingResultsOnMap();
265                                    showAlert(placeInput.val() + '<?=$this->lang->line("not_found")?>', 'alert');
266                                }
267                            } else {
268                                clearSecondaryAlerts();
269                                clearRoutingResultsOnMap();
270                                showAlert('<?=$this->lang->line("Connection_problem")?>', 'alert');
271                            }
272                        });
273                    } else {
274                        // Coordinates are already available, skip searching
275                        completedLatLon++;
276                    }
277                }
278            });
279        if (completedLatLon == 2) {
280            checkCoordinatesThenRoute(coordinates);
281        }
282    }
283
284 /**
285 * Convert lon/lat into text representation
286 * @return the lon/lat in string, 5 digits after '.'
287 */
288 function latLangToString(lonLat) {
289     return lonLat['lat'].toFixed(5) + ',' + lonLat['lng'].toFixed(5);
290 }
291
292 /**
293 * Checks if the text provided is in a lat/lng format.
294 * @return true if it is, false otherwise.
295 */
296 function isLatLang(text) {
297     return text.match(/^-?[0-9.]+,-?[0-9.]+/);
298 }
299
300 function resetScreen() {
301     clearRoutingResultsOnTable();
302     clearRoutingResultsOnMap();
303     clearAlerts();
304     clearStartFinishMarker();
305     $.each(['start', 'finish'], function (sfIndex, sfValue) {
306         var placeInput = $('#' + sfValue + 'Input');
307         placeInput.val('');
308         placeInput.prop('disabled', false);
309         $('#' + sfValue + 'Select').addClass('hidden');
310     });
311 }
312
313 /**
314 * Sets a cookie in browser, adapted from http://www.w3schools.com/js/js_cookies.asp
315 */
316 function setCookie(cname, cvalue) {
317     var d = new Date();
318     d.setTime(d.getTime() + (365 * 24 * 60 * 60 * 1000));
319     var expires = "expires=" + d.toGMTString();
320     document.cookie = cname + "=" + cvalue + ";u" + expires;
321 }
322
323 /**
324 * Show alert message
325 * @param message the message
326 * @param cssClass the bootstrap css class
327 */
328 function showAlert(message, cssClass) {
329     if (cssClass === 'alert') {
330         cssClass = 'danger'
331     }
332     var alert = $('

