# **Mengoptimalkan Rute Pengiriman Barang di Perkotaan dengan Sistem Berbasis Web: Kupas Tuntas Konsep, Algoritma, dan Rancangan Sistemnya**

## **I. Mari Kita Mulai: Pengantar**

### **A. Kenapa Ini Penting? (Latar Belakang)**

Bicara soal bisnis, urusan logistik itu ibarat jantungnya. Kalau efisien, bisnis bisa berjalan mulus, terutama buat teman-teman Usaha Mikro, Kecil, dan Menengah (UMKM) yang berjibaku di tengah padatnya kota. Mengirim barang dari satu titik pusat (entah itu gudang atau toko penjual) ke banyak pelanggan di berbagai sudut kota itu punya tantangan tersendiri. Kalau masih pakai cara manual buat nentuin rute, wah, bisa makan waktu, boros, dan pusing sendiri kalau harus mikirin kapasitas angkut mobil, mana yang harus dikirim duluan, atau jam berapa pelanggan maunya barang sampai.1 Akibatnya? Biaya operasional bisa membengkak, terutama buat bensin dan bayar waktu sopir, plus pelanggan bisa kecewa kalau barangnya telat.1

Nah, tantangan buat ngatur rute terbaik ini, kalau di dunia akademis atau riset operasi, punya nama keren: *Vehicle Routing Problem* (VRP).7 Masalah VRP ini, plus variasinya, itu kayak teka-teki super rumit di dunia matematika optimasi. Saking rumitnya, sampai dikasih label 'NP-hard'. Artinya, nyari solusi paling sempurna itu susah banget kalau jumlah pelanggannya banyak atau aturannya makin ribet.8 Tapi tenang, teknologi komputasi dan algoritma pintar terus berkembang. Makanya, bikin sistem optimasi rute yang gampang diakses lewat web bisa jadi solusi jitu buat UMKM dan bisnis lokal lainnya. Harapannya, sistem ini bisa otomatis merencanakan rute, mempertimbangkan segala macam aturan main, dan akhirnya ngasih rute yang lebih hemat, biar biaya bisa ditekan dan bisnis makin kompetitif.13

### **B. Apa Saja yang Ingin Kita Jawab? (Rumusan Masalah)**

Lewat penelitian dan pengembangan sistem ini, kita mau cari jawaban buat pertanyaan-pertanyaan ini:

1. Gimana sih cara merancang dan membangun sistem optimasi rute pengiriman barang berbasis web yang bisa bikin total jarak atau waktu tempuh jadi seminimal mungkin? Ini khususnya buat skenario kirim barang dari satu gudang ke banyak pelanggan di dalam kota, sambil tetap mikirin batasan kayak muatan mobil (CVRP) dan jadwal waktu kirim (VRPTW).
2. Dari sekian banyak algoritma buat mecahin VRP (misalnya, yang simpel kayak *Clarke-Wright Savings* dan *Nearest Neighbor*, atau yang lebih canggih macam *Tabu Search*, *Simulated Annealing*, dan *Genetic Algorithm*), mana yang paling pas buat skala pengiriman lokal atau UMKM? Kita cari yang hasilnya bagus (rute paling hemat) tapi nggak butuh waktu lama buat ngitungnya.
3. Seperti apa arsitektur aplikasi web yang cocok buat sistem ini? Termasuk gimana cara nyambungin tampilan buat pengguna (*frontend*), otak di belakang layar (*backend*), tempat nyimpan data (basis data), jembatan komunikasi ke peta digital (API Peta, contohnya Google Maps API atau OpenStreetMap/OSRM), sama mesin optimasinya (misalnya, Google OR-Tools)?

### **C. Mari Batasi Dulu Biar Fokus (Batasan Masalah)**

Supaya proyek ini jelas arahnya dan bisa dikerjakan, kita batasi dulu ya ruang lingkupnya:

1. **Area Main:** Sistem ini fokusnya buat ngatur rute di dalam satu kota atau wilayah metropolitan aja.
2. **Model VRP:** Kita utamakan model *Capacitated Vehicle Routing Problem* (CVRP) 1 (yang mikirin kapasitas angkut) dan/atau *Vehicle Routing Problem with Time Windows* (VRPTW) 15 (yang mikirin jadwal waktu). Dua model ini relevan banget buat pengiriman lokal.
3. **Gudang Tunggal:** Kita anggap semua mobil berangkat dan pulang ke satu gudang pusat yang sama.
4. **Armada Kendaraan:** Anggap aja semua mobilnya sama (kapasitasnya seragam) atau paling banter ada beberapa jenis mobil dengan spek yang udah jelas.
5. **Data Tetap:** Kita anggap data pesanan, lokasi pelanggan, dan jadwal waktu itu udah pasti dan diketahui sebelum mulai ngitung rute. Jadi, sistem ini belum bisa nangani pesanan dadakan atau ubah rute di tengah jalan secara otomatis.
6. **Hitungan Waktu Tempuh:** Waktu antar lokasi dihitung berdasarkan perkiraan jarak dan kecepatan rata-rata, atau pakai data dari API peta (misalnya OSRM atau Google Maps API). Tapi, data macet *real-time* nggak otomatis masuk hitungan, kecuali API yang dipakai memang mendukung dan kita sambungkan secara khusus.
7. **Fitur Inti:** Aplikasi webnya fokus ke fungsi utama: masukin data (pesanan, pelanggan, mobil), mulai proses optimasi, dan nampilin hasil rutenya. Fitur canggih kayak lacak sopir *real-time*, atur stok barang, atau analisis mendalam, itu belum masuk di tahap awal ini.
8. **Algoritma & Alat Bantu:** Kita bakal pakai *library* yang udah ada kayak Google OR-Tools buat algoritma optimasinya, pakai cara pencarian bawaannya atau algoritma heuristik/metaheuristik yang didukung, atau mungkin bikin implementasi khusus algoritma kayak *Clarke-Wright Savings*.

### **D. Apa Sih Target Kita? (Tujuan Penelitian)**

Tujuan utama dari proyek ini adalah:

1. Menganalisis apa aja yang dibutuhin (fungsi dan non-fungsi) terus merancang arsitektur sistem optimasi rute berbasis web yang pas buat pengiriman barang skala kota.
2. Mempelajari dan bandingin berbagai algoritma heuristik dan metaheuristik buat nyelesaiin VRP (khususnya CVRP dan VRPTW), dilihat dari sisi efisiensi dan sebagus apa solusi rutenya.
3. Mengimplementasikan algoritma VRP yang terpilih, idealnya pakai *library* kayak Google OR-Tools, buat mecahin masalah CVRP dan/atau VRPTW.
4. Membangun prototipe aplikasi web yang bisa dipakai. Pengguna (misalnya manajer logistik) bisa masukin data pengiriman (lokasi, jumlah barang, jadwal, detail mobil) dan dapat hasil berupa rute yang udah dioptimalkan, lengkap dengan tampilan peta dan daftar urutan kunjungan.
5. Melakukan tes awal buat lihat seberapa bagus prototipe sistem ini, baik dari kualitas rutenya (misalnya total jarak/waktu) maupun seberapa cepat dia ngitung, pakai data simulasi atau contoh kasus kecil.

### **E. Apa Manfaatnya Nanti? (Manfaat Penelitian)**

Harapannya, sistem yang kita kembangkan ini bisa bawa manfaat:

1. **Buat Pengguna (UMKM/Distributor Lokal):** Ngasih alat bantu berbasis web yang gampang dipakai buat bikin rencana rute pengiriman jadi lebih optimal, nggak perlu lagi pusing ngatur manual.5
2. **Operasional Lebih Efisien:** Bantu tekan biaya logistik lewat penghematan bensin, potong waktu perjalanan, dan bikin penggunaan mobil jadi lebih maksimal.1
3. **Produktivitas Naik:** Bikin proses perencanaan dan penjadwalan distribusi barang jadi lebih cepat dan efisien.1
4. **Layanan Pelanggan Makin Oke:** Dengan mikirin jadwal waktu (VRPTW), sistem bisa bantu kirim barang lebih tepat waktu, bikin pelanggan makin senang dan setia.1
5. **Sumbangan Akademik:** Ngasih pemahaman praktis gimana konsep VRP, algoritma optimasi (heuristik & metaheuristik), plus aspek rancang bangun sistem informasi web diterapkan di dunia logistik perkotaan. Ini relevan banget, apalagi buat mata kuliah Analisis Perancangan Berbasis Objek.

## **II. Pahami Dulu Dasarnya: Vehicle Routing Problem (VRP)**

### **A. Apa Itu VRP dan Kenapa Penting?**

*Vehicle Routing Problem* (VRP) itu istilah umum buat sekelompok masalah optimasi yang tujuannya merancang rute paling bagus buat sekumpulan mobil (armada) biar bisa melayani banyak pelanggan atau titik lokasi yang tersebar.1 Anggap aja VRP ini versi lebih kompleks dari *Traveling Salesman Problem* (TSP). Kalau TSP cuma mikirin satu 'salesman' cari rute terpendek buat datengin semua 'kota' sekali jalan terus balik lagi, VRP ini melibatkan banyak mobil yang melayani banyak pelanggan dari satu atau beberapa gudang.7

Tujuan utama VRP biasanya sih biar biaya operasional seminimal mungkin. Ini sering diartikan sebagai meminimalkan total jarak tempuh, total waktu perjalanan, atau jumlah mobil yang dipakai, sambil tetap patuh sama aturan mainnya.1 Aturan main ini (kendala) mencerminkan batasan nyata di lapangan, kayak kapasitas muatan mobil, jadwal waktu layanan yang diminta pelanggan, jam kerja sopir, dan lain-lain.7

VRP ini penting banget di banyak industri dan jasa, contohnya 1:

* Distribusi barang (kirim paket online, makanan minuman, koran, bensin)
* Pengumpulan (angkut sampah industri, susu dari peternak, barang bekas)
* Layanan di lapangan (jadwal teknisi perbaikan, inspeksi kesehatan)
* Transportasi (rute bus sekolah, layanan antar-jemput panggilan)

Dengan rute yang optimal, perusahaan bisa hemat banyak (bensin, gaji, perawatan mobil), pakai aset (mobil, sopir) lebih maksimal, bikin pelanggan senang karena barang datang tepat waktu, bahkan bisa bantu kurangi polusi.1 Tapi, karena VRP ini masalah NP-hard 8, nyari solusi paling sempurna buat kasus skala besar dalam waktu singkat itu seringkali mustahil pakai cara eksak. Makanya, penelitian dan aplikasi VRP banyak fokus ke pengembangan algoritma heuristik dan metaheuristik yang efisien buat nemuin solusi yang 'cukup bagus' (mendekati optimal) dalam waktu yang wajar.10

### **B. Varian VRP yang Cocok Buat Pengiriman Lokal**

Nah, buat kasus pengiriman barang di dalam kota dari satu gudang/penjual, ada beberapa jenis VRP yang paling relevan:

