BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pelaksanaan kerja magang dilakukan di PT CreateIT Solution Indonesia, sebuah perusahaan yang bergerak di bidang pengembangan perangkat lunak berbasis teknologi informasi. Periode magang dilaksanakan pada Divisi Pengembangan Web (*Web Module*) dengan posisi sebagai *Backend Developer*. Dalam struktur organisasi proyek, supervisi dilakukan secara langsung oleh Vincentius Marco Melandri selaku Supervisor Lapangan, yang berperan dalam memberikan arahan teknis, bimbingan profesional, serta evaluasi terhadap setiap tahapan pekerjaan yang dilakukan.

Pelaksanaan magang berlangsung dalam proyek pengembang proyek Warehouse Management System (WMS) yang dikembangkan untuk Ritra Logistics, salah satu klien PT CreateIT Solution Indonesia. Dalam tim tersebut, peran yang ditugaskan adalah sebagai Backend Developer, berfokus pada perancangan dan pengembangan Application Programming Interface (API) menggunakan bahasa pemrograman Go (Golang) dengan framework Go Fiber. Pendekatan pengembangan backend berbasis RESTful API dipilih karena mampu meningkatkan modularitas sistem serta mempermudah integrasi dengan layanan frontend dan sistem lain [4]. Pekerjaan dilakukan secara kolaboratif dengan seorang rekan yang juga bertugas di sisi backend, serta melakukan koordinasi rutin dengan tim frontend yang menggunakan Next.js dan TypeScript untuk memastikan kesesuaian antara API dan antarmuka pengguna, sebagaimana praktik kolaborasi lintas tim yang umum diterapkan dalam pengembangan perangkat lunak modern [5].

Kegiatan magang dilaksanakan secara hybrid, di mana sebagian pekerjaan dilakukan secara daring dan sebagian lainnya secara tatap muka di kantor PT CreateIT Solution Indonesia. Proses koordinasi dilakukan melalui berbagai platform komunikasi dan manajemen proyek seperti Slack, Google Meet, dan Trello, yang digunakan untuk pembagian tugas, pelaporan kemajuan, serta diskusi teknis terkait pengembangan sistem. Penggunaan tools kolaborasi digital terbukti mampu meningkatkan efektivitas komunikasi tim serta transparansi progres pengembangan perangkat lunak [6]. Evaluasi dan pelaporan progres dilakukan

secara mingguan, di mana setiap anggota tim menyampaikan capaian pekerjaan, kendala yang dihadapi, serta rencana tindak lanjut yang akan dilakukan pada iterasi berikutnya.

Melalui mekanisme koordinasi tersebut, setiap tahapan pengembangan sistem dapat berjalan secara terstruktur, terpantau, dan sesuai dengan target fungsional yang telah ditetapkan. Selain memastikan keterpaduan antar bagian sistem, kegiatan ini juga menjadi sarana bagi mahasiswa untuk mengembangkan kemampuan teknis dan profesional dalam lingkungan kerja pengembangan perangkat lunak yang sesungguhnya.

3.2 Tugas yang Dilakukan

Selama masa magang di PT CreateIT Solution Indonesia, posisi yang ditempati berada pada divisi Backend Development dalam proyek Warehouse Management System (WMS) untuk klien Ritra Logistics. Tanggung jawab utama mencakup pengembangan sisi backend sistem berbasis web menggunakan bahasa pemrograman Golang dengan framework Go Fiber. Penggunaan bahasa Go dipilih karena keunggulannya dalam performa, konkurensi, serta kemudahan pemeliharaan sistem berskala besar [7].

Pekerjaan difokuskan pada perancangan dan implementasi Application Programming Interface (API) yang berfungsi sebagai penghubung antara basis data dan antarmuka pengguna yang dikembangkan oleh tim frontend menggunakan Next.js dan TypeScript. Setiap API dikembangkan berdasarkan API contract yang terdokumentasi melalui Swagger, sehingga komunikasi data antara backend dan frontend berjalan secara konsisten, terstruktur, dan terdokumentasi dengan baik [8].

Pengembangan dilakukan untuk berbagai entitas dalam sistem, seperti modul Purchase Order (PO), Manajemen Barang, dan Inventori Gudang. Proses pengembangan mencakup pembuatan endpoint untuk operasi Create, Read, Update, dan Delete (CRUD), serta fitur pencarian data berdasarkan UUID maupun ID. Pendekatan CRUD ini merupakan praktik umum dalam pengelolaan data terstruktur pada sistem informasi modern [9]. Selain itu, sistem autentikasi dan otorisasi pengguna juga diterapkan untuk memastikan pembatasan akses sesuai dengan peran masing-masing pengguna dalam sistem [10].