' +
333         '&times;</a></div>'\);


```

```

334     $($('#routingresults')).prepend(alert);
335 }
336 /**
337 * Shows the routing result on panel an map
338 * @param results the routing result JSON response
339 */
340 function showRoutingResults(results) {
341     clearStartFinishMarker();
342     clearRoutingResultsOnTable();
343     clearSecondaryAlerts();
344     var kiriURL = encodeURIComponent('<?=$_base_url()_?>?start=' + encodeURIComponent($('#startInput').val()) + '&finish=' + encodeURIComponent($('#finishInput').val()));
345     var kiriMessage = encodeURIComponent('<?=$this->lang->line("I_utake_public_transport")?>'.replace('%finish%', $('#finishInput').val().replace('%start%', $('#startInput').val())));
346     var sectionContainer = $('<div></div>');
347     var templ = $('<ul class="nav-tabs" role="tablist"></ul>');
348     var temp2 = $('<div class="tab-content"></div>');
349     $('#routingresults').append(sectionContainer);
350     $.each(results.routingresults, function(resultIndex, result) {
351         var resultHTML1 = resultIndex === 0 ? '<li><a class="nav-link active active-tabs" :<li><a class="nav-link" :';
352         resultHTML1 += 'text-decoration:none" data-toggle="tab" href="#panel1-' + (resultIndex + 1) + '" role="tab">' + (
353             result.traveltime === null ? '<?= $this->lang->line("Oops")?' : result.traveltime) + '</a></li>';
354         var resultHTML2 = '<div id="panel1-' + (resultIndex + 1) + '">';
355         resultHTML2 += resultIndex === 0 ? '<table class="tab-pane active" role="tabpanel"><table class="table-striped"> : <class="x-tab-pane" role="tabpanel"><table class="table-striped">';
356         $.each(result.steps, function (stepIndex, step) {
357             resultHTML2 += '<tr><td class="p-1">' + step[1] + '</td><td class="p-1">' + step[3];
358             resultHTML2 += '</td></tr>';
359         });
360         resultHTML2 += "<tr><td class=\"p-1 center\" colspan=\"2\">";
361         resultHTML2 += "<a href=\"#panel1-" + (resultIndex + 1) + "\">\n";
362         resultHTML2 += "Share to Facebook" + "\n";
363         resultHTML2 += "<a href=\"https://www.facebook.com/sharer/sharer.php?u=" + kiriURL + "\"><img alt=" + "\n";
364         resultHTML2 += "Share to Twitter" + "\n";
365         resultHTML2 += "<a href=\"https://twitter.com/intent/tweet?via=kiriupdate&text=" + kiriMessage + "+" + "\n";
366         resultHTML2 += "</a>" + "\n";
367         resultHTML2 += '</td></tr>\n';
368         resultHTML2 += '</table></div>';
369         templ.append(resultHTML1);
370         temp2.append(resultHTML2);
371     });
372     sectionContainer.append(templ);
373     sectionContainer.append(temp2);
374     $(".nav-link").on("click", function(){
375         $(".nav").find(".active").removeClass("active");
376         $(".tab-pane").removeClass("active");
377         $(this).addClass("active");
378         $(this).attr("href").addClass("active");
379     });
380     $.each(results.routingresults, function(resultIndex, result) {
381         $('a[href="#panel1-' + (resultIndex + 1) + '"]').click(function() {
382             showSingleRoutingResultOnMap(result);
383         });
384     });
385     showSingleRoutingResultOnMap(results.routingresults[0]);
386 }
387 /**
388 * Shows a single routing result on map
389 * @param result the JSON array for one result
390 */
391 function showSingleRoutingResultOnMap(result) {
392     clearRoutingResultsOnMap();
393     let trackCounter = 0;
394     let bounds = null;
395     $.each(result.steps, function (stepIndex, step) {
396         if (step[0] === 'none') {
397             // Don't draw line
398         } else {
399             let coordinates = stringArrayToPointArray(step[2]);
400             map.addSource('source_' + stepIndex, {
401                 'type': 'geojson',
402                 'data': {
403                     'type': 'Feature',
404                     'properties': {
405                         'color': step[0] == 'walk' ? walkColor : trackColors[trackCounter++ % trackColors.length]
406                     },
407                     'geometry': {
408                         'type': 'LineString',
409                         'coordinates': coordinates
410                     }
411                 });
412             map_component_ids.push('source_' + stepIndex);
413             map.addLayer({
414                 'id': 'layer_' + stepIndex,
415                 'type': 'line',
416                 'source': 'source_' + stepIndex,
417                 'layout': {
418                     'line-join': 'round',
419                     'line-cap': 'round'
420                 },
421                 'paint': {
422                     'line-color': ['get', 'color'],
423                     'line-width': 5
424                 }
425             });
426     });
427 }

```

```

426    map_component_ids.push('layer_' + stepIndex);
427    for (let i = 0; i < coordinates.length; coordinates++) {
428      if (bounds) {
429        bounds.extend(coordinates[i]);
430      } else {
431        bounds = new mapboxgl.LngLatBounds(coordinates[i], coordinates[i]);
432      }
433    }
434  }
435
436  if (stepIndex === 0) {
437    let marker = new mapboxgl.Marker({
438      element: startMarkerElement,
439      anchor: 'bottom-right'
440    });
441    marker.setLngLat(stringToLonLat(step[2][0]));
442    marker.addTo(map);
443    routingResultMarkers.push(marker);
444  } else {
445    var lonlat = stringToLonLat(step[2][0]);
446    if (step[0] != "walk") {
447      let angkotMarkerElement = document.createElement('img');
448      angkotMarkerElement.setAttribute('src', '../../../../../images/means/' + step[0] + '/balloon/' + step[1] + '.png');
449      angkotMarkerElement.setAttribute('alt', 'angkot_marker');
450      let marker = new mapboxgl.Marker({
451        element: angkotMarkerElement,
452        anchor: 'bottom-left'
453      });
454      marker.setLngLat(lonlat);
455      marker.addTo(map);
456      routingResultMarkers.push(marker);
457    } else {
458      let marker = new mapboxgl.Marker({
459        element: walkMarkerElement,
460        anchor: 'bottom-right'
461      });
462      marker.setLngLat(lonlat);
463      marker.addTo(map);
464      routingResultMarkers.push(marker);
465    }
466  }
467
468  if (stepIndex === result.steps.length - 1) {
469    let marker = new mapboxgl.Marker({
470      element: finishMarkerElement,
471      anchor: 'bottom-left'
472    });
473    marker.setLngLat(stringToLonLat(step[2][step[2].length - 1]));
474    marker.addTo(map);
475    routingResultMarkers.push(marker);
476  }
477 });
478
479 map.fitBounds(bounds, {
480   padding: 20
481 });
482 }
483
484 /**
485 * Converts "lat,lon" array into coordinate object array.
486 * @return the converted Point array object
487 */
488 function stringArrayToPointArray(textArray) {
489   var lonlatArray = new Array();
490   $.each(textArray, function (index, value) {
491     lonlatArray[index] = stringToLonLat(value);
492   });
493   return lonlatArray;
494 }
495
496 /**
497 * Converts "lat,lng" into lonlat array
498 * @return the converted lonlat array
499 */
500 function stringToLonLat(text) {
501   var latlon = text.split(/\s*/);
502   return [parseFloat(latlon[1]), parseFloat(latlon[0])];
503 }
504
505 /**
506 * Swap the inputs
507 */
508 function swapInput() {
509   var startInput = $('#startInput');
510   var finishInput = $('#finishInput');
511   var temp = startInput.val();
512   startInput.val(finishInput.val());
513   finishInput.val(temp);
514   coordinates['start'] = null;
515   coordinates['finish'] = null;
516
517   if (startInput.val() != '' & finishInput.val() != '') {
518     findRouteClicked();
519   }
520 }
521
522 /**
523 * Updates the region information in this page.
524 */

```