1. **Capacitated Vehicle Routing Problem (CVRP):**
   * **Apa itu?** Ini VRP klasik di mana tiap mobil (anggap aja semua sama) punya batas muatan (berat atau volume), dan tiap pelanggan punya jumlah permintaan barang tertentu.1
   * **Tujuannya?** Cari rute-rute yang total biayanya (biasanya jarak) paling kecil. Syaratnya: tiap rute mulai dan selesai di gudang, tiap pelanggan didatangi sekali aja sama satu mobil, dan total barang di tiap rute nggak boleh lebih dari kapasitas mobil.1
   * **Aturan Mainnya:**
     + Tiap pelanggan didatangi tepat satu kali.8
     + Semua rute mulai dan selesai di gudang.8
     + Kapasitas mobil nggak boleh kelebihan muatan ( ∑i∈Routek​​qi​≤Q buat tiap rute k, di mana qi​ itu permintaan pelanggan i dan Q itu kapasitas mobil).1
   * **Kenapa Cocok?** Cocok banget, soalnya kebanyakan pengiriman barang (minuman, sembako, paket) pasti dibatasi sama ukuran atau daya angkut mobil/motornya.1
2. **Vehicle Routing Problem with Time Windows (VRPTW):**
   * **Apa itu?** Ini pengembangan dari VRP (seringnya CVRP juga) yang nambahin aturan jadwal waktu buat tiap pelanggan.12 Jadi, tiap pelanggan i punya rentang waktu [ei​,li​] (paling cepat ei​, paling lambat li​) kapan layanan harus dimulai.
   * **Tujuannya?** Biasanya, yang utama itu pakai mobil sesedikit mungkin, baru setelah itu minimalkan total jarak atau waktu tempuh. Tentu sambil tetap patuh sama aturan kapasitas (kalau ada) dan jadwal waktu.12
   * **Aturan Main Tambahan (Selain aturan VRP/CVRP biasa):**
     + Layanan di pelanggan i harus mulai antara jam ei​ sampai li​.15
     + Kalau mobil datang sebelum ei​, harus nunggu (biasanya waktu nunggu nggak dihitung biaya).15
     + Mobil nggak boleh datang setelah li​ (ini kalau aturannya ketat, *hard time windows*).15 (Ada juga versi *soft time windows* yang ngebolehin telat sedikit tapi kena denda biaya).16
     + Waktu perjalanan antar lokasi (tij​) dan waktu layanan di tiap lokasi (si​) harus ikut dihitung pas bikin jadwal.15
   * **Kenapa Cocok?** Penting banget buat bisnis yang pelanggannya punya jadwal spesifik (misalnya, kirim makanan ke restoran pas jam sibuk, layanan ke kantor lain yang jam bukanya terbatas, atau pengiriman terjadwal ke rumah).2
3. **Capacitated Vehicle Routing Problem with Time Windows (CVRPTW):**
   * **Apa itu?** Ini gabungan CVRP dan VRPTW. Jadi, aturan kapasitas mobil dan jadwal waktu pelanggan dipakai barengan.20 Ini salah satu varian VRP yang paling sering dipelajari dan paling mirip sama banyak situasi logistik di dunia nyata.
   * **Tujuan dan Aturan Main:** Gabungin tujuan dan aturan dari CVRP dan VRPTW. Intinya, cari rute paling hemat yang bisa memenuhi permintaan pelanggan sesuai jadwalnya tanpa bikin mobil kelebihan muatan.
   * **Kenapa Cocok?** Ini model paling lengkap buat banyak kasus pengiriman UMKM atau distributor lokal yang harus mikirin kapasitas truk/motor sekaligus permintaan waktu kirim dari pelanggan.

Penting banget buat paham varian VRP mana yang paling pas sama masalah kita. Apakah kapasitas mobil yang jadi batasan utama? Atau justru jadwal pengiriman yang lebih kritis? Menjawab ini adalah langkah awal buat bikin sistem optimasi yang efektif. Buat pengiriman UMKM di dalam kota, CVRPTW seringkali jadi model yang paling mewakili kondisi sebenarnya.

## **III. Jurus Jitu Optimasi Rute (Metode Pemecahan Masalah)**

Menyelesaikan VRP, apalagi yang ada embel-embel CVRP atau VRPTW, itu sama aja kayak nyari jalan terbaik dari sekian banyak kemungkinan rute. Karena rumitnya minta ampun (ingat, NP-hard!), muncullah berbagai jurus algoritma. Ada yang eksak (pasti nemu solusi terbaik, tapi cuma kuat buat masalah kecil), ada juga yang heuristik dan metaheuristik (tujuannya nemu solusi bagus dalam waktu cepat).8 Buat aplikasi praktis skala kota, biasanya kita pakai jurus heuristik dan metaheuristik.12

### **A. Algoritma Heuristik (Pendekatan Konstruktif: Bangun dari Awal)**

Algoritma heuristik ini kayak membangun rumah bata demi bata, pakai aturan tertentu, dan biasanya 'serakah' (ambil pilihan terbaik saat itu juga). Cepat sih, tapi nggak jamin hasilnya paling bagus.9 Dua contoh klasik yang relevan:

1. **Algoritma Clarke-Wright Savings (CWS): Hemat Pangkal Optimal?**
   * **Gimana Caranya?** Awalnya, anggap tiap pelanggan dilayani mobil sendiri-sendiri (gudang -> pelanggan -> gudang). Terus, pelan-pelan rute-rute ini digabung berdasarkan seberapa besar 'penghematan' (*savings*) jarak yang didapat kalau digabung.9
   * **Langkah-langkahnya** 9**:**
     1. **Hitung Penghematan:** Buat tiap pasang pelanggan (i, j), hitung hematnya (Sij​) kalau mereka digabung satu rute dibanding dilayani pisah: Sij​=ci0​+c0j​−cij​. Di sini, cxy​ itu biaya/jarak dari x ke y, dan 0 itu gudang.9
     2. **Urutkan:** Bikin daftar semua pasangan (i, j) yang Sij​-nya positif, urut dari yang paling hemat ke yang paling sedikit hematnya.9
     3. **Rute Awal:** Mulai dengan n rute pisah: (0, i, 0) buat tiap pelanggan i.
     4. **Mulai Gabung:** Cek daftar penghematan dari atas ke bawah. Buat tiap pasangan (i, j) yang paling hemat:
        + Lihat dulu, bisa digabung nggak? Syaratnya:
          - Pelanggan i itu titik terakhir sebelum balik ke gudang di rutenya, DAN pelanggan j itu titik pertama setelah dari gudang di rutenya.
          - Kalau rute i dan j digabung, total muatannya nggak boleh lebih dari kapasitas mobil (Q).9
          - (Kalau ada aturan lain kayak jadwal waktu, cek juga).
        + Kalau bisa digabung, ya gabungkan aja rutenya dengan nyambungin i ke j (misal, rute (..., k, i, 0) dan (0, j, l,...) jadi (..., k, i, j, l,...)).
     5. **Ulangi:** Terus gabung-gabungin sampai daftar penghematan habis atau nggak ada lagi yang bisa digabung.
   * **Plus Minus:** CWS ini lumayan gampang dibuat dan cepat.33 Hasilnya seringkali cukup oke buat permulaan, tapi jarang banget jadi yang paling optimal.32 Ada versi paralel (lihat semua kemungkinan gabung tiap langkah) dan sekuensial (selesaikan satu rute dulu baru mulai lagi).33 Hasilnya bisa beda tergantung parameter tambahan.35 Enaknya, algoritma ini otomatis mikirin kapasitas pas proses penggabungan.9
2. **Algoritma Nearest Neighbor (NN): Siapa Tetangga Terdekat?**
   * **Gimana Caranya?** Algoritma ini super 'serakah'. Dia selalu pilih pelanggan terdekat yang belum didatangi sebagai tujuan berikutnya.31
   * **Langkah-langkahnya (Versi Sekuensial buat CVRP)** 38**:**
     1. **Mulai:** Dari gudang (titik 0). Ambil mobil pertama.
     2. **Cari Tetangga:** Dari lokasi sekarang, cari pelanggan terdekat yang belum didatangi DAN barangnya masih muat di sisa kapasitas mobil.
     3. **Datangi & Update:** Pergi ke pelanggan itu, layani permintaannya, kurangi sisa kapasitas mobil, tandai pelanggan sudah (sebagian) didatangi.
     4. **Ulangi:** Ulangi langkah 2 dan 3 dari lokasi pelanggan baru. Terus tambah pelanggan terdekat yang muat ke rute sampai:
        + Nggak ada lagi pelanggan yang belum didatangi yang bisa ditambah ke rute ini tanpa bikin mobil kelebihan muatan.
     5. **Pulang:** Balik ke gudang dari pelanggan terakhir di rute ini.
     6. **Rute Berikutnya:** Kalau masih ada pelanggan yang belum dilayani, ambil mobil berikutnya, ulangi dari langkah 2 (mulai dari gudang lagi, layani sisa permintaan).
     7. **Selesai:** Berhenti kalau semua permintaan pelanggan sudah terpenuhi.
   * **Versi Lain:** NN bisa juga bikin satu rute raksasa dulu (kayak TSP) baru dipecah-pecah jadi rute kecil sesuai kapasitas.38 Ada juga versi paralel yang bikin beberapa rute barengan.41
   * **Plus Minus:** NN ini simpel banget dan super cepat (kompleksitas O(n²)) 41, tapi hasilnya sering jauh dari bagus karena terlalu 'serakah' dan tergantung banget sama titik mulainya.37 Performanya jelek kalau banyak aturan main.41 Tapi karena simpel, gampang dimodifikasi.36

### **B. Algoritma Metaheuristik (Pendekatan Perbaikan: Dari yang Ada Jadi Lebih Baik)**

Metaheuristik ini kayak pelatih pintar yang ngasih arahan ke heuristik biar bisa jelajahi kemungkinan solusi lebih luas dan nggak gampang puas sama solusi bagus pertama yang ditemui (optima lokal). Biasanya mulai dari satu atau beberapa solusi awal (seringnya hasil dari heuristik konstruktif tadi) terus coba diperbaiki terus-menerus.15