Proses kerja dilaksanakan secara kolaboratif dengan anggota tim lain melalui platform Gitea sebagai sistem version control dan OpenProject untuk manajemen tugas serta pelacakan progres proyek. Koordinasi dilakukan melalui

daily meeting, baik secara daring menggunakan Microsoft Teams maupun secara langsung di kantor, serta komunikasi informal menggunakan WhatsApp untuk pembahasan teknis harian. Setiap perubahan kode melalui proses code review sebelum digabungkan (merge) ke cabang utama repositori, guna menjaga kualitas kode dan meminimalkan potensi kesalahan implementasi [11].

Pengujian terhadap setiap *endpoint* dilakukan menggunakan Postman untuk memastikan bahwa seluruh fungsi API berjalan sesuai spesifikasi dan menghasilkan respons data yang tepat. Tahapan ini juga digunakan untuk memverifikasi kesesuaian API dengan kebutuhan *frontend* sebelum proses integrasi dilakukan.

Melalui proses tersebut, kegiatan pengembangan sistem berjalan secara terstruktur, efisien, dan terkoordinasi dengan baik, menghasilkan komponen *backend* yang stabil dan sesuai dengan spesifikasi fungsional sistem Warehouse Management System.

3.3 Uraian Pelaksanaan Magang

Proses pelaksanaan kegiatan magang di PT CreateIT Solution Indonesia berlangsung selama enam bulan sesuai dengan ketentuan yang tercantum dalam kontrak magang. Kegiatan magang dimulai pada tanggal 4 Agustus 2025 dan berakhir pada 3 Februari 2026. Selama periode tersebut, aktivitas magang dilaksanakan secara rutin pada hari kerja. Adapun hari Sabtu, Minggu, hari libur nasional, serta hari libur perusahaan tidak termasuk dalam jadwal pelaksanaan magang.

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Mempelajari struktur kode backend, standar penamaan variabel dan fungsi, serta mulai mengembangkan modul autentikasi dengan membuat repository, service, dan handler untuk proses registrasi dan verifikasi pengguna, termasuk pengelolaan OTP.

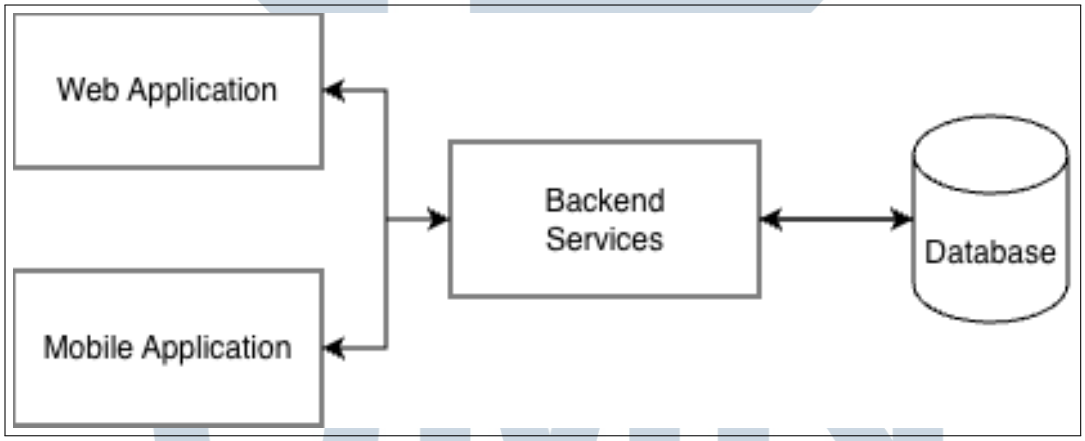
Minggu Ke -	Pekerjaan yang dilakukan
2	Melanjutkan pengembangan modul autentikasi dengan penambahan field dan logic verifikasi pengguna, serta memulai pengerjaan frontend proyek Seilbahnen melalui persiapan aset parallax dan implementasi animasi scroll parallax.
3	Mengimplementasikan smooth scrolling untuk meningkatkan pengalaman parallax, memastikan responsivitas tampilan pada berbagai resolusi layar, serta melakukan perbaikan animasi scrolling parallax.
4	Memulai pengerjaan backend proyek Seilbahnen dengan melakukan remodeling struktur database dan penyesuaian response API sesuai permintaan klien dan arahan supervisor.
5	Melanjutkan pengembangan modul Activity dengan menambahkan field yang belum tersedia pada response API serta melakukan perbaikan response agar sesuai dengan kebutuhan frontend.
6	Menyelesaikan pengembangan modul Activity dan memulai pengembangan fitur Winter dengan pembuatan model, DTO, handler, service, dan repository sesuai standar proyek.
7	Melanjutkan pengembangan fitur Winter, memperbaiki error response, membantu pengerjaan halaman admin frontend, serta memastikan integrasi backend dan frontend berjalan dengan baik.
8	Melakukan penyesuaian lanjutan pada fitur Winter sesuai permintaan klien dan memulai pengembangan modul Risk dengan pembuatan model, validator, request, response, serta handler, service, dan repository.
9	Melakukan pengembangan API Risk dengan dukungan multibahasa dan endpoint tambahan, serta memulai proyek RITRA dengan membangun sistem autentikasi berbasis JWT untuk aplikasi web dan mobile.

