KERJA PRAKTIK - IF184801

Pengembangan Algoritma Vehicle Routing Problem dan Implementasi API MapBox

PT Gruu Tumbuh Bersama
Jl. Pondok Jaya VIII No. 10B, Jakarta Selatan, DKI Jakarta 12720
Periode: 21 Februari 2022 - 31 Mei 2022

Oleh:

Ryan Garnet Andrianto 05111940000063

Pembimbing Departemen
Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.
Pembimbing Lapangan
Irvin Nathaniel Tobing

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2023

KERJA PRAKTIK – IF184801

Pengembangan Algoritma Vehicle Routing Problem dan Implementasi API MapBox

PT Gruu Tumbuh Bersama
Jl. Pondok Jaya VIII No. 10B, Jakarta Selatan, DKI Jakarta 12720
Periode: 21 Februari 2023 – 31 Mei 2020

Oleh:

Ryan Garnet Andrianto 05111940000063

Pembimbing Departemen
Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.

Pembimbing Lapangan
Irvin Nathaniel Tobing

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2023

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN
KERJA PRAKTIK

Pengembangan Algoritma Vehicle Routing Problem dan Implementasi API MapBox

Oleh:

Ryan Garnet Andrianto 05111940000063

Disetujui oleh Pembimbing Kerja Praktik:

1. Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.
   NIP. 198106222005012002

(Pembimbing Departemen)

2. Irvin Nathaniel Tobing

(Pembimbing Lapangan)

[Halaman ini sengaja dikosongkan]

Pengembangan Algoritma Vehicle Routing Problem dan Implementasi API MapBox

Nama Mahasiswa	: Ryan Garnet Andrianto
NRP	: 05111940000063
Departemen	: Teknik Informatika FTEIC-ITS
Pembimbing Departemen	: Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.
Pembimbing Lapangan	: Irvin Nathaniel Tobing

ABSTRAK

PT Gruu Tumbuh Bersama merupakan perusahaan yang bergerak di bidang logistik. Perusahaan tersebut membutuhkan sebuah aplikasi untuk menentukan rute terbaik dalam mengantarkan sejumlah barang. Setiap titik antar mempunyai variabel jarak, waktu tempuh, dan berat barang yang diantar.

Aplikasi API Penentu Rute Terbaik dibuat untuk mengatasi masalah penentuan rute sedemikian hingga biaya transportasi menjadi efisien. Biaya transportasi menjadi efisien dalam arti yaitu semua barang berhasil diantar dengan urutan yang tepat sehingga nilai akumulasi jarak dan waktu tempuh menjadi minimum.

Metode nearest neighbour algorithm dan simulated annealing menjadi solusi tahap awal dari aplikasi API Penentu Rute Terbaik yang berhasil diimplementasikan dalam program Python dan lolos tahap pengujian.

Kata Kunci : Nearest Neighbour Algorithm, Simulated Annealing, Program Python

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT atas penyertaan dan karunia-Nya sehingga penulis dapat menyelesaikan salah satu kewajiban penulis sebagai mahasiswa Departemen Teknik Informatika ITS yaitu Kerja Praktik yang berjudul: Pengembangan Algoritma *Vehicle Routing Problem* dan Implementasi API MapBox.

Penulis menyadari bahwa masih banyak kekurangan baik dalam melaksanakan kerja praktik maupun penyusunan buku laporan kerja praktik ini. Namun penulis berharap buku laporan ini dapat menambah wawasan pembaca dan dapat menjadi sumber referensi.

Melalui buku laporan ini penulis juga ingin menyampaikan rasa terima kasih kepada orang-orang yang telah membantu menyusun laporan kerja praktik baik secara langsung maupun tidak langsung antara lain:

1. Kedua orang tua penulis.
2. Ibu Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. selaku dosen pembimbing kerja praktik.
3. Bapak Irvin Nathaniel Tobing selaku pembimbing lapangan selama kerja praktik berlangsung.
4. Teman-teman penulis yang senantiasa memberikan semangat ketika penulis melaksanakan KP.

Surabaya, 28 Februari 2023

Ryan Garnet Andrianto

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1. Latar Belakang

Saat ini dunia telah berkembang menjadi era digital. Semua layanan dapat disajikan secara online dengan adanya teknologi yang dapat dimanfaatkan untuk mempermudah manusia dalam melakukan layanan tersebut. Bila dibandingkan dengan yang dulu, kita melakukan navigasi dengan membeli peta fisik dan kompas. Kini, untuk melakukan navigasi kita dapat membuka website atau aplikasi penyedia peta untuk melakukannya. Proses navigasi peta dapat tidak hanya memperhatikan jarak terpendek melainkan dapat memperhatikan faktor-faktor lain seperti tingkat kemacetan jalan dan jumlah barang yang harus diantar.

Setiap hari suatu penjual dapat mengantarkan sejumlah paket kepada pembelinya. Jumlah paket yang diantarkan dalam satu waktu dapat berjumlah lebih dari satu paket. Selain itu, ukuran paket juga dapat bervariasi. Navigasi peta ditujukan untuk meminimalisasi biaya transportasi. Dengan demikian, penentuan rute pada peta tidak hanya dipengaruhi oleh jarak saja melainkan faktor jumlah paket dan ukuran paket. Pada saat KP, saya diberi kesempatan untuk merancang dan mengimplementasikan algoritma *vehicle routing problem* (VRP) dan mengimplementasikan API MapBox.

1.2. Tujuan

Tujuan kerja praktik ini adalah menyelesaikan kewajiban nilai kerja praktik sebesar 2 sks dan membantu PT Gruu Tumbuh Bersama untuk menyelesaikan permasalahan penentuan rute terbaik dalam bentuk API.

1.3. Manfaat

Manfaat yang diperoleh dengan adanya API penentuan rute terbaik antara lain adalah meminimalisasi biaya transportai dan melihat rute secara online. Jadi penentuan rute tidak dilakukan secara manual. Penentuan rute dilakukan cukup dengan mengisi parameter input API.

1.4. Rumusan Masalah

Rumusan masalah dari kerja praktik ini adalah sebagai berikut.

1. Bagaimana arsitektur server yang dapat memberikan layanan aplikasi API Penentuan Rute Terbaik?
2. Bagaimana implementasi *Nearest Neighbour Algorithm* pada aplikasi API Penentu Rute Terbaik dalam program Python?
3. Bagaimana implementasi *Simulated Annealing* pada aplikasi API Penentu Rute Terbaik dalam program Python?