1. **Tabu Search (TS): Jangan Kembali ke Jalan yang Sama!**
   * **Gimana Caranya?** TS ini metode pencarian lokal yang lebih pintar. Dia lihat solusi-solusi 'tetangga' (mirip tapi beda dikit) dari solusi sekarang, terus pindah ke tetangga terbaik, *walaupun* tetangga itu lebih jelek dari solusi sekarang (biar bisa keluar dari jebakan optima lokal). Biar nggak bolak-balik ke solusi yang sama terus, TS pakai 'daftar tabu' (*tabu list*). Daftar ini nyatet gerakan atau solusi yang baru aja dipakai dan melarang gerakan itu dipakai lagi untuk sementara waktu.7
   * **Cara Kerja Umum:**
     1. **Mulai:** Ambil solusi awal (misal dari CWS atau NN).
     2. **Lihat Tetangga:** Bikin beberapa solusi tetangga dari solusi sekarang pakai 'operator gerakan' (*move operators*). Contoh operator buat VRP:
        + *Tukar (Swap/Exchange):* Tukar posisi dua pelanggan di satu rute atau antar rute.28
        + *Pindah (Insertion/Relocation):* Pindahin satu pelanggan dari satu tempat ke tempat lain di rute yang sama atau beda.26
        + *2-opt/k-opt:* Hapus beberapa sambungan rute terus sambungin lagi dengan cara beda biar lebih pendek.32
        + *CROSS Exchange:* Operator canggih yang nukar potongan rute antar dua mobil.28
     3. **Evaluasi & Pilih:** Hitung biaya (misal, total jarak + denda kalau langgar aturan) dari semua solusi tetangga. Pilih gerakan yang hasilnya paling murah.
     4. **Cek Daftar Tabu:** Gerakan yang dipilih tadi ada di daftar tabu nggak?
     5. **Kriteria Aspirasi:** Kalau ternyata tabu, cek lagi, ada 'kriteria aspirasi' nggak? Kriteria ini ngebolehin gerakan tabu kalau, misalnya, hasilnya lebih bagus dari solusi terbaik yang pernah ditemuin (*best-so-far*).42
     6. **Lakukan Gerakan:** Kalau gerakannya nggak tabu ATAU memenuhi kriteria aspirasi, lakukan gerakan itu. Jadilah solusi sekarang yang baru.
     7. **Update Daftar Tabu:** Masukin info gerakan tadi ke daftar tabu (dan hapus yang paling lama kalau daftarnya penuh). Info ini bisa berupa pelanggan yang dipindah, pasangan yang ditukar, dll. 'Masa tabu' (*tabu tenure*) nentuin berapa lama gerakan itu dilarang.42
     8. **Update Solusi Terbaik:** Kalau solusi sekarang lebih bagus dari *best-so-far*, update *best-so-far*.
     9. **Ulangi:** Balik ke langkah 2 sampai berhenti (misal, udah sekian kali iterasi, waktu habis, atau nggak nemu yang lebih bagus lagi).
   * **Tambahan:** TS sering ditambah strategi *intensification* (fokus cari di area solusi yang menjanjikan) dan *diversification* (dorong pencarian ke area baru biar nggak mentok).42 Memori adaptif (nyimpen rute bagus dari solusi lama) juga bisa dipakai buat bikin titik awal baru.28
   * **Plus Minus:** TS ini jago banget buat VRP dan variasinya.25 Kinerjanya tergantung gimana kita desain tetangganya, operator gerakannya, cara ngatur daftar tabu, dan strategi intensifikasi/diversifikasinya. Ngatur batasan (kapasitas, jadwal waktu) biasanya dengan cek apakah solusi tetangga masih valid atau pakai sistem denda di perhitungan biaya.22
2. **Simulated Annealing (SA): Meniru Proses Pendinginan Logam**
   * **Gimana Caranya?** SA ini metaheuristik yang idenya dari proses pendinginan logam (*annealing*) biar dapet struktur paling stabil (energi minimum).23 Algoritma ini mulai nyari solusi pas 'suhu' lagi tinggi, jadi dia berani pindah ke solusi yang lebih jelek dengan kemungkinan besar. Terus, suhunya pelan-pelan diturunin, kemungkinan terima solusi jelek makin kecil, sampai akhirnya nemu solusi bagus.46
   * **Cara Kerja Umum** 23**:**
     1. **Mulai:** Ambil solusi awal S. Setel suhu awal tinggi T0​, suhu akhir Tf​, dan jadwal pendinginan (misal, faktor pendinginan α). Catat solusi terbaik Sbest​=S.
     2. **Loop Suhu:** Selama suhu T masih di atas Tf​:
        + **Loop Iterasi:** Ulangi beberapa kali (misal, Niter​) di suhu T sekarang:
          1. **Bikin Tetangga:** Bikin solusi tetangga S′ dari solusi S pakai operator gerakan acak (misal, tukar, pindah, balik urutan).23
          2. **Hitung Perubahan:** Hitung beda biayanya ΔE=cost(S′)−cost(S).
          3. **Terima atau Tolak?:**

Kalau ΔE<0 (S′ lebih bagus), langsung terima S′ jadi solusi sekarang: S=S′.

Kalau ΔE≥0 (S′ sama atau lebih jelek), terima S′ dengan kemungkinan P=e−ΔE/T.47 Caranya, lempar 'dadu' acak r (angka 0-1). Kalau r<P, terima S=S′. Kalau nggak, tetap pakai S.

* + - * 1. **Update Solusi Terbaik:** Kalau biaya S lebih kecil dari biaya Sbest​ DAN S itu valid (menuhi semua aturan), update Sbest​=S.
      * **Turunkan Suhu:** Kurangi suhu sesuai jadwal: T=T×α (geometris) atau cara lain.47
    1. **Hasil Akhir:** Kalau loop suhu selesai, kasih solusi terbaik Sbest​ yang ketemu.
  + **Plus Minus:** SA bisa lolos dari jebakan optima lokal karena berani terima solusi jelek, apalagi pas suhu tinggi.44 Kinerjanya tergantung banget sama settingan: suhu awal, jadwal pendinginan, berapa kali iterasi per suhu, dan operator gerakannya.45 Kalau dinginnya kecepetan, bisa dapet solusi kurang bagus. Kalau kelamaan, boros waktu.46 Ngatur batasan kayak kapasitas biasanya dengan cek dulu S′ valid atau nggak, atau masukin denda ke fungsi biayanya.23 SA udah sukses dipakai buat macam-macam VRP.23

1. **Genetic Algorithms (GA): Evolusi Mencari Solusi Terbaik**
   * **Gimana Caranya?** GA ini terinspirasi dari evolusi biologi dan seleksi alam.25 GA bekerja pakai sekumpulan solusi (disebut populasi kromosom) terus menerapkan 'operator genetik' kayak seleksi, kawin silang (*crossover*), dan mutasi buat ngasilin generasi solusi baru yang diharapkan makin lama makin bagus.25
   * **Cara Kerja Umum** 51**:**
     1. **Bikin Populasi Awal:** Bikin sekumpulan N 'individu' (kromosom). Tiap kromosom itu representasi satu solusi VRP (misalnya, urutan semua pelanggan yang nanti dibagi jadi rute). Biasanya pakai urutan angka pelanggan.51 Bisa dibikin acak atau pakai heuristik biar dapet solusi awal yang lumayan.49
     2. **Ukur Kebugaran (Fitness):** Hitung nilai *fitness* buat tiap individu. Buat VRP, *fitness* biasanya kebalikan dari total biaya/jarak rute yang dihasilkan kromosom itu (setelah diubah jadi rute beneran).50 Bisa ditambah denda kalau melanggar aturan.
     3. **Loop Generasi:** Ulangi proses ini beberapa generasi sampai berhenti:
        + **Seleksi:** Pilih individu (orang tua) buat 'berkembang biak'. Cara pilihnya macem-macem, misalnya *roulette wheel* (makin bagus *fitness*-nya, makin besar peluang terpilih) atau *tournament*.25 Yang *fitness*-nya tinggi lebih mungkin kepilih. Sering juga pakai *elitism* (individu terbaik langsung dicopy ke generasi berikutnya).25
        + **Crossover (Kawin Silang):** Pasangkan orang tua terpilih, terus pakai operator *crossover* dengan kemungkinan tertentu (pc​) buat bikin 'anak' (keturunan). Operator *crossover* buat VRP harus hati-hati biar hasilnya tetap urutan pelanggan yang valid dan idealnya anaknya juga valid. Contoh:
          - *Partially Mapped Crossover (PMX)* 52
          - *Order Crossover (OX)* 52
          - *Edge Crossover (EC)* 52
          - *Improved Route Crossover* (kayak di 51): coba pertahanin potongan rute bagus dari orang tua sambil pastiin semua pelanggan tetap dilayani dan aturan (misal, kapasitas) terpenuhi di anak.
        + **Mutasi:** Terapkan operator mutasi ke anak (kadang ke individu asli juga) dengan kemungkinan kecil (pm​) buat nambah variasi baru dan cegah mentok terlalu cepat. Contoh operator mutasi buat urutan:
          - *Swap Mutation:* Tukar posisi dua gen (pelanggan).50
          - *Insert Mutation:* Pindahin satu gen ke posisi lain.52
          - *Inversion Mutation:* Balik urutan sebagian gen.52
          - *2-Exchange Mutation* (kayak di 51): tukar dua gen.
        + **Bikin Populasi Baru:** Bentuk populasi baru dari anak-anak yang dihasilkan dan mungkin beberapa individu dari populasi lama (tergantung strategi seleksi dan *elitism*).
     4. **Hasil Akhir:** Kalau udah berhenti, kasih individu terbaik (yang *fitness*-nya paling tinggi) yang ketemu selama evolusi.
   * **Plus Minus:** GA jago menjelajah banyak kemungkinan solusi karena sifat populasinya.25 Tapi kinerjanya tergantung banget sama cara representasi kromosom, operator genetiknya, ukuran populasi, kemungkinan *crossover* & mutasi, dan cara nangani aturan.49 Nangani aturan (kapasitas, jadwal waktu) bisa dengan:
     + Representasi yang selalu valid.
     + Fungsi denda di *fitness*.50
     + Operator perbaikan buat benerin solusi nggak valid.49
     + Operator *crossover* & mutasi khusus yang jaga validitas.51 GA sering digabung sama pencarian lokal (jadi *Memetic Algorithm*) biar makin jago nyari solusi bagus di sekitar area tertentu.44 GA udah banyak dipakai buat VRP.12

**Mana yang Dipilih?**

Nggak ada satu algoritma 'juara' buat semua VRP. Pilihannya tergantung:

* **Ukuran & Kerumitan Masalah:** Buat masalah kecil banget atau buat bikin solusi awal, heuristik simpel (CWS, NN) mungkin cukup. Buat yang lebih besar dan rumit (misal, >50 pelanggan, banyak aturan), metaheuristik (TS, SA, GA) biasanya hasilnya jauh lebih bagus.8
* **Kualitas Solusi vs Waktu:** Metaheuristik butuh waktu ngitung lebih lama, tapi hasilnya lebih mendekati optimal.15 Ada tawar-menawar antara waktu nyari sama kualitas hasil.15 TS dan GA sering dilaporkan ngasih hasil bagus buat VRP standar.25
* **Jenis Aturan:** Beberapa algoritma lebih gampang diubah buat nangani aturan spesifik (misal, TS dan SA biasanya fleksibel pakai denda atau cek validitas tetangga).
* **Gampang Dibuat:** Heuristik konstruktif lebih gampang. Metaheuristik lebih repot dan perlu setting parameter (*tuning*). Pakai *library* kayak Google OR-Tools bisa bikin implementasi metaheuristik jadi lebih simpel.10

Buat proyek sistem optimasi rute UMKM skala kota ini, saran pendekatannya:

1. Pakai heuristik konstruktif (kayak CWS atau strategi *savings* di OR-Tools) buat bikin solusi awal yang cepat dan valid.51
2. Pakai metaheuristik (kayak *Guided Local Search* atau *Tabu Search* yang ada di OR-Tools, atau bikin GA/SA/TS sendiri kalau perlu) buat perbaiki solusi awal tadi dan nemuin solusi berkualitas tinggi dalam waktu yang masuk akal.10

## **IV. Apa Saja yang Perlu Diperhatikan? Variabel Kunci dan Kendala Sistem Optimasi Rute Lokal**

Merancang model VRP yang pas buat sistem optimasi rute pengiriman lokal itu butuh ketelitian dalam mengenali dan memodelkan variabel serta kendala yang relevan. Variabel ini mewakili apa saja yang ada dan keputusan apa yang diambil, sementara kendala itu aturan mainnya.