Minggu Ke -	Pekerjaan yang dilakukan
10	Mengembangkan fitur lanjutan pada modul Winter dengan penambahan logic weekday, weekend, dan holiday, memperbaiki bug minor, serta mengimplementasikan logic baru pada sisi frontend.
11	Melakukan optimalisasi tampilan parallax pada proyek Seilbahnen dengan mengganti aset gambar sesuai permintaan klien, menambahkan animasi scrolling, dan melakukan kompresi gambar untuk meningkatkan performa.
12	Menyesuaikan struktur dan kualitas parallax, mengatur kecepatan animasi scrolling, serta mengganti background tertentu menjadi video untuk meningkatkan pengalaman visual pengguna.
13	Melakukan optimalisasi kualitas gambar parallax dan melanjutkan pengembangan proyek RITRA dengan menambahkan API permissions, termasuk fitur bulk assign dan bulk remove.
14	Melakukan perbaikan logic dan response pada beberapa API, menyesuaikan kebutuhan frontend, serta mengoptimalkan logic pada PO service agar lebih efisien dan sesuai kebutuhan klien.
15	Melanjutkan pengembangan modul PO dan HU dengan menambahkan field baru, memisahkan model delivery, menghubungkan relasi antar model, serta melakukan penyesuaian aset visual sesuai permintaan klien.
16	Melakukan perbaikan pada tampilan parallax dan frontend, menambahkan field timestamp pada beberapa response API, serta menyesuaikan struktur response PO dan delivery.
17	Melakukan perbaikan minor bug pada beberapa API, menambahkan status stuffing pada PO, serta mengubah value status PO menjadi enumerasi yang lebih terstruktur sesuai kebutuhan sistem.

3.3.1 Gambaran Umum Arsitektur Sistem

Arsitektur sistem Ritra Logistics dirancang secara terpusat dengan satu layanan *backend* yang digunakan oleh dua jenis *platform*, yaitu aplikasi *web* dan aplikasi *mobile*. Aplikasi *web* digunakan oleh admin untuk melakukan proses persetujuan (*approval*) serta pengelolaan data operasional, sedangkan aplikasi *mobile* digunakan oleh petugas lapangan untuk melakukan pemindaian *barcode* dan input data barang. Skema arsitektur aplikasi Ritra Logistics dapat dilihat pada Gambar 3.1.

Kedua aplikasi tersebut terhubung ke sistem *backend* yang sama melalui antarmuka layanan berbasis API. *Backend* sistem berperan sebagai pusat pengolahan data dan logika bisnis, termasuk validasi data, pengelolaan proses gudang, serta komunikasi dengan basis data. Seluruh data operasional disimpan dalam satu database terpusat sehingga konsistensi data antara aplikasi *web* dan aplikasi *mobile* dapat terjaga.



Gambar 3.1. Skema arsitektur umum sistem aplikasi Ritra Logistics

Dengan arsitektur ini, sistem mampu mendukung proses operasional gudang secara terintegrasi, di mana data yang diinput melalui aplikasi *mobile* dapat langsung dikelola dan diverifikasi oleh admin melalui aplikasi *web*.