1.5. Lokasi dan Waktu Kerja Praktik

Sehubungan dengan adanya pandemi dan diberlakukannya *Work From Home*, pengerjaan kerja praktik ini lakukan secara *remote*. Adapun kerja praktik dimulai pada tanggal 21 Februari 2022 hingga 31 Mei 2022.

1.6. Metodologi Kerja Praktik

Metodologi dalam pembuatan buku kerja praktik meliputi:

1.6.1. Perumusan Masalah

Untuk mengetahui kebutuhan dari API, penulis mengikuti pertemuan bersama tim *developer* dan tim riset. Pada pertemuan

tersebut, faktor-faktor dan format *input* serta *output* di bahas. Selain itu, arsitektur dari sistem API pun dibahas.

1.6.2. Studi Literatur

Setelah mendapat gambaran bagaimana sistem tersebut berjalan, penulis diberitahu tinjauan apa saja yang akan diimplementasikan untuk membuat API beroperasi. Tinjauan yang dipakai meliputi FastAPI, *Simulated Annealing*, *Nearest Neighbour Algorithm*, dan *MapBox*. Selain itu, penulis diberitahu aturan-aturan dalam menuliskan konfigurasi agar konfigurasi dapat mudah dipahami oleh *developer* yang lain.

1.6.3. Analisis dan Perancangan Sistem

Setelah tinjauan yang dipakai telah diberitahu, desain arsitektur sistem juga diberitahu.

1.6.4. Implementasi Sistem

Implementasi sistem merupakan realisasi dari tahap perancangan sistem. Pada tahap ini penulis menulis program API Penentu Rute Terbaik dalam bahasa pemrograman Python.

1.6.5. Pengujian dan Evaluasi

Setelah API Penentu Rute Terbaik yang telah direncanakan telah diimplementasikan, pengujian dan evaluasi dilakukan untuk mengetahui apakah API sesuai dengan harapan perusahaan.

1.6.6. Kesimpulan dan Saran

Pengujian yang dilakukan ini telah memenuhi syarat yang diinginkan dan berjalan dengan baik dan lancar.

1.7. Sistematika Laporan

1.7.1. BAB I Pendahuluan

Bab ini berisi latar belakang, tujuan, manfaat, rumusan masalah, lokasi dan waktu kerja praktik, metodologi, dan sistematika laporan.

1.7.2. BAB II Profil Perusahaan

Bab ini berisi gambaran umum PT Gruu Tumbuh Bersama mulai dari profil dan lokasi perusahaan.

1.7.3. BAB III Tinjauan Pustaka

Bab ini berisi dasar teori dari teknologi yang digunakan dalam menyelesaikan proyek kerja praktik.

1.7.4. BAB IV Analisis dan Perancangan Infrastruktur Sistem

Bab ini berisi mengenai tahap analisis sistem aplikasi dalam menyelesaikan proyek kerja praktik.

1.7.5. BAB V Implementasi Sistem

Bab ini berisi uraian tahap - tahap yang dilakukan untuk proses implementasi aplikasi.

1.7.6. BAB VI Pengujian dan Evaluasi

Bab ini berisi hasil uji coba dan evaluasi dari aplikasi yang telah dikembangkan selama pelaksanaan kerja praktik.

1.7.7. BAB VII Kesimpulan dan Saran

Bab ini berisi kesimpulan dan saran yang didapat dari proses pelaksanaan kerja praktik.

BAB II
PROFIL PERUSAHAAN

2.1. Profil PT Gruu Tumbuh Bersama

PT Gruu Tumbuh Bersama merupakan perusahaan logistik yang menyediakan sistem manajemen pengiriman berbasis cloud. Di perusahaan ini, logistik harus efisien, terintegrasi, dan sehat. Perusahaan ini memikirkan bagaimana perusahaan ini dapat memindahkan barang yang tepat ke tempat yang tepat pada waktu yang tepat dengan alat transportasi yang tepat dengan cara yang sederhana. Perusahaan ini memecahkan banyak masalah dan membuat setiap proses dalam logistik menjadi lancar.

2.2. Lokasi

Perusahaan ini berlokasi di Jl. Pondok Jaya VIII No. 10B, Jakarta Selatan, DKI Jakarta 12720.

[Halaman ini sengaja dikosongkan]

BAB III
TINJAUAN PUSTAKA

3.1. Python

Python adalah sebuah bahasa pemrograman *high-level* yang sederhana dan mudah untuk dipelajari sehingga dapat mereduksi biaya *maintenance* dari program. Python mendukung *modules* dan *packages* yang dapat membuatnya menjadi program yang modular dan mempunyai kode yang *reusable*. [1]

3.2. Simulated Annealing

Simulated Annealing adalah sebuah algoritma yang menganalogikan sebuah proses untuk melunakkan atau mengeraskan suatu logam dan kaca dengan memanaskannya pada suhu tinggi dan kemudian secara bertahap mendinginkannya sehingga memungkinkan material mencapai keadaan kristal berenergi rendah. [2]

Pada *Simulated Annealing*, jika suatu perubahan meningkatkan fungsi objektif, maka perubahan tersebut akan diterima. Jika tidak, maka algoritma akan menerima perubahan tersebut dengan sebuah nilai probabilitas yang kurang dari satu. Nilai probabilitas tersebut menurun secara eksponensial yaitu dengan menggunakan properti dari distribusi Boltzmann ($e^{\Delta E/T}$). [2]

3.3. Nearest Neighbour Algorithm

Apabila ada sekumpulan data *item* yang masing-masing memiliki fitur yang bernilai numerik (seperti tinggi, berat, usia, dan lain-lain), maka sebuah klasifikasi sederhana dapat dibuat berdasarkan selisih perbedaan nilai fitur. [3]

Misalnya, pada sebuah graf ada beberapa *edge* yang menghubungkan dua buah *vertex*. Setiap *edge* mempunyai fitur

yaitu jarak tempuh. Dengan *start vertex* a_0, sebuah urutan pengunjungan semua *vertex* dapat dibuat dengan mengambil jarak tempuh yang terpendek dari a_i menuju ke a_j dengan $i < j$ dan $a_i, a_j \in V$.