```

525     function updateRegion(newRegion, updateCookie) {
526         region = newRegion;
527         if (updateCookie) {
528             setCookie('region', region);
529         }
530         var point = [regions[region].lon, regions[region].lat];
531         map.flyTo({ center: point, zoom: regions[region].zoom, bearing: 0, pitch: 0 });
532     }
533
534     /**
535      * Computes distance between two position (from http://www.movable-type.co.uk/scripts/latlong.html)
536     */
537     function computeDistance(p1, p2) {
538         var R = 6371; // km
539         var p1Lat = p1[1] * Math.PI / 180;
540         var p2Lat = p2[1] * Math.PI / 180;
541         var dLat = (p2[1] - p1[1]) * Math.PI / 180;
542         var dLon = (p2[0] - p1[0]) * Math.PI / 180;
543
544         var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
545             Math.sin(dLon / 2) * Math.sin(dLon / 2) *
546             Math.cos(p1Lat) * Math.cos(p2Lat);
547         var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
548         var d = R * c;
549         return d;
550     }
551 });

```

Kode A.9: protocol.js

```

1 <?php
2 defined('BASEPATH') OR exit('No direct script access allowed');
3 ?>function CicaheumLedengProtocol(apikey, errorHandler) {
4     // IE fix: when window.location.origin is not available
5     if (!window.location.origin) {
6         window.location.origin = window.location.protocol + "://" + window.location.hostname + (window.location.port ? ':' + window.location.port : '');
7     }
8     var HANDLE_URL = window.location.origin + '/api';
9     this.searchPlace = function(query, region, successHandler) {
10        $.ajax({
11            url: HANDLE_URL,
12            type: "GET",
13            data: {
14                mode: "searchplace",
15                version: "3",
16                region: region,
17                apikey: apikey,
18                querystring: query
19            },
20            success: function(data) {
21                successHandler(data);
22            },
23            error: function(jqxhr, textStatus, error) {
24                errorHandler();
25            }
26        });
27    };
28    this.findRoute = function(start, finish, algo, locale, successHandler) {
29        $.ajax({
30            url: HANDLE_URL,
31            type: "GET",
32            data: {
33                mode: "findroute",
34                version: "3",
35                apikey: apikey,
36                start: start,
37                finish: finish,
38                algo: algo,
39                locale: locale,
40                presentation: "desktop"
41            },
42            success: function(data) {
43                successHandler(data);
44            },
45            error: function(jqxhr, textStatus, error) {
46                errorHandler();
47            }
48        });
49    };
50 }

```

Kode A.10: Api.php

```

1 <?php
2 defined('BASEPATH') or exit('No direct script access allowed');
3
4 class Api extends CI_Controller
5 {
6
7     public $cache;
8     public $db;
9     public $Api_model;
10    public $Cache_model;
11

```