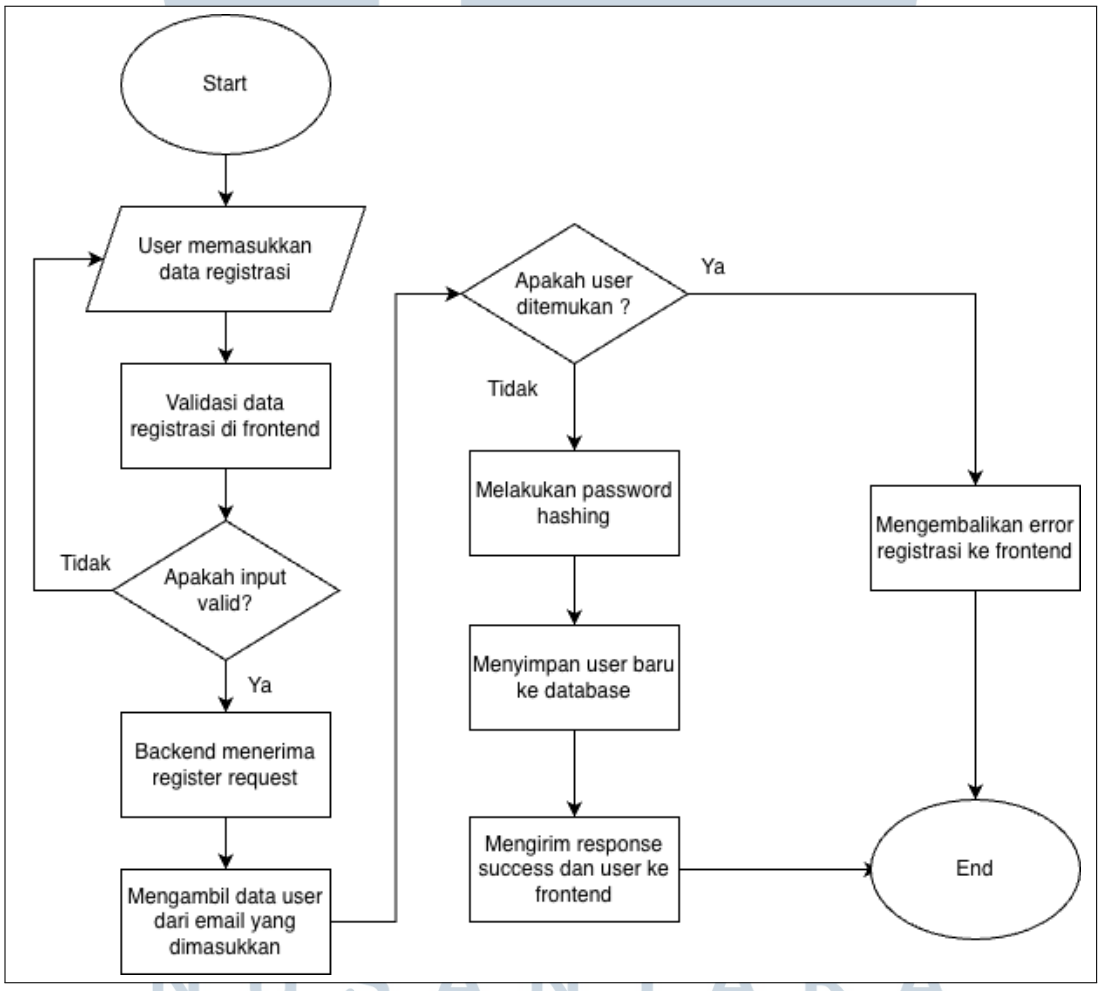
3.3.2 Analisis Perancangan Sistem

Perancangan sistem *backend* pada *Warehouse Management System* (WMS) Ritra Logistics dilakukan berdasarkan rancangan awal yang telah disusun oleh *Supervisor* Lapangan. Rancangan tersebut berfungsi sebagai acuan utama dalam

proses pengembangan sistem *backend*, khususnya dalam menentukan alur proses bisnis, urutan eksekusi logika program, serta interaksi antar modul dalam sistem.

Pendekatan perancangan yang digunakan tidak berfokus pada pemodelan berbasis objek secara penuh, melainkan menitikberatkan pada alur proses operasional sistem. Dengan tujuan memperjelas perancangan sistem yang telah dibuat, *flowchart* digunakan untuk menggambarkan tahapan-tahapan proses yang terjadi dalam sistem *backend*, mulai dari penerimaan permintaan (*request*) hingga pengolahan data dan pengembalian respons (*response*).