3.4. FastAPI

FastAPI adalah sebuah web framework yang modern dan cepat (*high-performance*) untuk membangun sebuah API dengan Python 3.7+. FastAPI merupakan salah satu *web framework* yang *robust* dan mudah untuk digunakan. Selain itu, FastAPI mengadopsi *open standard* untuk OpenAPI dan JSON Schema. [4]

BAB IV
ANALISIS DAN PERANCANGAN INFRASTRUKTUR SISTEM

4.1. Analisis Sistem

Pada bab ini akan dijelaskan mengenai tahapan dalam membangun infrastruktur aplikasi API Penentu Rute Terbaik yaitu analisis dari infrastruktur sistem yang akan dibangun. Hal tersebut dijelaskan ke dalam dua bagian, definisi umum aplikasi dan analisis kebutuhan.

4.1.1. Definisi Umum Aplikasi

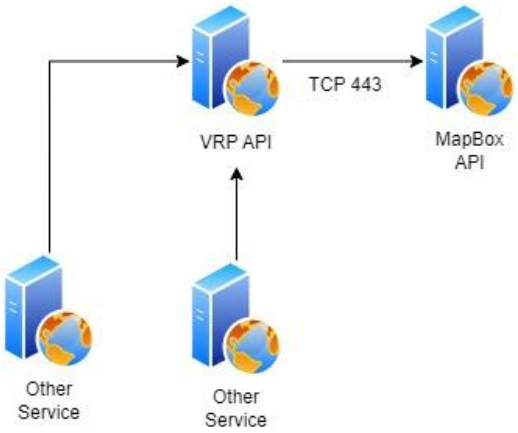
Secara umum, aplikasi API Penentu Rute Terbaik merupakan sistem yang menerima *input* berupa data *depot* dan data titik antar dan memberikan *output* berupa rute perjalanan yang terbaik. Terbaik berarti menggunakan biaya transportasi yang seminimum mungkin dengan tetap memperhatikan batasan-batasan pengantaran.

Batasan-batasan pengataran yang ada adalah batas waktu tempuh dalam satu rute dan batas kapasitas kendaraan. Batas waktu tempuh dalam satu rute berarti dilarang ada sebuah rute dengan total waktu tempuh melebihi nilai yang telah ditentukan.

4.2. Perancangan Infrastruktur Sistem

4.2.1. Desain Arsitektur Sistem

Desain arsitektur sistem pada API Penentu Rute Terbaik menggunakan 2 service. Service yang digunakan adalah Server VRP API dan Server API MapBox. Desain arsitektur sistem dapat dilihat pada Gambar 1.



Gambar 1 Desain Arsitektur Sistem

Gambar 1 merupakan desain arsitektur sistem yang digunakan pada API Penentu Rute Terbaik. Satu server Ubuntu 20.04 digunakan sebagai VRP API yaitu tempat terinstallnya FastAPI. VRP API mengambil data geografis dari MapBox API dan mengubah data tersebut menjadi informasi dengan menggunakan algoritma VRP.

4.2.2. Desain API

API Penentu Rute Terbaik mempunyai tugas untuk menentukan rute pengantaran barang dari *depot* menuju titik antar. Rute yang ditentukan harus mempunyai jarak yang minimum dan waktu yang minimum. Rute yang ditentukan juga harus memenuhi batasan kapasitas dari kendaraan. Adapun *input* dan *output* dari API ini ditunjukkan oleh Tabel 1 dan Tabel 2.

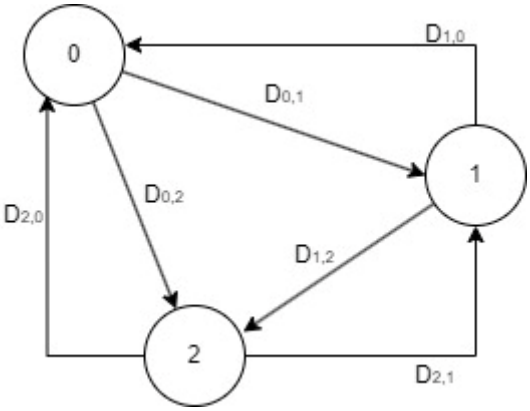
Tabel 1 Input API Penentuan Rute Terbaik

No.	**Nama *Input***	**Deskripsi**
1.	*back_to_depot*	Jika bernilai *true*, maka setiap rute harus kembali ke depot, *vice versa*.
2.	*capacity*	Kapasitas dari setiap kendaraan. Semua kendaraan dianggap mempunyai kapasitas yang sama.
3.	*price_per_km*	Harga per kilometer yang diasumsikan pada transportasi.
4.	*depot_coord*	Posisi koordinat dari *depot* yang berupa *latitude* dan *longtitude*.
5.	*depot_id*	ID yang *unique* yang digunakan sebagai *identifier* dari depot.
6.	*max_travel_time*	Batas waktu tempuh per rute.
7.	*orders*	Data titik antar yang meliputi ID order yang *unique*, koordinat titik antar, dan berat barang yang diantar.

Tabel 2 Output API Penentu Rute Terbaik

No.	Nama Output	Deskripsi
1.	*total_mileage*	Total jarak tempuh akumulasi semua rute.
2.	*deliveries*	Data rute terbaik meliputi bentuk geometri rute pada peta dan jarak tempuh rute.
3.	*impossible_order*	Data titik antar yang tidak dapat diantar karena melanggar batasan.

Setiap titik antar direpresentasikan sebagai *vertex* pada suatu *directed graph*. Depot menjadi *vertex* ID 0 sehingga terdapat $n(orders) + 1$ *vertex*. Jalur-jalur yang dapat digunakan oleh rute direpresentasikan sebagai *edge*. Gambar 2 menunjukkan contoh representasi *graph* pada API Penentu Rute Terbaik.