```

12|     public function __construct()
13|     {
14|         parent::__construct();
15|         $this->load->model('Api_model');
16|         $this->load->model('Cache_model');
17|         $this->load->config('tirtayasa');
18|         $this->load->config('credentials');
19|
20|
21|     public function index()
22|     {
23|         try {
24|             $version = $this->Api_model->getInput('version', false);
25|             $version = is_null($version) ? 1 : $version;
26|             $apikey = $this->Api_model->getInput('apikey');
27|             $this->Api_model->checkApiKey($apikey);
28|             $mode = $this->Api_model->getInput('mode');
29|
30|             switch ($mode) {
31|                 case 'findroute':
32|                     $this->_findroute($version, $apikey);
33|                     break;
34|                 case 'searchplace':
35|                     $this->_searchplace($version, $apikey);
36|                     break;
37|                 case 'reporterror':
38|                     throw new Exception('501_This_mode_is_no_longer_supported.');
39|                 case 'nearbytransports':
40|                     $this->_nearbytransports($version, $apikey);
41|                     break;
42|                 default:
43|                     throw new Exception('400_Mode_not_understood:' . $mode);
44|             }
45|         } catch (Exception $e) {
46|             $message = $e->getMessage();
47|             $this->Logging_model->logError($message);
48|             if (preg_match('/^([1-5][0-9])_(.+)$/', $message, $matches) === 1) {
49|                 $httpcode = $matches[1];
50|                 $message = $matches[2];
51|             } else {
52|                 $httpcode = '500';
53|             }
54|             $this->Api_model->outputJson(array(
55|                 'status' => 'error',
56|                 'message' => $message
57|             ), $version >= 4 ? $httpcode : '200');
58|         }
59|
60|
61|     public function _findroute($version, $apikey)
62|     {
63|         $start = $this->Api_model->getInput('start');
64|         $finish = $this->Api_model->getInput('finish');
65|         $algo = $this->Api_model->getInput('algo');
66|         $locale = $this->Api_model->getInput('locale');
67|
68|         $language = $this->config->item('languages')[getLocale];
69|         if (is_null($language)) {
70|             throw new Exception("400_Locale_not_found:_$locale");
71|         }
72|         $this->lang->load('tirtayasa', $language['file']);
73|
74|         $presentation = $this->Api_model->getInput('presentation', false);
75|         if (is_null($presentation)) {
76|             $presentation = 'desktop';
77|         }
78|
79|         // Retrieve from menjangan server.
80|         $results = array();
81|         if ($version >= 2) {
82|             $alternatives = $this->config->item('routing-alternatives');
83|             $count = $presentation === 'mobile' ? 1 : sizeof($alternatives);
84|             for ($i = 0; $i < $count; $i++) {
85|                 $url = $this->config->item('url-menjangan') . "?start=$start&finish=$finish&algo=$algo";
86|                 $url .= '&' . 'mw' . '=' . $alternatives[$i]['mw'];
87|                 $url .= '&' . 'wm' . '=' . $alternatives[$i]['wm'];
88|                 $url .= '&' . 'pt' . '=' . $alternatives[$i]['pt'];
89|                 $result = file_get_contents($url);
90|                 if ($result === FALSE) {
91|                     throw new Exception("There's_an_error_while_reading_the_menjangan_response.");
92|                 }
93|                 $results[$result] = true;
94|             }
95|         } else {
96|             $result = file_get_contents($this->config->item('url-menjangan') . "?start=$start&finish=$finish");
97|             if ($result === FALSE) {
98|                 throw new Exception("There's_an_error_while_reading_the_menjangan_response.");
99|             }
100|             $results[$result] = true;
101|         }
102|
103|         foreach ($results as $result => $dummy) {
104|             $travel_time = 0;
105|             $route_output = array();
106|             $steps = explode("\n", $result);
107|             foreach ($steps as $step) {
108|                 $step = trim($step);
109|                 if ($step === '') {
110|                     // Could be the last line, ignore if empty.

```