### **A. Keputusan Utama yang Diambil (Variabel Keputusan)**

Inti keputusan dalam VRP biasanya soal nentuin rute dan urutan kunjungan:

1. **Variabel Aliran/Penugasan (misalnya, xijk​ atau xij​):**
   * Seringnya berupa angka biner (0 atau 1). xijk​=1 kalau mobil k jalan langsung dari lokasi i ke j, dan 0 kalau nggak.16
   * Kalau pakai dua indeks (misal mobilnya sama semua), xij​=1 kalau ada mobil jalan dari i ke j, dan 0 kalau nggak.4
   * Variabel ini secara nggak langsung nentuin rute dan urutan kunjungan.
2. **Variabel Waktu Kedatangan/Mulai Layanan (misalnya, tik​ atau ui​):**
   * Variabel angka (kontinu) yang nunjukin jam berapa mobil k sampai atau mulai melayani di lokasi i.16
   * Penting buat ngatur kendala jadwal waktu (VRPTW).15
3. **Variabel Muatan Kendaraan (misalnya, Lik​):**
   * Variabel angka (kontinu) yang nunjukin sisa muatan atau total muatan mobil k setelah dari lokasi i.30
   * Penting buat ngatur kendala kapasitas (CVRP).
4. **Variabel Kunjungan/Penugasan Pelanggan (misalnya, yik​):**
   * Variabel biner, yik​=1 kalau pelanggan i dilayani mobil k, dan 0 kalau nggak. Kadang dipakai di model tertentu, tapi seringnya udah tercakup di variabel aliran.

### **B. Data yang Dibutuhkan (Parameter Sistem)**

Ini data-data yang perlu dimasukkan ke sistem optimasi buat mendefinisikan masalahnya:

1. **Lokasi:**
   * **Gudang (Depot):** Titik pusat (cuma satu di kasus ini) tempat semua mobil mulai dan selesai jalan.4 Perlu koordinat (lintang, bujur) atau alamat yang bisa dicari di peta.
   * **Pelanggan/Tujuan:** Daftar semua lokasi yang harus didatangi.11 Masing-masing perlu koordinat atau alamat. Jumlah pelanggan (n) ini parameter penting buat nentuin seberapa besar masalahnya.10
2. **Permintaan Pelanggan (qi​):**
   * Jumlah barang (misal, unit, kg, volume) yang diminta tiap pelanggan i.1 Permintaan gudang (q0​) biasanya 0.11
3. **Armada Kendaraan:**
   * **Jumlah Mobil (m atau K):** Total mobil yang siap jalan.15
   * **Kapasitas Mobil (Q):** Muatan maksimal tiap mobil (anggap sama buat CVRP homogen).1
   * **Lokasi Awal/Akhir:** Biasanya sama kayak lokasi gudang.11
4. **Matriks Biaya/Jarak/Waktu Tempuh (cij​ atau dij​ atau tij​):**
   * Tabel yang isinya biaya, jarak, atau waktu tempuh antar tiap pasang lokasi (gudang dan pelanggan).4
   * Ini data krusial buat tujuan utama (minimalkan total jarak/waktu).
   * Data ini bisa didapat dari:
     + Hitungan jarak lurus (Euclidean) kalau cuma punya koordinat (kurang akurat buat jalanan asli).33
     + API Peta (Google Maps, OSRM, dll.) yang ngasih jarak/waktu tempuh berdasarkan jalanan sebenarnya.7 API ini mungkin bisa ngitung faktor macet juga (historis atau *real-time*).56
5. **Jadwal Waktu Pelanggan ([ei​,li​]):**
   * Buat VRPTW, tiap pelanggan i punya waktu paling awal (ei​) dan paling lambat (li​) buat mulai dilayani.15 Gudang juga bisa punya jam operasional [e0​,l0​].28
6. **Waktu Layanan (si​):**
   * Waktu yang dibutuhin buat ngelayanin di lokasi pelanggan i (misalnya, bongkar muat barang).15 Waktu layanan di gudang (s0​) biasanya 0.

### **C. Aturan Main yang Harus Dipatuhi (Kendala/Constraints)**

Kendala ini adalah aturan-aturan yang harus dipenuhi biar solusi rutenya dianggap sah dan bisa dijalankan:

1. **Kendala Kunjungan Pelanggan:**
   * Tiap pelanggan harus didatangi tepat satu kali oleh tepat satu mobil.4 (Rumusnya: ∑k∈K​∑j∈Δ+(i)​xijk​=1,∀i∈Pelanggan 16).
2. **Kendala Aliran Kendaraan:**
   * Tiap mobil yang masuk ke satu lokasi pelanggan harus keluar lagi dari situ (kecuali gudang).26 (Rumusnya: ∑i∈Δ−(j)​xijk​−∑i∈Δ+(j)​xjik​=0,∀k∈K,∀j∈Pelanggan 16).
   * Tiap rute mobil harus mulai dari gudang dan berakhir di gudang.8 (Rumusnya: ∑j∈Δ+(0)​x0jk​=1,∀k∈K dan ∑i∈Δ−(0)​xi0k​=1,∀k∈K buat rute tertutup).
3. **Kendala Kapasitas Kendaraan (CVRP):**
   * Total permintaan dari semua pelanggan dalam satu rute nggak boleh lebih dari kapasitas mobilnya.1 Ini dimodelkan pakai variabel muatan kumulatif atau batasan lain.
4. **Kendala Jendela Waktu (VRPTW):**
   * Waktu mulai layanan di tiap pelanggan i, ui​, harus ada di antara ei​ dan li​.15 (Rumusnya: ei​≤ui​≤li​,∀i∈Pelanggan).
   * Hubungan waktu antar kunjungan: Waktu mulai layanan di pelanggan j (uj​) harus minimal sama dengan waktu mulai layanan di pelanggan sebelumnya i (ui​), ditambah waktu layanan di i (si​), ditambah waktu tempuh dari i ke j (tij​). Kalau mobil datang kepagian sebelum ej​, harus nunggu. (Rumusnya: ui​+si​+tij​≤uj​ kalau xij​=1. Buat ngatur waktu tunggu: uj​≥max(ej​,ui​+si​+tij​)).16
5. **Kendala Durasi Rute Maksimum (Opsional):**
   * Total waktu tempuh atau jam kerja buat tiap rute mobil mungkin dibatasi.15
6. **Kendala Eliminasi Sub-rute:**
   * Memastikan solusinya nggak ada rute-rute kecil yang nggak nyambung ke gudang.26 Ini sering otomatis tertangani di model aliran atau ditambah aturan khusus.
7. **Kendala Urutan (untuk VRPPD):**
   * Kalau ada kasus Ambil & Antar (*Pickup and Delivery*), lokasi ambil barang harus didatangi sebelum lokasi antar barangnya di rute yang sama.55
8. **Kendala Lainnya (Tergantung Kebutuhan):**
   * Bisa jadi ada prioritas pelanggan 2, aturan mobil mana yang boleh ke pelanggan mana, waktu istirahat sopir 7, jalan yang nggak boleh dilewati, dll.

Memodelkan variabel dan kendala ini dengan benar itu penting banget. Kalau salah atau ada yang kelewat pas masukin data atau nentuin aturan, hasil optimasinya bisa jadi nggak realistis atau nggak bisa dipakai di lapangan. Untungnya, pakai *library* kayak Google OR-Tools sering bikin proses ini lebih gampang karena udah ada cara terstruktur buat nentuin dimensi (kayak waktu dan kapasitas) dan nambahin kendala ke variabel-variabelnya.20

## **V. Rancangan Sistem dan Teknologi Pendukungnya**

Membangun sistem optimasi rute berbasis web butuh rancangan arsitektur yang kokoh biar semua komponen teknologi bisa nyambung dan sistemnya jalan lancar, bisa dikembangkan, dan gampang dirawat. Arsitektur ini harus bisa nangani input dari pengguna, ngolah data, jalanin algoritma optimasi yang rumit, dan nampilin hasilnya dengan jelas.

### **A. Model Arsitektur Umum: Tiga Lapis (3-Tier)**

Arsitektur tiga lapis ini udah jadi standar dan terbukti ampuh buat aplikasi web modern, termasuk sistem optimasi rute. Intinya, aplikasi dibagi jadi tiga lapisan utama, masing-masing punya tugasnya sendiri 13:

1. **Lapis Tampilan (Presentation Tier / Frontend):**
   * **Tugasnya:** Ini adalah 'wajah' aplikasi yang dilihat dan dipakai langsung sama pengguna (misalnya, manajer logistik). Tugasnya nampilin data, nerima input, dan nunjukin hasil (kayak peta rute).
   * **Letaknya:** Jalan di browser pengguna (*client-side*).
   * **Teknologinya:** HTML buat struktur, CSS buat tampilan cantik, dan JavaScript buat bikin interaktif.67 Biasanya pakai *framework* JavaScript modern kayak React, Angular, atau Vue.js biar tampilannya canggih, dinamis, dan enak dilihat di berbagai ukuran layar.65 Perlu juga *library* peta *frontend* (Leaflet, Mapbox GL JS, Google Maps JavaScript API) buat nampilin peta dan rutenya.71
2. **Lapis Aplikasi (Application Tier / Backend):**
   * **Tugasnya:** Ini 'otak' aplikasi yang jalan di server (*server-side*). Tugasnya ngolah permintaan dari *frontend*, ngatur data pengguna dan pesanan, ngobrol sama basis data, komunikasi sama layanan luar (kayak API peta), manggil mesin optimasi, dan kirim hasil balik ke *frontend*.13
   * **Komponennya:** Satu atau lebih server web dan server aplikasi.
   * **Teknologinya:** Bahasa pemrograman *backend* (Python, Java, Node.js, C#, PHP, Ruby) dan *framework* web pasangannya (Django/Flask buat Python, Spring buat Java, Express buat Node.js, ASP.NET buat C#, Laravel buat PHP, Rails buat Ruby).65 Lapisan ini juga ngatur API (*Application Programming Interfaces*) buat komunikasi sama *frontend* dan layanan lain.13
3. **Lapis Data (Data Tier / Persistence):**
   * **Tugasnya:** Tempat nyimpen data aplikasi secara permanen dan ngambil data kalau dibutuhin.13
   * **Komponennya:** Sistem manajemen basis data (DBMS).
   * **Data yang Disimpan:** Info pengguna (admin, manajer), data pelanggan (alamat, koordinat, permintaan, jadwal), detail mobil (kapasitas, gudang), detail pesanan, settingan sistem, dan hasil rute optimasi.
   * **Teknologinya:** Bisa pilih basis data SQL (relasional) kayak PostgreSQL atau MySQL (bagus buat data terstruktur kayak pesanan dan pelanggan) 13 atau NoSQL kayak MongoDB (bagus buat data yang strukturnya fleksibel atau data lokasi yang banyak banget).13 Bisa juga pakai *cache* kayak Redis buat percepat akses data yang sering dipakai.70 PostgreSQL plus ekstensi PostGIS sering jadi rekomendasi buat aplikasi lokasi karena jago banget ngolah data spasial.72

### **B. Komponen Kunci Sistem Optimasi Rute**

Selain tiga lapis tadi, sistem optimasi rute ini punya beberapa bagian penting:

1. **Antarmuka Pengguna (Frontend):**
   * Tempat pengguna (terutama Manajer Logistik) buat:
     + Ngatur data utama (pelanggan, lokasi, mobil).
     + Masukin atau impor data pesanan harian (tujuan, jumlah barang, jadwal).
     + Milih parameter optimasi (mobil mana yang dipakai, tanggal berapa).
     + Mulai proses optimasi.
     + Lihat hasil rute di peta interaktif dan daftar urutan kunjungan.
     + (Kalau perlu) Bikin penyesuaian kecil manual di rute.
2. **Server Aplikasi (Backend):**
   * Ngurus login dan hak akses pengguna.
   * Nyediain API buat *frontend*.
   * Validasi dan proses data dari pengguna.
   * Ngatur interaksi sama basis data (Tambah, Baca, Ubah, Hapus - CRUD).
   * Nyiapin data buat mesin optimasi.
   * Komunikasi sama API Peta buat dapetin matriks jarak/waktu.
   * Manggil mesin optimasi.
   * Terima dan olah hasil optimasi (rute).
   * Simpan hasil rute ke basis data.
   * Kirim data rute yang udah rapi ke *frontend* buat ditampilin.
3. **Basis Data (Persistence):**
   * Nyimpen semua data penting kayak yang dijelasin di Lapis Data. Strukturnya harus dirancang biar gampang cari data pelanggan, pesanan, dan rute.
4. **Integrasi Layanan Peta (Mapping Service Integration):**
   * **Tugasnya:** Nyediain data geografis penting: peta dasar buat tampilan, layanan geocoding (ubah alamat jadi koordinat), *reverse geocoding*, dan yang paling vital, ngitung matriks jarak dan/atau waktu tempuh antar semua lokasi (gudang dan pelanggan).56
   * **Interaksinya:** *Backend* ngobrol sama API layanan peta. *Frontend* juga bisa ngobrol langsung (misal pakai JavaScript API) buat nampilin peta dan rute.
   * **Pilihannya:** Google Maps Platform APIs, OpenStreetMap (OSM) lewat mesin kayak OSRM (*Open Source Routing Machine*), Mapbox, ArcGIS Platform, Bing Maps API.7 Nanti kita bahas lebih lanjut pilihannya.
5. **Mesin Optimasi (Optimization Engine):**
   * **Tugasnya:** Ini komponen inti yang jalanin algoritma VRP (heuristik/metaheuristik) buat nemuin solusi rute terbaik atau mendekati terbaik berdasarkan data yang dikasih (lokasi, permintaan, mobil, aturan).7
   * **Implementasinya:** Bisa berupa:
     + *Library* optimasi yang langsung dipasang di *backend* (misalnya, Google OR-Tools buat Python, Java, C++, C# 6).
     + Algoritma bikinan sendiri.
     + Layanan optimasi di cloud yang dipanggil lewat API.
   * **Interaksinya:** *Backend* nyiapin data sesuai format mesin, kirim permintaan optimasi, terus terima hasilnya (daftar rute).

Penting banget semua komponen ini bisa kerja bareng dengan mulus. Arsitekturnya harus bikin data bisa ngalir lancar antar lapisan dan komponen (misal, input pengguna di *frontend* -> diolah *backend* -> data dikirim ke mesin optimasi -> hasil diterima *backend* -> simpan di database -> tampil di *frontend*).13 Pilihan teknologi di satu lapis bisa ngaruh ke kemudahan nyambung ke lapis lain; contohnya, pakai Python di *backend* bikin lebih gampang pakai *library* OR-Tools Python.68

Satu lagi, mesin optimasi ini 'otak'-nya, tapi bisa jadi biang kerok kalau lemot. Nyelesaiin VRP itu bisa makan waktu lama, apalagi buat masalah besar atau pakai metaheuristik rumit.8 Permintaan web biasanya ada batas waktunya.7 Kalau optimasi rumit dijalanin langsung pas ada permintaan web, bisa *timeout* atau bikin pengguna nunggu kelamaan. Jadi, arsitekturnya perlu mikirin gimana proses optimasi ini dipicu. Buat optimasi yang mungkin lama, pendekatan asinkron (misalnya, pakai antrian tugas kayak Celery di Python/Django) sering lebih bagus. *Backend* bisa taruh tugas optimasi di antrian, langsung kasih respons ke pengguna, terus kasih notif kalau hasilnya udah siap. Pola interaksi *backend* dan mesin optimasi ini perlu dirancang hati-hati.

### **C. Bandingkan Layanan Peta: Google Maps vs OSRM**

Hitungan jarak dan waktu tempuh yang akurat itu dasar banget buat VRP, karena ini jadi patokan biaya yang mau diminimalkan.4 API Peta nyediain data ini.

| **Fitur Kunci** | **Google Maps Platform API** | **OSRM (OpenStreetMap)** |
| --- | --- | --- |
| Sumber Data | Milik Google (Proprietary) | OpenStreetMap (Data terbuka, kualitas beda-beda) |
| Model Biaya | Bayar per pemakaian (tergantung panggilan/elemen) 56 | Gratis (kalau host sendiri), Biaya hosting/pihak ketiga |
| Data Lalu Lintas | Ada *Real-time* & Historis 57 | Standarnya nggak ada (perlu data luar) 57 |
| Kualitas Geocoding | Umumnya sangat bagus | Bervariasi, tergantung data OSM lokal |
| API Matriks Jarak | Ada (Distance Matrix API) 57 | Ada (Table Service) 72 |
| Kemudahan Pakai | Tinggi (Dokumentasi bagus, API matang) 71 | Lumayan (Perlu setup kalau host sendiri) 72 |
| Lisensi | Komersial, ada batasan | Open Source (Lisensi bebas, misal BSD) |

**Pertimbangan Memilih:**

* **Google Maps API:** Datanya bagus, fiturnya lengkap termasuk macet *real-time*, gampang dipakai. Tapi, biayanya bisa lumayan, apalagi buat VRP yang butuh matriks jarak (N x N). Tiap elemen matriks biasanya bayar, jadi kalau N-nya besar, biayanya bisa cepat bengkak.56
* **OSRM (pakai data OpenStreetMap):** Alternatif *open-source* yang kuat dan gratis (software-nya). Data OSM makin bagus. OSRM cepat banget ngitung rute terpendek/tercepat. Layanan 'Table'-nya efisien buat bikin matriks jarak/waktu.72 Tapi, kualitas data OSM bisa beda-beda. Data macet *real-time* nggak ada standarnya, perlu integrasi tambahan yang rumit. Kalau di-host sendiri, perlu server dan perawatan.56
* **Implikasi Biaya Matriks:** Pilihan API peta ini ngaruh banget ke biaya, terutama karena VRP butuh matriks jarak/waktu. Kalau naif bikin N² panggilan API titik-ke-titik, pakai Google Maps bisa mahal banget. Pakai API Matriks (Google atau OSRM) yang dirancang buat permintaan *many-to-many* jauh lebih hemat.56 Jadi, cara dapetin matriks jarak/waktu harus dioptimalkan biar biaya operasional terkendali.

Pilihan antara Google Maps dan OSRM (atau layanan lain kayak Mapbox 71) tergantung prioritas: data paling akurat dan macet *real-time* (cenderung ke Google Maps, kalau budget cukup) versus biaya lebih murah dan kontrol lebih besar (cenderung ke OSRM, kalau data OSM cukup bagus dan macet *real-time* bukan prioritas utama atau bisa diurus terpisah).

### **D. Pertimbangan Saat 'Go Live' (Deployment)**

Setelah sistem jadi, perlu di-deploy biar bisa dipakai pengguna. Beberapa pilihan umum:

* **Cloud Hosting:** Platform kayak Amazon Web Services (AWS), Google Cloud Platform (GCP), atau Microsoft Azure nawarin infrastruktur yang bisa disesuaikan ukurannya, layanan terkelola (basis data, *load balancing*, dll.), dan bayar sesuai pemakaian.13 Ini populer karena fleksibel dan gampang kalau mau nambah kapasitas.
* **Hosting Sendiri (On-Premise):** Jalanin aplikasi di server fisik milik sendiri. Kontrol penuh tapi butuh modal awal lebih besar buat hardware dan tanggung jawab penuh buat perawatan, keamanan, dan skalabilitas.3 Kurang fleksibel dibanding cloud.
* **Kontainerisasi (Containerization):** Pakai teknologi kayak Docker buat 'bungkus' aplikasi dan semua kebutuhannya jadi satu paket standar. Ini bikin lingkungan pengembangan, tes, dan produksi jadi konsisten, plus bikin proses deploy lebih simpel.14
* **Orkestrasi (Orchestration):** Buat aplikasi yang lebih kompleks atau butuh skala besar, platform orkestrasi kontainer kayak Kubernetes bisa dipakai buat otomatisasi deploy, skala, dan manajemen aplikasi dalam kontainer.14

Pilih strategi deploy harus mikirin budget, butuh skala seberapa besar, keahlian tim, dan syarat keamanan.

## **VI. Siapa Saja yang Terlibat dan Bagaimana Mereka Berinteraksi? (Perspektif Use Case)**

Penting banget buat tahu siapa aja yang bakal pakai sistem ini dan gimana cara mereka pakainya. Ini kunci buat bikin tampilan yang enak dipakai dan fitur yang pas. Kalau pakai kacamata Analisis Perancangan Berbasis Objek (OOAD), ini soal nentuin aktor dan *use case* utama.

### **A. Pengguna Sistem (Aktor)**

Aktor itu siapa aja (biasanya orang, tapi bisa juga sistem lain) yang berinteraksi sama sistem kita. Buat sistem optimasi rute pengiriman lokal ini, aktor utamanya:

1. **Administrator Sistem:**
   * **Perannya:** Yang megang kendali sistem secara keseluruhan.
   * **Interaksi Utama:** Ngatur akun pengguna (bikin, ubah, hapus akun Manajer Logistik, mungkin Sopir juga), setting parameter umum sistem (misal, settingan default, jenis mobil), pantau kondisi sistem.
2. **Manajer Logistik / Dispatcher:**
   * **Perannya:** Pengguna utama sehari-hari yang tugasnya bikin rencana rute.
   * **Interaksi Utama:** Masukin dan ngatur data utama (pelanggan/lokasi tujuan plus detailnya kayak alamat, koordinat, permintaan, jadwal, waktu layanan), ngatur data mobil (ID, kapasitas, ketersediaan, gudang), masukin atau impor data pesanan harian, milih pesanan dan mobil mana yang mau dioptimalkan, mulai proses optimasi, lihat dan analisis hasil rutenya (di peta dan daftar), kalau perlu bikin penyesuaian manual, terus akhirnya ngasih rute final ke sopir.5 Aktor inilah fokus utama fitur inti sistem.
3. **Sopir (Aktor Potensial/Tambahan):**
   * **Perannya:** Yang jalanin rute sesuai rencana.
   * **Interaksi Utama:** Bisa jadi nggak langsung (terima rute cetak atau lisan dari Manajer Logistik). Tapi, kalau sistemnya lebih canggih (mungkin di luar lingkup awal proyek ini), sopir bisa pakai aplikasi mobile buat:
     + Terima dan lihat detail rute (urutan kunjungan, alamat, instruksi khusus).
     + Update status pengiriman di tiap tempat (misal, terkirim, gagal, alasannya).19
     + Ambil bukti kirim elektronik (ePOD), kayak tanda tangan atau foto.19
     + Mungkin lihat navigasi belokan (kalau nyambung sama aplikasi navigasi).58
4. **Pelanggan (Aktor Tidak Langsung):**
   * **Perannya:** Penerima akhir barang atau layanan.
   * **Interaksi:** Pelanggan ngasih input ke proses logistik secara umum (lewat pesanan, alamat, preferensi waktu), dan terima hasilnya (barang dikirim). Tapi, mereka biasanya nggak langsung interaksi sama *alat perencanaannya*.17 Manfaat yang mereka rasakan (barang datang tepat waktu) itu hasil dari Manajer Logistik pakai sistem ini.