Gambar 2 Contoh Representasi Graf pada API Penentu Rute Terbaik

Setiap *edge* mempunyai nilai *weight* yang terdiri dari jarak antara *start vertex* dan *finish vertex* dan waktu tempuh dari *start vertex* dan *finish vertex*. Jarak dan waktu tempuh direpresentasikan

dalam bentuk matriks 2 dimensi seperti yang ditunjukkan oleh Persamaan (1) dan Persamaan (2).

$$D = \begin{bmatrix} D_{0,0} & D_{0,1} & \dots & D_{0,n-1} \\ D_{1,0} & D_{1,1} & \dots & D_{1,n-1} \\ \dots & \dots & \dots & \dots \\ D_{n-1,0} & D_{n-1,1} & \dots & D_{n-1,n-1} \end{bmatrix} \quad (1)$$

$$T = \begin{bmatrix} T_{0,0} & T_{0,1} & \dots & T_{0,n-1} \\ T_{1,0} & T_{1,1} & \dots & T_{1,n-1} \\ \dots & \dots & \dots & \dots \\ T_{n-1,0} & T_{n-1,1} & \dots & T_{n-1,n-1} \end{bmatrix} \quad (2)$$

Data $D_{i,j}$ merepresentasikan jarak antara *vertex* ke-i dengan *vertex* ke-j, sedangkan data $T_{i,j}$ merepresentasikan waktu tempuh dari *vertex* ke-i menuju *vertex* ke-j. Berdasarkan observasi sederhana dari sistem lalu lintas di Indonesia, ada beberapa kesimpulan yang dapat diambil:

1. $D_{i,j} = 0$ untuk $i = j$ sebab sudah pasti tidak ada jalur yang menghubungkan titik antar yang sama. Dalam kata lain, tidak ada *edge* yang menghubungkan dua *vertex* yang sama.
2. $D_{i,j} \neq D_{j,i}$ untuk $0 \leq i, j < n(V)$ karena perhitungan jarak tempuh antara titik antar menggunakan *manhattan distance* sehingga ada kemungkinan bahwa jarak tempuh titik antar ke-i menuju ke titik antar ke-j tidak sama dengan jarak tempuh titik antar ke-j menuju ke titik antar ke-i. Hal ini disebabkan oleh adanya jalan yang hanya dapat dilalui satu arah di Indonesia.
3. $T_{i,j} = 0$ untuk $i = j$ karena waktu tempuh yang dibutuhkan untuk mengantarkan barang dari posisi *start* dan *finish* yang sama adalah 0.
4. $T_{i,j} \neq T_{j,i}$ untuk $0 \leq i, j < n(V)$ karena $D_{i,j} \neq D_{j,i}$.

Nilai dari $T_{i,j}$ dihitung dengan mempertimbangkan faktor kepadatan lalu lintas dan jarak tempuh. Dengan demikian, apabila $D_{a,b} < D_{c,d}$ belum tentu $T_{a,b} < T_{c,d}$ untuk $a, b, c, d \in V$. Nilai dari variabel $T_{i,j}$ dihitung oleh MapBox API.

Data matriks D dan matriks T didapatkan dari API MapBox dengan cara mengirimkan *TCP Request* dengan port 443 (SSL) ke URI API MapBox Directions seperti yang ditunjukkan oleh Gambar 3.

```
GET /directions/v5/{profile}/{coordinates}
HTTP/1.1
Host: api.mapbox.com
```

Gambar 3 Request API ke MapBox

Setelah matriks D dan matriks T diisi dengan data yang didapatkan dari API MapBox. Proses pencarian rute terbaik dilakukan dengan menggunakan dua metode yaitu *Nearest Neighbour Algorithm* dan *Simulated Annealing*.

Nearest Neighbour Algorithm dilakukan untuk mendapatkan *initial route* atau rute mula-mula. Kemudian, *Simulated Annealing* dilakukan untuk mengoptimalisasi *initial route* tersebut. Adapun rancangan algoritma pencarian rute terbaik dapat dilihat pada Pseudocode 1.

Pseudocode 1 Fungsi *do_optimize_route*

```
1: do_optimize_route(Input)
2:   D = getDistanceMatrix()
3:   T = getTimeMatrix()
4:   initRoutes = nearestNeighbour()
5:   for r in initRoutes
6:     r = simulatedAnnealing(r)
7:   geometries = getGeo(initRoutes)
```

```
8:    mileages = calcMileage(initRoutes)
9:    return r
```

Fungsi *nearestNeighbour()* menjalankan algoritma *Nearest Neighbour Algorithm* (NNA) untuk membuat sebuah *initial routes* yang disimpan dalam bentuk himpunan bernama *tour* seperti yang ditunjukkan oleh Persamaan (3).

$$tour = \{t_0, t_1, t_2, t_3, \dots, t_{k-2}, t_{k-1}\}, t_k \in V \quad (3)$$

Himpunan *tour* akan dipecah menjadi subhimpunan-subhimpunan yang bernama *subtour* seperti yang ditunjukkan oleh Persamaan (4).

$$tour = \{subtour_0, subtour_1, \dots, subtour_k\} \quad (4)$$

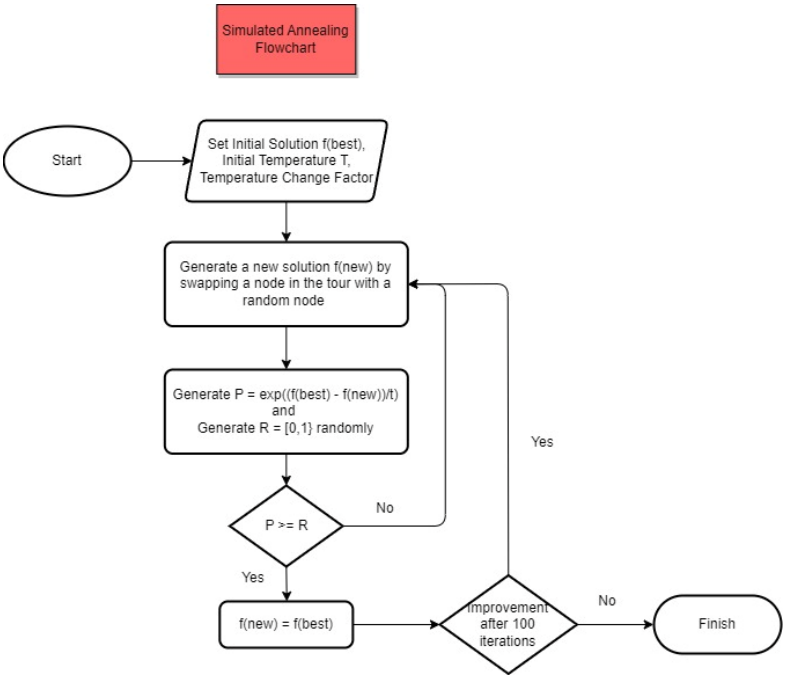
Himpunan *subtour* berisi ID dari *vertex* yang diantar dengan satu kendaraan yang dianggap sebagai *vertex* yang dikunjungi secara sekuensial pada satu rute. Dalam kata lain, jumlah kendaraan yang dibutuhkan untuk mengantarkan semua barang ke titik antar adalah $n(subtour)$. Dengan demikian, ada sejumlah subtotal jarak tempuh per kendaraan dapat dihitung dengan Persamaan (5).