```

111     continue;
112 }
113 // Path is not found
114 if ($step === 'none') {
115     if (sizeof($results) === 1) {
116         // There is not other alternative
117         $route_output[] = array("none", "none", array($start, $finish), $this->lang->line('Route_not_found'));
118         $travel_time = null;
119         break;
120     } else {
121         // There is alternative, hence we just skip this step.
122         continue 2;
123     }
124 }
125 list($means, $means_detail, $route, $distance, $nearbyplaceids) = explode("/", $step);
126 if (!isset($means) || !isset($route) || !isset($distance) || !isset($nearbyplaceids)) {
127     throw new Exception("Incomplete_response_in_this_line:{$step}");
128 }
129 $points = explode(" ", $route);
130 $from = $points[0];
131 $to = $points[sizeof($points) - 1];
132 // Replace keywords with real location, then construct the detailed path
133 for ($i = 0, $size = sizeof($points); $i < $size; $i++) {
134     if ($points[$i] === 'start') {
135         $points[$i] = $start;
136     }
137     if ($points[$i] === 'finish') {
138         $points[$i] = $finish;
139     }
140 }
141
142 // Construct the human readable form of the walk
143 $humanized_from = $this->Api_model->humanizePoint($from);
144 $humanized_to = $this->Api_model->humanizePoint($to);
145 // Convert whole path to human readable form
146 if ($means === 'walk') {
147     // Remove unnecessary information if not needed.
148     if ($humanized_from === $humanized_to) {
149         // When we're in mobile, skip this step (not really necessary)
150         if ($presentation === 'mobile') {
151             $humanreadable = null;
152         } else {
153             $humanreadable = $this->lang->line('Walk_slightly_at');
154             $humanreadable = str_replace('%street', $humanized_from, $humanreadable);
155             $humanreadable = str_replace('%distance', $this->Api_model->formatDistance($distance), $humanreadable);
156             ;
157         }
158     } else {
159         if ($presentation === 'mobile') {
160             $humanized_from .= '_%fromicon';
161             $humanized_to .= '_%toicon';
162         }
163         $humanreadable = $this->lang->line('Walk_from_to');
164         $humanreadable = str_replace('%from', $humanized_from, $humanreadable);
165         $humanreadable = str_replace('%to', $humanized_to, $humanreadable);
166         $humanreadable = str_replace('%distance', $this->Api_model->formatDistance($distance), $humanreadable);
167     }
168     $travel_time += $distance / $this->config->item('speed-walk');
169     $booking_url = null;
170 } else {
171     $trackDetail = $this->Api_model->getTrackDetails($means, $means_detail);
172
173     // Construct the human readable form of the walk
174     if ($presentation === 'mobile') {
175         $humanized_from .= '_%fromicon';
176         $humanized_to .= '_%toicon';
177     }
178     $humanreadable = $this->lang->line('Take_public_transport');
179     $humanreadable = str_replace('%from', $humanized_from, $humanreadable);
180     $humanreadable = str_replace('%to', $humanized_to, $humanreadable);
181     $humanreadable = str_replace('%distance', $this->Api_model->formatDistance($distance), $humanreadable);
182     $humanreadable = str_replace('%trackname', $trackDetail->trackName, $humanreadable);
183     $humanreadable = str_replace('%tracktype', $trackDetail->trackTypeName, $humanreadable);
184
185     $travel_time += $distance / intval($trackDetail->speed);
186     if (!is_null($trackDetail->ticketURL) && !is_null($trackDetail->extraParameters)) {
187         $booking_url = $trackDetail->ticketURL . $trackDetail->extraParameters;
188     } else {
189         $booking_url = null;
190     }
191
192     // compatibility patch for older 3rd party apps
193     if ($means === 'bdo_angkot' && $version < 3) {
194         $means = 'angkot';
195     }
196     if (!is_null($humanreadable)) {
197         $route_output[] = array($means, $means_detail, $points, $humanreadable, $booking_url);
198     }
199 }
200 $routing_result['steps'] = $route_output;
201 $routing_result['traveltime'] = $this->Api_model->formatTravelTime($travel_time);
202 $routing_results[] = $routing_result;
203 }
204
205 //input log statistic
206 $this->Logging_model->logStatistic($apikey, 'FINDROUTE', "$start/$finish/" . sizeof($results));
207
208 if (!is_null($version) && $version >= 2) {

```