Fokus utama perancangan sistem ini adalah interaksi Manajer Logistik/Dispatcher. Kemampuan mereka buat masukin data dengan gampang, mulai optimasi, pahami hasilnya, dan bikin penyesuaian kalau perlu, itu kunci suksesnya sistem ini.5 Tampilan antarmuka harus dirancang sesuai alur kerja mereka.

### **B. Interaksi Kunci (Gambaran Kasus Penggunaan)**

Ini beberapa contoh *use case* utama yang gambarin interaksi aktor dan sistem, relevan buat pemodelan OOAD:

* **(Admin) Kelola Pengguna:** Tambah, lihat, ubah, hapus (CRUD) akun pengguna (Manajer, Sopir).
* **(Admin/Manajer) Kelola Kendaraan:** CRUD detail mobil (ID, nama/tipe, kapasitas, gudang asal, status).
* **(Manajer) Kelola Pelanggan/Lokasi:** CRUD detail pelanggan/tujuan (ID, nama, alamat, koordinat (otomatis/manual), kontak, catatan).
* **(Manajer) Kelola Pesanan/Permintaan:** CRUD detail pesanan (hubungkan pelanggan, tentukan jumlah, setel jadwal waktu (kalau ada), waktu layanan).
* **(Manajer) Hasilkan Rute Optimal:**
  1. Pilih pesanan mana aja yang mau dikirim.
  2. Pilih mobil mana aja yang siap jalan.
  3. Setel parameter optimasi (misal, tanggal, prioritas (kalau ada)).
  4. Mulai mesin optimasi.
  5. Terima dan tampilkan hasil: daftar rute per mobil (urutan kunjungan, perkiraan waktu), visualisasi di peta, ringkasan (total jarak/waktu, pemakaian mobil).
* **(Manajer) Tinjau dan Kirim Rute:**
  1. Analisis rute hasil sistem.
  2. (Opsional) Modifikasi manual sedikit (misal, pindah satu kunjungan antar rute kalau boleh dan dicek ulang validitasnya).
  3. Simpan/finalisasi rute.
  4. Kasih rute ke sopir (misal, cetak, kirim ke HP).
* **(Sopir - Opsional/Masa Depan) Lihat Rute Tugas:** Login aplikasi mobile, lihat daftar kunjungan, detail, dan urutan.
* **(Sopir - Opsional/Masa Depan) Update Status Kirim:** Tandai kunjungan selesai atau gagal, tambah catatan, ambil ePOD.19

### **C. Bagaimana Sistem Ini Masuk dalam Alur Kerja Logistik?**

Sistem optimasi rute ini penting, tapi cuma satu bagian dari alur kerja logistik yang lebih besar. Biasanya alurnya gini 5:

1. **Terima Pesanan:** Pesanan masuk dari pelanggan (telepon, web, dll.) dicatat di sistem manajemen pesanan (bisa jadi sistem beda).
2. **Rencanakan Rute (Fokus Sistem Ini):** Manajer Logistik pakai sistem optimasi buat kelompokin pesanan, tentuin mobil, dan bikin rute efisien berdasarkan data pesanan, pelanggan, mobil, dan aturan.
3. **Kasih Tugas (Dispatching):** Rute final dikasih ke sopir.
4. **Jalankan Rute:** Sopir jalan sesuai rute, kirim barang/layanan di tiap lokasi.
5. **Pantau (Opsional):** Manajer Logistik mungkin pantau kemajuan sopir *real-time* (kalau ada sistem pelacakan).
6. **Selesai & Lapor:** Sopir lapor tugas selesai (misal, balikin bukti kirim). Data aktual (waktu tempuh, waktu layanan) bisa dicatat buat analisis atau perbaikan rencana ke depan.

Meskipun Sopir mungkin nggak langsung pakai antarmuka perencanaannya, hasil sistem (rute) sangat mempengaruhi kerja mereka. Mengintegrasikan masukan dari Sopir atau data status pengiriman kembali ke sistem (walaupun awalnya manual oleh Manajer Logistik) bisa banget ningkatin kualitas perencanaan di masa depan.3 Contohnya, kalau sopir sering lapor waktu layanan di lokasi A lebih lama dari perkiraan, Manajer Logistik bisa update parameter waktu layanan di sistem biar optimasi berikutnya lebih akurat. Begitu juga info soal jalan susah atau kendala lain dari sopir bisa dipakai buat update data lokasi atau aturan rute. Memikirkan gimana data pelaksanaan rute ini bisa balik lagi ke sistem perencanaan, walau nggak langsung, akan nambah nilai jangka panjangnya.

## **VII. Amunisi Teknologi: Bahasa, Framework, dan Alat Bantu**

Milih teknologi yang pas itu krusial buat bikin sistem optimasi rute berbasis web yang jalan bagus, andal, dan gampang dirawat. Ini soal milih bahasa pemrograman, *framework*, basis data, API peta, *library* optimasi, dan alat bantu lainnya.

### **A. Bahasa Pemrograman**

* **Backend (Otak di Balik Layar):**
  + **Python:** Pilihan populer banget buat aplikasi web (pakai *framework* Django atau Flask). Ekosistemnya kaya buat sains data dan optimasi, termasuk dukungan kelas satu buat Google OR-Tools.65 Sintaksnya bersih dan gampang dibaca jadi nilai plus.75
  + **Java:** Bahasa matang, kuat, banyak dipakai di perusahaan besar. *Framework* Spring (khususnya Spring Boot) populer buat bikin aplikasi web tangguh.66 OR-Tools juga ada buat Java.20
  + **JavaScript (Node.js):** Bikin bisa pakai JavaScript buat semua bagian (*full-stack*). Cocok buat aplikasi yang banyak nunggu respons (I/O-bound) dan *real-time*. *Framework* Express sering dipakai.65
  + **C# (.NET):** Pilihan utama di ekosistem Microsoft, pakai *framework* ASP.NET Core yang modern.65 OR-Tools ada buat C#.20
  + **PHP:** Bahasa yang sangat banyak dipakai buat web, *framework*-nya matang kayak Laravel dan Symfony.65
  + **Ruby:** Terkenal karena *framework* Ruby on Rails yang bikin pengembangan cepat.66
  + **Go (Golang):** Dikenal kenceng, jagoan konkurensi, dan gampang di-deploy.68
* **Frontend (Tampilan Depan):**
  + **JavaScript:** Bahasa wajib buat interaksi di sisi pengguna.65 Hampir semua web modern pakai ini.
  + **TypeScript:** Versi JavaScript yang lebih 'tertib' dengan tipe data statis. Bagus buat proyek besar biar lebih andal dan gampang dirawat.70 Banyak *framework frontend* modern dukung TypeScript.
* **Saran:** **Python** sering jadi pilihan kuat buat proyek kayak gini. Alasannya? Ada *library* optimasi keren kayak OR-Tools, dukungan bagus buat analisis data (kalau nanti butuh), dan *framework* web matang kayak Django (lengkap) atau Flask (lebih fleksibel) yang bikin pengembangan lebih cepat.6

### **B. Web Frameworks (Kerangka Kerja Biar Cepat)**

* **Backend:** Pilihannya tergantung bahasa yang dipakai.
  + Python: **Django** (banyak fitur bawaan, cocok buat aplikasi berbasis data) 66, **Flask** (lebih simpel, bebas pilih komponen).66 FastAPI juga populer buat bikin API kenceng.70
  + Node.js: **Express.js** (simpel, fleksibel, standar) 66, NestJS (lebih terstruktur, pakai TypeScript).70
  + Java: **Spring Boot** (bikin gampang develop aplikasi Spring, populer di enterprise).68
  + PHP: **Laravel**, **Symfony**, CodeIgniter.69
  + Ruby: **Ruby on Rails** (produktif, banyak aturan baku).66
  + C#: **ASP.NET Core**.68
* **Frontend:**
  + **React:** *Library* super populer dari Facebook, basisnya komponen, ekosistemnya besar.65
  + **Angular:** *Framework* lengkap dari Google, pakai TypeScript, cocok buat aplikasi skala besar.67
  + **Vue.js:** Dianggap lebih gampang dipelajari, fleksibel, dokumentasinya bagus.67
  + **Svelte:** Pendekatan beda pakai kompilator, hasilnya kenceng.69

### **C. Basis Data (Tempat Nyimpen Data)**

* **SQL (Relasional):**
  + **PostgreSQL:** Kuat banget, sesuai standar SQL, fiturnya canggih, plus dukungan ekstensi geospasial **PostGIS** yang mantap.14 Pilihan bagus buat aplikasi yang banyak urusan sama lokasi.
  + **MySQL:** Populer banget, banyak didukung penyedia hosting, performa oke buat kasus umum.14
* **NoSQL (Non-Relasional):**
  + **MongoDB:** Basis data dokumen populer, skemanya fleksibel, bagus buat data nggak terstruktur.14
  + **Redis:** Penyimpanan data di memori, super cepat, sering dipakai buat *caching*, sesi, atau antrian pesan.14
* **Saran:** **PostgreSQL dengan ekstensi PostGIS** sangat direkomendasikan karena jago nyimpen, indeks, dan cari data geografis (koordinat, area) dengan efisien, pas banget buat aplikasi optimasi rute.72

### **D. API dan Library Peta**

* **API Backend:** Google Maps Platform (Distance Matrix API, Geocoding API, dll.), OSRM (Table Service, Route Service), Mapbox API, ArcGIS Platform API, Bing Maps API.7 Pilihannya tergantung perbandingan biaya, fitur, dan kualitas data (lihat Bagian V.C).
* **Library Frontend:** Buat nampilin peta interaktif dan rute di browser:
  + **Leaflet:** *Library* JavaScript *open-source* ringan dan populer.72
  + **Mapbox GL JS:** *Library* kuat buat peta vektor interaktif.71
  + **Google Maps JavaScript API:** Nyambung langsung sama ekosistem Google Maps.71
  + **OpenLayers:** *Library* *open-source* lain yang fiturnya kaya.