$$sumDistance(S) = \sum_{i=1}^{n(S)-1} D_{s_{i-1},s_i} \quad (5)$$

Persmaan (5) digunakan untuk menghitung total jarak tempuh untuk satu rute. Dalam aplikasi API Penentu Rute Terbaik, rute yang diberikan dapat lebih dari satu sehingga nilai *total_mileage* yang merupakan salah satu *output* dari API dapat dihitung dengan Persamaan (6).

$$total\_mileage = \sum_{subtour}^{subtour \in tour} sumDistance(subtour) \quad (6)$$

Setelah NNA membuat *initial routes* yang disimpan pada himpunan *tour* dan *subtour*. Setiap rute di *initial routes* akan dioptimalisaasi dengan menggunakan *Simulated Annealing* (SA). Gambar 4 menunjukkan *flowchart* dari proses SA.



Gambar 4 Flowchart Proses Simulated Annealing

Fungsi objektif pada proses SA ditunjukkan oleh Persamaan (7). Fungsi ini menjumlahkan jarak tempuh pada

subtour S. Tujuan dari proses SA-nya adalah untuk mendapatkan nilai *subtour S* dengan nilai fungsi objektif yang seminimum mungkin (*global optimum*).

$$f_{obj}(S) = sumDistance(S) \tag{7}$$

Pada setiap iterasi SA, dua elemen dari *subtour* S akan ditukar posisinya seperti yang ditunjukkan oleh Persamaan (8).

$$\begin{aligned} S &= \{s_0, s_1, \dots, s_i, s_{i+1}, \dots, s_k, s_{k+1}, \dots, s_w\}, i < k < w \\ S' &= \{s_0, s_1, \dots, s_k, s_i, s_{i+1}, \dots, s_i, s_{k+1}, \dots, s_w\}, i < k < w \end{aligned} \tag{8}$$

S' adalah subtour setelah penukaran. Setelah dua elemen dari *subtour* S ditukar, maka nilai dari fungsi objektif dibandingkan ($f_{obj}(S') < f_{obj}(S)$). Jika nilai fungsi objektif dari *subtour* S' adalah kurang dari nilai fungsi objektif dari *subtour* S, maka *subtour* S' akan diambil sebagai kandidat baru. Jika tidak, maka nilai *random* dan *metropolis criterion* akan menjadi penentu apakah *subtour* S' akan diambil sebagai kandidat baru atau tidak. *Metropolis criterion* tersebut dihitung dengan Persamaan (9).

$$\begin{aligned} \Delta D &= sumDistance(S') - sumDistance(S) \\ criterion &= e^{-\frac{\Delta D}{T}} \end{aligned} \tag{9}$$

Jika nilai *random* yang mempunyai *range* dari 0.0 sampai dengan 1.0, kurang dari *metropolis criterion*, maka *subtour* S' akan dianggap sebagai kandidat baru.

Pada setiap akhir iterasi *Simulated Annealing*, nilai dari temperatur T diubah dengan menggunakan metode *geometry* seperti yang ditunjukkan oleh Persamaan (10). *geo* adalah faktor perkalian dari temperatur dengan nilai *default* 0.99.

$$T' = T(geo) \tag{10}$$

Iterasi *Simulated Annealing* (SA) terus dilakukan sampai nilai dari *counter* lebih besar dari nilai *max iteration* yakni *100*. Nilai *counter* akan di-*reset* setiap kali ada kandidat *subtour S* baru.

BAB V
IMPLEMENTASI SISTEM

Bab ini membahas tentang implementasi dari sistem yang dibuat. Implementasi ini akan dibagi ke dalam beberapa bagian yaitu installasi library python dan implementasi program.

5.1. Installasi Library Python

Library python yang digunakan untuk membuat sistem ini ditunjukkan oleh Tabel 3.

Tabel 3 Library Python yang Digunakan

No.	Nama Library	Versi	Deskripsi
1.	NumPy	1.21.4	Untuk menyimpan matriks dan melakukan operasi matematis.
2.	FastAPI	0.70.0	Untuk menjalankan *web framework* dari API.

5.2. Implementasi Program

Subbab ini menjelaskan implementasi proses algoritma secara keseluruhan. Hasil dari implementasi ini merupakan program yang digunakan untuk menentukan rute terbaik pada aplikasi API Penentu Rute Terbaik. Berikut adalah penjelasan masing-masing struktur dan fungsi dari implementasi program.

5.2.1. Implementasi fungsi *do_optimize_route*

Fungsi *do_optimize_route* merupakan fungsi utama yang dieksekusi ketika sebuah request API diterima oleh server. Peran

fungsi tersebut pada aplikasi API Penentu Rute Terbaik adalah sebagai *main program*.

Proses yang terjadi pada fungsi *do_optimize_route* secara berurut adalah sebagai berikut.

1. Membuat sebuah array yang berisi data *order_id, latitude, longtitude,* berat barang. Depot direpresentasikan pada data indeks ke-0 dengan berat barang 0 dan *depot_id* sebagai *order_id*.
2. Mengambil data dari API MapBox melalui fungsi *generate_cache_by_nodes_async*. Proses pengambilan data dilakukan dengan *multithreading* dan dilakukan dengan *async.*
3. Menjalankan algoritma *Nearest Neighbour Algorithm* untuk mendapatkan *initial route* melalui fungsi *generate_tour_by_nearest_neighbour*.
4. Menghitung *initial cost* atau mileage mula-mula dan menyimpannya pada variabel *init_cost*. Nilai ini dihitung dengan menggunakan fungsi *count_all_vehicle_distance* yang merupakan implementasi dari Persamaan (6).
5. Menjalankan algoritma *Simulated Annealing* untuk mendapatkan rute yang diekspektasikan lebih baik daripada rute mula-mula yang disimpan pada variabel *best_tour*. Proses ini dilakukan oleh fungsi *vrp_simulated_annealing*.
6. Tour merupakan sebuah himpunan yang menyimpan urutan pengunjungan *vertex* seperti yang telah dijelaskan pada subbab 4.2.2. Setelah *Simulated Annealing* dilakukan dan *best_tour* didapatkan, *best_tour* dipecah atau dibagi kembali menjadi subhimpunan *subtour* yang bernama *best_subtour*. Proses ini dilakukan melalui fungsi *tour_to_subtour*.