```

209         $json_output = array(
210             'status' => 'ok',
211             'routingresults' => $routing_results
212         );
213     } else {
214         $json_output = array(
215             'status' => 'ok',
216             'routingresult' => $routing_results[0]['steps'],
217             'traveltime' => $routing_results[0]['traveltime']
218         );
219     }
220     $this->Api_model->outputJson($json_output);
221 }
222
223 public function _searchplace($version, $apikey)
224 {
225     $querystring = $this->Api_model->getInput('querystring');
226     $region = $this->Api_model->getInput('region', $version >= 2);
227     $region = is_null($region) ? 'bdo' : $region;
228
229     // Check if there is region modifier from the query string
230     $regions = $this->config->item('regions');
231     foreach ($regions as $key => $value) {
232         if (preg_match('/' . $value['searchplace_regex'] . '/i', $querystring, $matches, PREG_OFFSET_CAPTURE)) {
233             $region = $key;
234             $querystring = substr($querystring, 0, $matches[0][1]);
235             break;
236         }
237     }
238
239     $querystring = urlencode($querystring);
240     $cached_searchplace = $this->Cache_model->get('searchplace', "$region/$querystring");
241     if (!is_null($cached_searchplace)) {
242         $json_output = json_decode($cached_searchplace, true);
243         $this->Logging_model->logStatistic("$apikey", "SEARCHPLACE", "$querystring/cache");
244     } else {
245         $city_lat = $regions[$region]['lat'];
246         $city_lon = $regions[$region]['lon'];
247         $city_radius = $regions[$region]['radius'];
248         $full_url = $this->config->item('url-searchplace') . '?key=' . $this->config->item('google-server-key') . "&input=" .
249             $querystring&inputtype=textquery&locationbias=circle:$city_radius@$city_lat,$city_lon&fields=name,geometry";
250         $result = file_get_contents($full_url);
251         if ($result === FALSE) {
252             throw new Exception("There's an error while reading the places response ($full_url).");
253         }
254
255         $json_result = json_decode($result, true);
256         if ($json_result['status'] === 'OK' || $json_result['status'] === 'ZERO_RESULTS') {
257             $search_result = array();
258             if ($json_result['status'] === 'ZERO_RESULTS') {
259                 $this->Logging_model->logError("Place_search_not_found:$querystring");
260                 $size = 0;
261             } else {
262                 $size = min(sizeof($json_result['candidates']), $this->config->item('searchplace-maxresult'));
263             }
264             for ($i = 0; $i < $size; $i++) {
265                 $current_venue = $json_result['candidates'][$i];
266                 $search_result[$i]['placename'] = $current_venue['name'];
267                 $search_result[$i]['location'] = sprintf(
268                     '%.5f,.5f',
269                     $current_venue['geometry']['location']['lat'],
270                     $current_venue['geometry']['location']['lng']
271                 );
272             }
273             $json_output = array(
274                 'status' => 'ok',
275                 'searchresult' => $search_result,
276                 'attributions' => isset($json_result['html_attributions']) ? $json_result['html_attributions'] : []
277             );
278
279             //input log statistic
280             $this->Logging_model->logStatistic("$apikey", "SEARCHPLACE", "$querystring/$size");
281             // Store to cache
282             if ($size > 0) {
283                 $this->Cache_model->put('searchplace', "$region/$querystring", json_encode($json_output));
284             }
285         } else {
286             throw new Exception('Place_Search_returned_error:' . $json_result['status'] . "(for this request:$full_url)");
287         }
288     }
289     $this->Api_model->outputJson($json_output);
290 }
291
292 public function _nearbytransports($version, $apikey)
293 {
294     $start = $this->Api_model->getInput('start');
295     if ($version >= 2) {
296         $lines = explode("\n", file_get_contents($this->config->item('url-menjangan') . "/?start=$start"));
297         $nearby_result = array();
298         foreach ($lines as $line) {
299             if (strlen(trim($line)) === 0) {
300                 continue;
301             }
302             list($trackTypeId, $trackId, $distance) = explode("/", $line);
303             $trackDetail = $this->Api_model->getTrackDetails($trackTypeId, $trackId);
304             $nearby_result[] = array(
305                 $trackTypeId,
306                 $trackId,
307                 $trackDetail->trackName,
308             );
309         }
310     }
311 }

```

```
307             $distance
308         );
309     }
310     usort($nearby_result, "_nearbytransports_result_compare");
311     $this->Logging_model->logStatistic($apikey, "NEARBYTRANSPORTS", "$start/" . sizeof($nearby_result));
312     $json_output = array(
313         'status' => 'ok',
314         'nearbytransports' => $nearby_result
315     );
316     $this->Api_model->outputJson($json_output);
317 } else {
318     throw new Exception("400_Nearby_transit_is_not_supported_in_version_1._Use_higher_version");
319 }
320 }
321 }
322 /**
323 * A sorting comparison function to be used in nearby transports
324 * @param array $a an array, where index 3 is the distance
325 * @param array $b an array, where index 3 is the distance
326 * @return number as in usort() spec
327 */
328 function _nearbytransports_result_compare($a, $b)
329 {
330     if ($a[3] > $b[3]) {
331         return +1;
332     } elseif ($a[3] < $b[3]) {
333         return -1;
334     } else {
335         return 0;
336     }
337 }
```