### **E. Library Optimasi (Fokus ke Google OR-Tools)**

* **Google OR-Tools:**
  + **Apa itu?** Kumpulan software *open-source* dari Google buat mecahin masalah optimasi kombinatorial, termasuk program linear, integer, *constraint programming*, dan masalah *routing*.6
  + **Dukungan VRP:** Punya modul *routing* canggih yang bisa nangani VRP dasar, CVRP, VRPTW, VRP dengan Ambil & Antar, dan aturan rumit lainnya.10
  + **Bahasa:** Ada API buat Python, C++, Java, dan C#.6
  + **Konsep Kunci Pemodelan VRP di OR-Tools (Agak Teknis Nih!):**
    - RoutingIndexManager: Ngatur pemetaan antara titik masalah VRP (gudang, pelanggan) sama indeks internal yang dipakai *solver*.54 Penting nih, karena *solver* bisa nambah titik internal.
    - RoutingModel: Kelas utama buat bangun model VRP. Dipakai buat nentuin jumlah titik, jumlah mobil, lokasi gudang.64
    - *Callbacks* (Fungsi Panggilan Balik): Fungsi yang kita bikin buat ngasih data spesifik ke *solver*. Contoh umum:
      * *Transit Callback:* Ngitung biaya (jarak/waktu) antar dua titik (pakai indeks *solver*). Didaftarin pakai RegisterTransitCallback atau RegisterUnaryTransitCallback.20
      * *Demand Callback:* Ngitung permintaan di tiap titik. Perlu buat aturan kapasitas.63
    - AddDimension: Metode penting buat nambahin aturan kumulatif sepanjang rute, kayak kapasitas, waktu, atau jumlah lain.20 Butuh *callback* transit (buat ngitung akumulasi), kapasitas dimensi (batas atas per rute), *slack* maksimum (waktu tunggu yang boleh, penting buat *time windows*), flag fix\_start\_cumul\_to\_zero (nilai kumulatif mulai dari 0 di gudang nggak?), dan nama dimensi.
    - AddDimensionWithVehicleCapacity: Metode bantu khusus buat nambah aturan kapasitas, pakai *demand callback*.63
    - SetArcCostEvaluatorOfAllVehicles: Nentuin tujuan utama, biasanya berdasarkan *transit callback* jarak atau waktu.54
    - **Modelin Time Windows:** Dibuat pakai AddDimension buat waktu. Terus, batasan jadwal waktunya diterapin ke variabel kumulatif waktu di tiap titik: time\_dimension.CumulVar(manager.NodeToIndex(lokasi\_pelanggan)).SetRange(waktu\_awal, waktu\_akhir).15
    - **Modelin Kapasitas:** Dibuat pakai AddDimension (atau AddDimensionWithVehicleCapacity) buat permintaan. Kapasitas mobil jadi batas atas dimensi ini.1
    - **Proses Nyari Solusi (Solving):** Pakai metode routing.Solve() atau routing.SolveWithParameters(). Parameter RoutingSearchParameters bisa ngatur strategi pencarian (misal, heuristik solusi pertama kayak PATH\_CHEAPEST\_ARC, SAVINGS; metaheuristik pencarian lokal kayak GUIDED\_LOCAL\_SEARCH, TABU\_SEARCH) dan batas waktu nyari.53
    - **Baca Hasil Solusi:** Solusi (kalau ketemu) bentuknya objek Assignment. Buat dapetin rute, telusuri NextVar(index) buat tiap mobil, mulai dari routing.Start(vehicle\_id) sampai ketemu routing.IsEnd(index). Pakai manager.IndexToNode(index) buat dapetin ID lokasi asli.53 Nilai dimensi (waktu tiba, sisa kapasitas) juga bisa diambil dari objek Assignment.
* **Alternatif Lain:**
  + Solver Komersial: Gurobi, CPLEX (kuat banget, lisensinya mahal).30
  + Open Source Lain: VROOM (fokus VRP, cepat) 83, jsprit (Java), OptaPlanner (Java, platform optimasi).84
  + Solver Akademik: VRPSolver (khusus VRP).82
* **Kelebihan OR-Tools:** Keseimbangan bagus antara kekuatan model, kinerja *solver*, dukungan buat banyak jenis VRP, *open-source*, dan dukungan banyak bahasa. Cocok banget buat proyek ini.10

Pakai *library* optimasi kayak OR-Tools itu sangat mempermudah implementasi algoritma VRP rumit dan ngatur kendala dibanding ngoding dari nol.6 Jadi, kita bisa lebih fokus ke logika aplikasi, integrasi sistem, dan tampilan pengguna, daripada pusing sama detail algoritma. Tapi, perlu diingat, pakai OR-Tools efektif itu butuh paham cara dia memodelkan masalah (pakai Dimensi, *Callbacks*, Indeks vs. Node), yang mungkin beda sama rumus VRP biasa.20 Pelajari dokumentasi dan contoh OR-Tools jadi kunci.

### **F. Alat Bantu Pengembangan**

* **Version Control System (VCS):** **Git** itu standar industri. Penting buat lacak perubahan kode, kerja bareng (kalau tim), dan balik ke versi lama kalau ada masalah.52 Platform kayak GitHub, GitLab, Bitbucket nyediain tempat simpen kode online dan fitur kolaborasi.
* **Integrated Development Environment (IDE) / Text Editor:** Pilihan populer: VS Code (serbaguna, banyak ekstensi), PyCharm (khusus Python), IntelliJ IDEA (khusus Java), dll. Bantu nulis kode, cari error (*debugging*), dan ngatur proyek.
* **Package Managers:** Sesuai bahasa: pip buat Python, npm atau yarn buat Node.js, Maven atau Gradle buat Java. Ngatur *library* luar yang dipakai.
* **Testing Frameworks:** Penting buat mastiin kode berkualitas. Contoh: PyTest atau unittest buat Python, JUnit buat Java, Jest atau Mocha buat JavaScript.
* **Containerization:** **Docker** berguna banget buat bikin lingkungan pengembangan, tes, dan produksi yang konsisten dan terpisah.14 Bikin gampang deploy aplikasi plus semua kebutuhannya.

Milih tumpukan teknologi harus seimbang antara keakraban tim, kebutuhan performa, kematangan ekosistem, kebutuhan skala, dan kecepatan pengembangan.68 Kombinasi Python + Django/Flask + React + PostgreSQL + OR-Tools itu tumpukan yang umum, kuat, dan didukung baik buat aplikasi optimasi rute web kayak gini.70

## **VIII. Membingkai Proyek Anda: Menentukan Arah Sistem Optimasi**

Setelah paham konsep VRP, algoritma, arsitektur, dan teknologinya, saatnya membingkai proyek Tugas Akhir ini lebih spesifik, sesuai konteks dan batasan yang ada.

### **A. Usulan Judul Proyek (Contoh Biar Lebih Spesifik)**

Judul awal "Sistem Optimasi Rute Pengiriman Barang Berbasis Website" itu masih umum banget. Judul yang lebih fokus lebih bagus buat Tugas Akhir. Ini beberapa alternatif yang lebih spesifik:

1. **"Optimasi Rute Pengiriman Harian Produk UMKM dalam Kota via Web Menggunakan Algoritma Tabu Search"** (Fokus: UMKM, Harian, Satu Kota, Algoritma TS).85
2. **"Sistem Perencanaan Rute Kirim Lokal via Web: Atasi Kendala Kapasitas & Jadwal Waktu (Studi Kasus: Distribusi Snack di Jakarta Selatan)"** (Fokus: Lokal, CVRP+VRPTW, Contoh Kasus).
3. **"Bikin Aplikasi Web Optimasi Rute Kurir Skala Kota Pakai Google OR-Tools"** (Fokus: Bikin Aplikasi, Skala Kota, Alat Spesifik).
4. **"Analisis & Rancangan Sistem Web Optimasi Rute Kirim Jarak Pendek buat Pedagang Pasar"** (Fokus: OOAD, Jarak Pendek, Target Pengguna).
5. **"Implementasi Algoritma Clarke-Wright Savings & Nearest Neighbor di Sistem Web buat Optimasi Rute Logistik UMKM"** (Fokus: Implementasi Heuristik, UMKM).

Judul-judul ini bikin cakupannya lebih sempit, nunjukkin fitur kunci (aturan, algoritma, alat, konteks), dan kedengeran lebih pas buat proyek akademik.85

### **B. Contoh Latar Belakang Masalah (Versi Lebih Mengalir)**

Dalam dunia bisnis, efisiensi pengiriman barang sampai ke tangan pelanggan (*last-mile delivery*) itu jadi kunci persaingan, termasuk buat Usaha Mikro, Kecil, dan Menengah (UMKM) di kota-kota besar Indonesia. Teman-teman UMKM yang jualan atau produksi barang seringkali harus kirim produk ke banyak pelanggan yang lokasinya mencar-mencar di dalam kota, padahal armadanya terbatas.1 Kalau masih ngatur rute pakai cara manual, wah, repot! Makan waktu, gampang salah, susah mikirin kapasitas mobil atau permintaan jadwal khusus pelanggan, dan hasilnya sering nggak efisien.1 Akibatnya? Biaya operasional jadi tinggi, terutama bensin, waktu sopir nggak maksimal, mobil cepat rusak. Belum lagi kalau telat kirim, pelanggan bisa kecewa. Nah, tantangan ini di dunia akademis dikenal sebagai *Vehicle Routing Problem* (VRP).7 Meskipun VRP ini rumit 8, teknologi optimasi modern bisa bantu. Makanya, perlu banget ada sistem optimasi rute berbasis web yang gampang dipakai dan terjangkau buat UMKM kita, biar mereka bisa otomatis bikin rencana rute, lebih efisien, dan hemat biaya logistik.13

### **C. Contoh Rumusan Masalah (Versi Lebih Fokus)**

1. Gimana cara rancang arsitektur dan bikin prototipe sistem optimasi rute web yang bisa nentuin urutan kunjungan terbaik buat satu atau lebih mobil dari satu gudang ke pelanggan UMKM di satu wilayah? Tujuannya biar total jarak tempuh paling sedikit, sambil tetap mikirin kapasitas muatan mobil (CVRP).
2. Gimana cara masukin aturan jadwal waktu kirim (VRPTW) ke model optimasi dan prototipe sistem web ini biar bisa nurutin permintaan jadwal pelanggan?
3. Seberapa bagus sih kinerja algoritma (dilihat dari kualitas solusi/total jarak dan waktu ngitung) buat nyelesaiin contoh kasus VRP/CVRP/VRPTW skala pengiriman UMKM lokal?
4. Gimana cara bikin tampilan pengguna web yang gampang dimengerti sama pengguna (misal, pemilik UMKM atau staf logistik) buat masukin data, jalanin optimasi, dan lihat hasil rutenya di peta?

### **D. Contoh Lingkup/Batasan Masalah (Biar Jelas Batasannya)**

1. **Fokus Varian:** Proyek ini fokus ke model *Capacitated Vehicle Routing Problem* (CVRP) dulu, mungkin nambah *Vehicle Routing Problem with Time Windows* (VRPTW) kalau waktunya cukup.
2. **Area Geografis:** Terbatas buat rute di satu area aja (misal, satu kecamatan atau kota).
3. **Struktur Logistik:** Anggap cuma ada satu gudang buat berangkat dan pulang.
4. **Armada:** Pakai mobil yang sama semua (kapasitas sama) atau maksimal 2-3 jenis mobil dengan kapasitas jelas.
5. **Algoritma & Alat:** Optimasi pakai *library* Google OR-Tools dengan Python. Strategi pencariannya pakai.
6. **Data:** Pakai data statis (lokasi, permintaan, jadwal udah tahu dari awal). Nggak ngurusin macet *real-time* atau pesanan dadakan. Jarak/waktu antar lokasi didapat dari.
7. **Fitur Prototipe:** Aplikasi webnya punya fungsi inti: input data pelanggan, pesanan (termasuk permintaan & jadwal kalau ada), mobil (kapasitas), mulai optimasi, dan tampilkan hasil rute di peta (pakai Leaflet/Google Maps API) plus daftar kunjungan per mobil. Belum ada login banyak pengguna, lacak *real-time*, atau laporan canggih.
8. **Evaluasi:** Kinerja diuji pakai data simulasi atau contoh kasus rekaan skala kecil (misal, 1 gudang, 10-30 pelanggan, 2-5 mobil).