7. Dari data *best_subtour*, *total mileage* kembali dihitung.
8. Setelah didapatkan *total mileage* mula-mula dan *total mileage* setelah *simulated annealing* dilakukan. Kedua nilai tersebut dibandingkan. Apabila *tour* baru mempunyai *total mileage* yang lebih kecil, maka *tour* baru diambil sebagai solusi, *vice versa*.
9. Kemudian, *geometry* dibuat dengan menggunakan fungsi *make_geometry* serta perhitungan jarak per rute dihitung dengan menggunakan fungsi *make_fleet* yang merupakan implementasi dari Persamaan (5).
10. Terakhir, *output* dari API disiapkan dengan menggabungkan semua data yang hendak menjadi *output* menjadi satu. Kemudian, data *output* yang berisi informasi akan dikirimkan ke *client*.

Adapun implementasi fungsi tersebut dapat dilihat pada Kode Sumber 1.

```
async def do_optimize_route(request:
OptimizeRouteRequest):
    id_list = [request.depot_id]
    nodes = []
    for index, order in
enumerate(request.orders):
        temp = []

        temp.append(index+1)
        id_list.append(order.order_id)


temp.append(order.coordinate.latitude)
temp.append(order.coordinate.longitude)

        temp.append(order.demand)

        temp.append(order.order_id)
```

```
16
17          nodes.append(temp)
18
19      depot = [0,
request.depot_coord.latitude,
request.depot_coord.longitude, 0,
request.depot_id]
20      cache = await
generate_cache_by_nodes_async(nodes, depot)
21      tour, subtour, impossible_node =
generate_tour_by_nearest_neighbour(cache,
depot, nodes, request.capacity,
request.max_travel_time)
22      init_cost =
count_all_vehicle_distance(cache, subtour,
depot, nodes,
include_return_depot=request.back_to_depot)
23      best_tour =
vrp_simulated_annealing(cache, tour,
request.capacity, nodes, depot, iter=100,
include_return_depot=request.back_to_depot)
24      best_subtour = tour_to_subtour(cache,
best_tour, request.capacity, nodes,
request.max_travel_time)
25      final_cost =
count_all_vehicle_distance(cache,
best_subtour, depot, nodes,
include_return_depot=request.back_to_depot)
26
27      if final_cost < init_cost:
28          result = best_subtour
29          result_cost = final_cost
30      else:
31          result = subtour
32          result_cost = init_cost
33
34      deliveries = []
35      for subtour in result:
```

```
36          geometry = make_geometry(subtour,
cache, request.back_to_depot)
37          fleet = make_fleet(subtour, id_list,
nodes, cache, depot, request.price_per_km,
request.back_to_depot)
38
39          deliveries.append({
40              "geometry": geometry,
41              "fleet": fleet
42          })
43
44      # preparing data for output
45      impossible_order = []
46      for node_id in impossible_node:
47
impossible_order.append(id_list[node_id])
48
49      return OptimizeRouteResponse(
50          total_mileage=result_cost,
51          deliveries=deliveries,
52          impossible_order=impossible_order
53      )
```

Kode Sumber 1 Implementasi Fungsi *do_optimize_route*

5.3. Implementasi Fungsi *find_impossible_node*

Fungsi *find_impossible_node* digunakan untuk mencari *node* mana saja yang tidak mungkin masuk ke dalam kandidat pada proses *nearest neighbour algorithm* sebab waktu tempuh dari depot menuju *node* tersebut adalah lebih besar dari batas total waktu pengantaran yang direpresentasikan oleh variabel *max_travel_time* pada input API. Adapun implementasi fungsi tersebut dapat dilihat pada Kode Sumber 2.

```
1     def find_impossible_node(matrix, depot,
nodes, capacity, max_travel_time):
2         impossible_node = []
```

```
3         for node in nodes:
4             # if node is not depot
5             if node[0] != depot[0]:
6                 travel_time =
get_travel_time(matrix, depot[0], node[0])
7                 if capacity < node[3] or
travel_time > max_travel_time:
8
impossible_node.append(node[0])
9         return impossible_node
```

Kode Sumber 2 Implementasi Fungsi *find_impossible_node*

5.4. Implementasi Fungsi *find_nearest_neighbour_candidate*

Fungsi *find_nearest_neighbour_candidate* digunakan untuk mencari kandidat *node* ketika proses pencarian dengan *nearest neighbour algorithm* dilakukan. Adapun implementasi fungsi tersebut dapat dilihat pada Kode Sumber 3.

```
1 def find_nearest_neighbour_candidate
2     (matrix, nodes, tour,
current_node_index,
3     current_capacity,
current_sum_travel_time,
4     capacity, max_travel_time,
except_nodes):
5     candidate_distance = sys.maxsize
6     candidate_travel_time = 0
7     candidate_index = -1
8
9     for i, node in enumerate(nodes):
10        # ignore depot
11        if node[0] == 0:
12            continue
13        # ignore except nodes
14        if node[0] in except_nodes:
15            continue
16        # ignore visited nodes
```

```
17          if node[0] in tour:
18              continue
19
20          travel_time =
get_travel_time(matrix,
21              current_node_index, node[0])
22          if current_capacity + node[3] <=
capacity and
23              current_sum_travel_time +
travel_time <= max_travel_time:
24              temp_distance = count_distance(
25                  matrix,
nodes[current_node_index], node)
26              if temp_distance <
candidate_distance:
27                  candidate_distance =
temp_distance
28                  candidate_travel_time =
travel_time
29                  candidate_index = i
30
31      return candidate_index,
candidate_travel_time
```

Kode Sumber 3 Implementasi Fungsi *find_nearest_neighbour_candidate*

5.5. Implementasi Fungsi *generate_tour_by_nearest_neighbour*

Fungsi *generate_tour_by_nearest_neighbour* digunakan untuk mencari *initial routes*. Adapun implementasi dari fungsi tersebut dapat dilihat pada Kode Sumber 4.

```
1 def generate_tour_by_nearest_neighbour
2       (matrix, depot, node, capacity,
max_travel_time):
3       nodes = node.copy()
4       nodes.insert(0, depot)
5
```

```
6       tour = []
7       subtour = []
8       impossible_nodes = find_impossible_node(
9           matrix, depot, nodes, capacity,
max_travel_time)
10
11      # depot is at index 0
12      current_node_index = 0
13      current_capacity = 0
14      current_sum_travel_time = 0
15      current_tour = []
16
17      while True:
18          # find nearest neighbour candidate
19          candidate_node_index,
20          candidate_node_travel_time =
find_nearest_neighbour_candidate(
21              matrix, nodes, tour,
current_node_index, current_capacity,
22              current_sum_travel_time,
capacity, max_travel_time,
23              impossible_nodes)
24
25          # if there is no candidate, then
26          if candidate_node_index == -1:
27              # if no node is visited, stop
the loop
28              if len(current_tour) == 0:
29                  break
30              else:
31                  subtour.append(current_tour)
32                  current_tour = []
33                  current_node_index = 0
34                  current_capacity = 0
35                  current_sum_travel_time = 0
36          else:
37              candidate_node_id =
nodes[candidate_node_index][0]
```

```
38              candidate_node_capacity =
nodes[candidate_node_index][3]
39
40              tour.append(candidate_node_id)
41
current_tour.append(candidate_node_id)
42              current_node_index =
candidate_node_index
43              current_capacity +=
candidate_node_capacity
44              current_sum_travel_time +=
candidate_node_travel_time
45
46      if len(current_tour):
47          subtour.append(current_tour)
48
49      return tour, subtour, impossible_nodes
```

Kode Sumber 4 Implementasi Fungsi *generate_tour_by_nearest_neighbour*

5.6. Implementasi Fungsi *vrp_simulated_annealing*

Fungsi *vrp_simulated_annealing* merupakan fungsi yang digunakan untuk melakukan proses *simulated annealing*. Adapun implementasi dari fungsi tersebut dapat dilihat pada Kode Sumber 5.

```
1 def vrp_simulated_annealing
2       (matrix, tour, capacity, nodes, depot,
iter=2000,
3       geo=0.99, temp=1000,
include_return_depot=True,
4       max_travel_time=10800):
5
6       if len(tour) == 1:
7           return tour
8       else:
9           best_tour = tour.copy()
```

```
10
11          stop = False
12          counter = 0
13          best_distance =
tour_to_all_vehicle_distance(
14              matrix, best_tour, capacity,
nodes, depot,
15              include_return_depot,
max_travel_time)
16
17          curr_tour, curr_distance =
best_tour.copy(), best_distance
18          it_num = 0
19          while not stop:
20              it_num += 1
21              # make a candidate
22              [x, y] =
sorted(random.sample(range(len(best_tour)),
2))
23              candidate_tour =
curr_tour.copy()
24              candidate_tour[x],
candidate_tour[y] = candidate_tour[y],
candidate_tour[x]
25              candidate_distance =
tour_to_all_vehicle_distance(
26                  matrix, candidate_tour,
capacity, nodes, depot,
27                  include_return_depot,
max_travel_time)
28              # if candidate solution is
better, save it
29              if candidate_distance <
best_distance:
30                  best_tour, best_distance =
candidate_tour.copy(), candidate_distance
31              # difference between candidate
and current
```

```
32              diff = candidate_distance -
curr_distance
33              # calculate metropolis
acceptance criterion
34              pow_val = -diff/temp
35              metropolis = 1.0
36              if pow_val < 0:
37                  metropolis = np.exp(-
diff/temp)
38              # check metropolis
39              if diff < 0 or np.random.rand()
< metropolis:
40                  curr_tour, curr_distance =
candidate_tour.copy(), candidate_distance
41                  counter = 0
42              else:
43                  counter += 1
44              # calculate new temperature
45              temp = temp * geo
46              # if there is no improvement
after iter iteration, then stop
47              if counter >= iter:
48                  stop = True
49          return best_tour
```

Kode Sumber 5 Implementasi Fungsi *vrp_simulated_annealing*

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini menjelaskan tahap uji coba terhadap Aplikasi API Penentu Rute Terbaik. Pengujian dilakukan untuk memastikan fungsionalitas dan kesesuaian hasil implementasi arsitektur dengan analisis dan perancangan arsitektur.

6.1. Tujuan Pengujian

Pengujian dilakukan terhadap Aplikasi API Penentuan Rute Terbaik guna menguji kemampuan sistem dalam melayani permintaan sistem aplikasi.

6.2. Kriteria Pengujian

Penilaian atas pencapaian tujuan pengujian didapatkan dengan memperhatikan beberapa hasil yang diharapkan yaitu kemampuan sistem untuk memberikan *output* yang sesuai dengan *input* yang diberikan.

6.3. Skenario Pengujian

Skenario pengujian dilakukan dengan menggunakan aplikasi Postman. Langkah-langkah skenario pengujian adalah sebagai berikut.

1. Isi URI di aplikasi Postman sesuai dengan URI dari server pengujian (https://IP:Port).
2. Masukkan *input* dalam format JSON. Detail *input* pada pengujian dapat dilihat pada Lampiran A.
3. Cek kebenaran hasil *output* dari API dengan cara membandingkannya dengan hasil *output* yang telah dihitung secara manual.

6.4. Evaluasi Pengujian

Hasil pengujian dilakukan terhadap kasus skenario uji coba. Tabel 4 di bawah ini menjelaskan hasil uji coba terhadap aplikasi yang telah dibuat. Adapun dokumentasi pengujian dapat dilihat pada Lampiran A.

Tabel 4 Hasil Evaluasi Pengujian

Kriteria Pengujian	**Hasil Pengujian**
API dapat diakses tanpa kendala	Terpenuhi
Output untuk uji coba *input* ke-1 adalah benar.	Terpenuhi
Output untuk uji coba *input* ke-2 adalah benar.	Terpenuhi
Output untuk uji coba *input* ke-3 adalah benar.	Terpenuhi

BAB VII
KESIMPULAN DAN SARAN

7.1. Kesimpulan

Kesimpulan yang didapat setelah melakukan perancangan arsitektur sistem aplikasi API Penentu Rute Terbaik pada kegiatan kerja praktek di PT Gruu Tumbuh Bersama adalah sebagai berikut.

a. Arsitektur server yang dapat memberikan layanan aplikasi API Penentuan Rute Terbaik telah berhasil didesain dan diimplementasikan.
b. Implementasi *Nearest Neighbour Algorithm* pada aplikasi API Penentu Rute Terbaik telah berhasil didesain dan diimplementasikan dalam program Python.
c. Implementasi *Simulated Annealing* pada aplikasi API Penentu Rute Terbaik telah berhasil didesain dan diimplementasikan dalam program Python.