Bikin batasan yang jelas kayak gini penting banget buat proyek VRP, apalagi buat tugas kuliah yang waktu dan sumber dayanya terbatas. Kalau coba nanganin terlalu banyak jenis VRP atau kerumitan dunia nyata (macet dinamis, mobil beda-beda, ambil-antar barengan) sekaligus, proyeknya bisa jadi nggak kelar.21 Fokus ke masalah inti yang jelas (misal, CVRP/VRPTW satu gudang, mobil sama, data statis) bikin peluang sukses lebih besar dan jadi dasar bagus buat pengembangan selanjutnya.

### **E. Contoh Tujuan Proyek (Target yang Mau Dicapai)**

1. Merancang arsitektur sistem optimasi rute web pakai pendekatan berorientasi objek, termasuk model data, komponen sistem (frontend, backend, database, mesin optimasi), dan cara kerjanya.
2. Mengimplementasikan mesin optimasi pakai Google OR-Tools (Python) buat mecahin masalah CVRP (dan VRPTW kalau bisa) berdasarkan data yang dimasukkan.
3. Membangun prototipe aplikasi web (misal, pakai Python Flask/Django dan JavaScript/React) yang punya tampilan buat:
   * Input data pelanggan (nama, alamat/koordinat, permintaan).
   * Input data mobil (jumlah, kapasitas).
   * (Kalau VRPTW) Input jadwal waktu pelanggan.
   * Jalanin proses optimasi.
   * Nampilin hasil rute (urutan kunjungan per mobil, total jarak/waktu) di peta (pakai Leaflet/Google Maps API) dan dalam bentuk teks.
4. Menguji fungsi prototipe dan ngukur waktu komputasi serta kualitas solusi (total jarak/waktu) pakai minimal satu contoh kasus simulasi pengiriman lokal.

### **F. Contoh Manfaat Proyek (Hasil yang Diharapkan)**

1. Nghasilin prototipe aplikasi web sebagai alat bantu praktis buat UMKM atau distributor kecil biar bisa rencanain rute lebih efisien dibanding cara manual.
2. Berpotensi ngurangin biaya operasional logistik (bensin, waktu) buat pengguna lewat rute yang optimal.1
3. Nyediain platform buat nunjukkin dan belajar konsep VRP, algoritma optimasi (khususnya yang dipakai), dan teknologi web terkait.
4. Memenuhi syarat Tugas Akhir mata kuliah Analisis Perancangan Berbasis Objek dengan bikin hasil rancangan (misal, diagram kelas, diagram *use case*) dan prototipe software yang terapin konsep itu.
5. (Kalau VRPTW diimplementasikan) Berpotensi ningkatin layanan pelanggan lewat jadwal kirim yang lebih baik.1

Pembingkaian proyek yang bagus itu harus nyambungin konsep teori VRP dan algoritma sama masalah nyata pengiriman lokal yang dihadapi target pengguna (misal, UMKM). Latar belakang, rumusan masalah, dan tujuan harus jelas nunjukkin *kenapa* teori dan implementasi teknis ini perlu buat ngatasin tantangan nyata. Ini bakal bikin cerita dan alasan proyeknya jadi lebih kuat.

## **IX. Kesimpulan dan Langkah Selanjutnya**

### **A. Intisari Pembahasan**

Laporan ini udah mengupas tuntas soal konsep, algoritma, dan rancangan sistem optimasi rute pengiriman barang berbasis web, khususnya buat skala lokal atau UMKM. Poin-poin pentingnya:

* **Dasar VRP:** *Vehicle Routing Problem* (VRP) itu masalah optimasi dasar di logistik buat bikin rute mobil yang efisien. Varian penting buat pengiriman lokal adalah *Capacitated VRP* (CVRP) yang mikirin kapasitas muatan, dan *VRP with Time Windows* (VRPTW) yang nambahin aturan jadwal waktu pelanggan.
* **Algoritma Optimasi:** Ada banyak jurus, dari heuristik konstruktif yang cepat tapi hasilnya biasa aja (*Clarke-Wright Savings*, *Nearest Neighbor*), sampai metaheuristik yang lebih canggih dan bisa nemu solusi bagus (*Tabu Search*, *Simulated Annealing*, *Genetic Algorithms*) tapi butuh waktu ngitung lebih lama. Pilihannya tergantung kebutuhan kualitas solusi vs kecepatan.
* **Variabel dan Kendala:** Model VRP yang akurat butuh definisi jelas soal data input (lokasi, permintaan, mobil, matriks jarak/waktu, jadwal) dan aturan main (kapasitas, jadwal, kunjungan sekali, dll.).
* **Arsitektur Sistem:** Arsitektur tiga lapis (*frontend*, *backend*, *database*) itu standar buat aplikasi web. Komponen kunci sistem optimasi rute meliputi tampilan pengguna, server aplikasi, basis data, sambungan ke API Peta (penting buat data jarak/waktu, pilihan kayak Google Maps atau OSRM punya plus minus biaya & fitur), dan mesin optimasi sebagai otaknya. Interaksi antar komponen, terutama *backend* dan mesin optimasi (yang bisa lemot), perlu dirancang baik, mungkin pakai proses asinkron.
* **Aktor dan Interaksi:** Pengguna utama sistem ini Manajer Logistik/Dispatcher. Desain tampilan dan alur kerja harus fokus ke kebutuhan mereka. Sopir itu aktor pendukung yang jalanin rute, masukan dari mereka bisa bikin perencanaan ke depan lebih bagus.
* **Teknologi:** Tumpukan teknologi yang disarankan: Python (dengan Django/Flask) buat *backend*, JavaScript (dengan React/Angular/Vue) buat *frontend*, PostgreSQL (dengan PostGIS) buat basis data, API Peta yang pas (Google Maps/OSRM), dan *library* optimasi kayak Google OR-Tools. OR-Tools nawarin cara ampuh dan relatif gampang buat implementasi berbagai algoritma dan aturan VRP.
* **Pembingkaian Proyek:** Penting banget nentuin lingkup proyek yang jelas, pilih judul spesifik, dan bikin latar belakang, masalah, tujuan, serta manfaat yang nyambungin teori VRP sama aplikasi praktis buat target pengguna (misal, UMKM).

### **B. Keterbatasan Studi/Sistem yang Diusulkan**

Prototipe sistem yang diusulkan ini punya beberapa keterbatasan yang perlu kita sadari:

1. **Data Statis:** Sistem ini anggap data inputnya tetap (lokasi, permintaan, jadwal udah tahu dari awal). Belum bisa nangani kondisi dinamis kayak pesanan baru muncul pas rute jalan atau perubahan macet *real-time*.
2. **Varian VRP Terbatas:** Fokusnya ke CVRP dan/atau VRPTW. Varian lebih rumit kayak VRP Ambil & Antar (VRPPD), mobil beda-beda, banyak gudang, atau aturan sopir (istirahat) belum dibahas mendalam.
3. **Kualitas Data Input:** Hasil optimasi bagus atau nggaknya tergantung banget sama kualitas data input, terutama matriks jarak/waktu. Kalau pakai perkiraan kasar atau data API peta tanpa penyesuaian lokal, hasilnya bisa kurang pas.
4. **Algoritma Spesifik:** Kinerja solusi tergantung algoritma dan settingan yang dipilih (misal, strategi pencarian bawaan OR-Tools). Algoritma lain atau settingan parameter yang beda mungkin ngasih hasil beda atau lebih baik.
5. **Fokus Perencanaan:** Sistem ini fokus ke tahap bikin rencana rute. Belum mencakup pelaksanaan, pantauan *real-time*, atau analisis otomatis setelah perjalanan selesai.

### **C. Arah Pengembangan Selanjutnya**

Melihat temuan dan keterbatasan tadi, ada beberapa ide buat pengembangan atau penelitian selanjutnya:

1. **Tangani Dinamisme:** Bikin sistem bisa nangani pesanan baru atau perubahan mendadak (*dynamic VRP*) dan sambungin data macet *real-time* dari API peta biar hitungan waktu lebih akurat dan bisa ubah rute di tengah jalan.3
2. **Varian VRP Lebih Lanjut:** Kembangin model biar bisa dukung varian VRP yang lebih rumit sesuai kebutuhan bisnis, kayak:
   * *VRP with Pickup and Delivery* (VRPPD).1
   * Armada Beda-beda (*Heterogeneous Fleet VRP*).45
   * Banyak Gudang (MDVRP).
   * Aturan Sopir (misal, jam kerja, istirahat wajib).7
3. **Integrasi Machine Learning (ML):** Manfaatin ML buat:
   * Prediksi waktu tempuh lebih akurat pakai data historis dan kondisi sekarang.17
   * Prediksi permintaan pelanggan.18
   * Prediksi waktu layanan di lokasi pelanggan.
   * Belajar preferensi rute atau otomatis setel parameter algoritma.
4. **Bikin Aplikasi Mobile Sopir:** Bikin aplikasi pendamping buat sopir biar bisa terima rute digital, navigasi, update status kirim *real-time*, dan ambil ePOD.19 Ini bakal bikin aliran data balik lebih cepat.
5. **Analitik dan Laporan Canggih:** Tambah fitur dasbor buat analisis kinerja logistik lama, bandingin rencana vs. aktual, lacak KPI (Key Performance Indicators), dan bikin laporan buat manajemen.3
6. **Eksplorasi Algoritma Canggih:** Coba atau implementasi algoritma optimasi yang lebih mutakhir atau gabungan (misal, *Adaptive Large Neighborhood Search* - ALNS 21, gabungan GA sama TS 25, atau pendekatan berbasis *Reinforcement Learning* 10).
7. **Optimasi Multi-Tujuan:** Pertimbangkan beberapa tujuan sekaligus, misal, minimalkan biaya sambil maksimalkan kepuasan pelanggan atau bikin beban kerja sopir seimbang.21

Intinya, sistem optimasi rute yang sukses itu jarang statis. Dunia logistik terus berubah, begitu juga teknologinya. Jadi, sistem optimasi rute idealnya dirancang biar bisa terus diperbaiki, datanya disempurnakan berdasarkan pengalaman lapangan, dan bisa adaptasi sama aturan baru atau teknologi baru kayak data *real-time* atau AI/ML.3 Merancang sistem yang modular dan bisa dikembangkan, meskipun nggak semua fitur dibuat di awal, bakal sangat berguna buat jangka panjangnya. Prototipe yang dibahas di laporan ini adalah langkah awal yang penting ke arah sana.