7.2. Saran

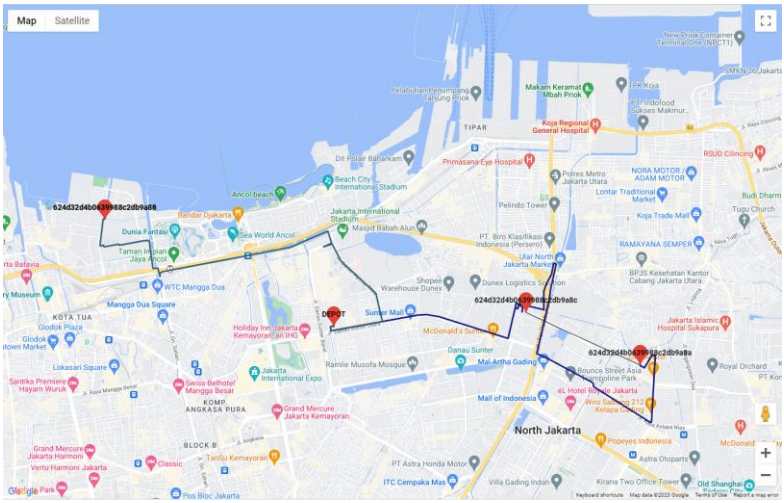
Saran untuk perancangan arsitektur sistem aplikasi API Penentu Rute Terbaik adalah sebagai berikut.

a. Jika di masa yang akan datang jumlah *request* yang masuk ke server API meningkat, maka ubah arsitektur dari server API menjadi *distributed server* yang membagi beban *request* secara merata ke semua *node* server.
b. Setiap kali sebuah *request* dikirimkan ke MapBox API, pemilik *key* yaitu perusahaan akan dikenai biaya. Ada kemungkinan bahwa *request* yang sama dikirimkan lebih dari satu kali. Dengan mengimplemntasikan *cache*, biaya penggunaan API MapBox dapat menjadi lebih terminimalisasi.

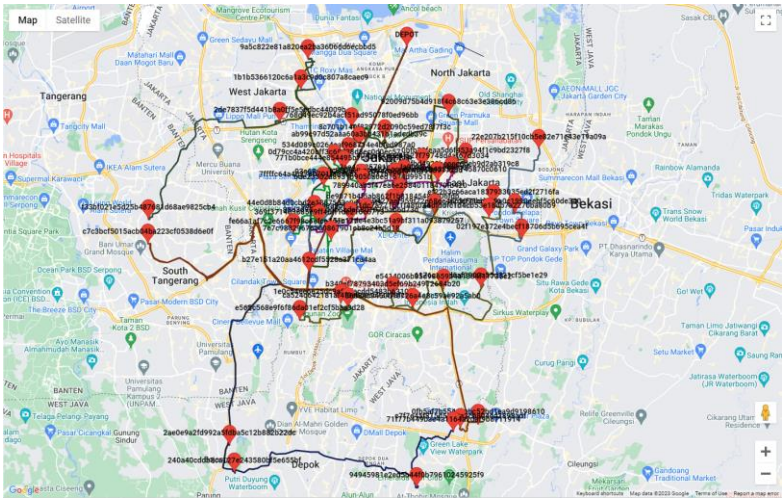
[Halaman ini sengaja dikosongkan]

LAMPIRAN A

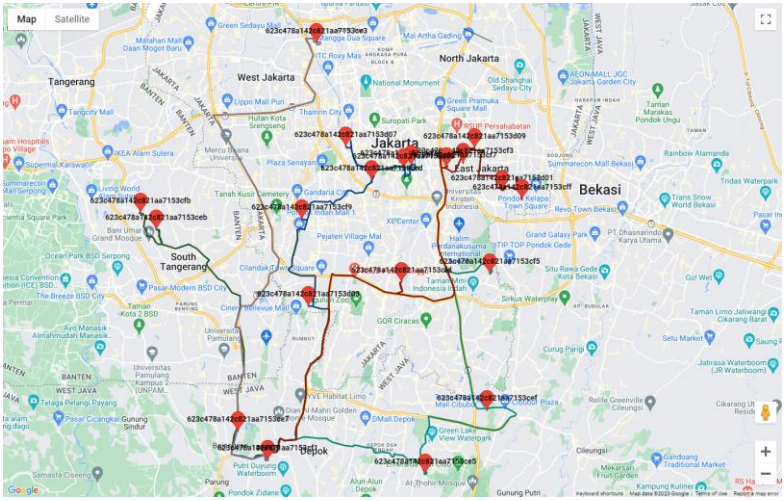
Berikut adalah dokumentasi pengujian API Penentu Rute Terbaik.



Gambar A.1 *Output* Uji Coba *Input* Ke-1



Gambar A.2 *Output* Uji Coba *Input* Ke-2



Gambar A.3 *Output* Uji Coba *Input* Ke-3

DAFTAR PUSTAKA

[1] "What is Python? Executive Summary," [Online]. Available: https://www.python.org/doc/essays/blurb. [Accessed 16 2 2023].

[2] P. Norvig and S. J. Russell, Artificial Intelligence A Modern Approach Fourth Edition, Hoboken: Pearson, 2021.

[3] "GeeksForGeeks," [Online]. Available: https://www.geeksforgeeks.org/implementation-k-nearest-neighbors. [Accessed 16 2 2023].

[4] "FastAPI," [Online]. Available: https://fastapi.tiangolo.com. [Accessed 16 2 2023].

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Nama	: Ryan Garnet Andrianto
Tempat, Tanggal Lahir	: Surabaya, 5 Februari 2001
Jenis Kelamin	: Laki-laki
Telepon	: +62 877 5082 7825
E-mail	: ryangarnetandrianto@gmail.com

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC , ITS
Angkatan	: 2019
Semester	: 8 (Delapan)