A Flowchart Register User



Gambar 3.2. Flowchart *register user*

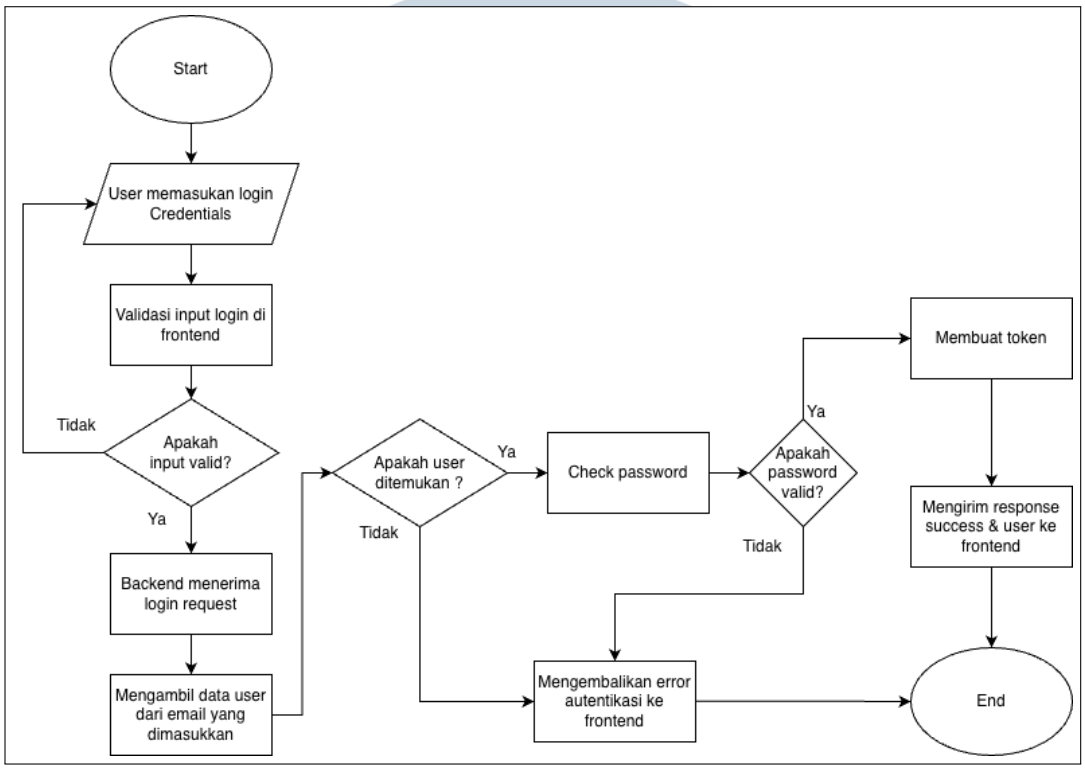
Flowchart proses registrasi pengguna menggambarkan alur pendaftaran akun baru pada sistem Ritra Logistics, sebagaimana ditunjukkan pada Gambar

3.2. Proses diawali ketika pengguna memasukkan data registrasi berupa email dan kata sandi melalui antarmuka aplikasi. Setelah data diinput, sistem *frontend* melakukan validasi awal untuk memastikan seluruh data telah diisi dengan benar dan memenuhi ketentuan yang ditetapkan, seperti panjang karakter dan kelengkapan field. Apabila data tidak lolos validasi, pengguna akan diminta untuk memperbaiki atau melengkapi kembali data hingga sesuai dengan aturan yang berlaku.

Apabila proses validasi pada sisi *frontend* berhasil, data registrasi akan dikirimkan ke *backend* melalui *request* API untuk diproses lebih lanjut. *Backend* kemudian melakukan pengecekan terhadap email yang digunakan dengan mencocokkannya pada basis data. Jika email tersebut telah terdaftar sebelumnya, sistem akan mengembalikan respons berupa pesan kesalahan yang menandakan bahwa proses registrasi tidak dapat dilanjutkan.

Jika email belum terdaftar, *backend* akan melanjutkan proses dengan melakukan *hashing* terhadap kata sandi pengguna sebagai langkah pengamanan data sebelum penyimpanan. Setelah itu, data pengguna baru akan disimpan ke dalam basis data dan sistem akan mengirimkan respons *success* beserta informasi data pengguna kepada *frontend*. Dengan alur tersebut, proses registrasi dapat berjalan secara aman, terstruktur, dan sesuai dengan rancangan sistem.

B Flowchart Login User



Gambar 3.3. Flowchart *login user*

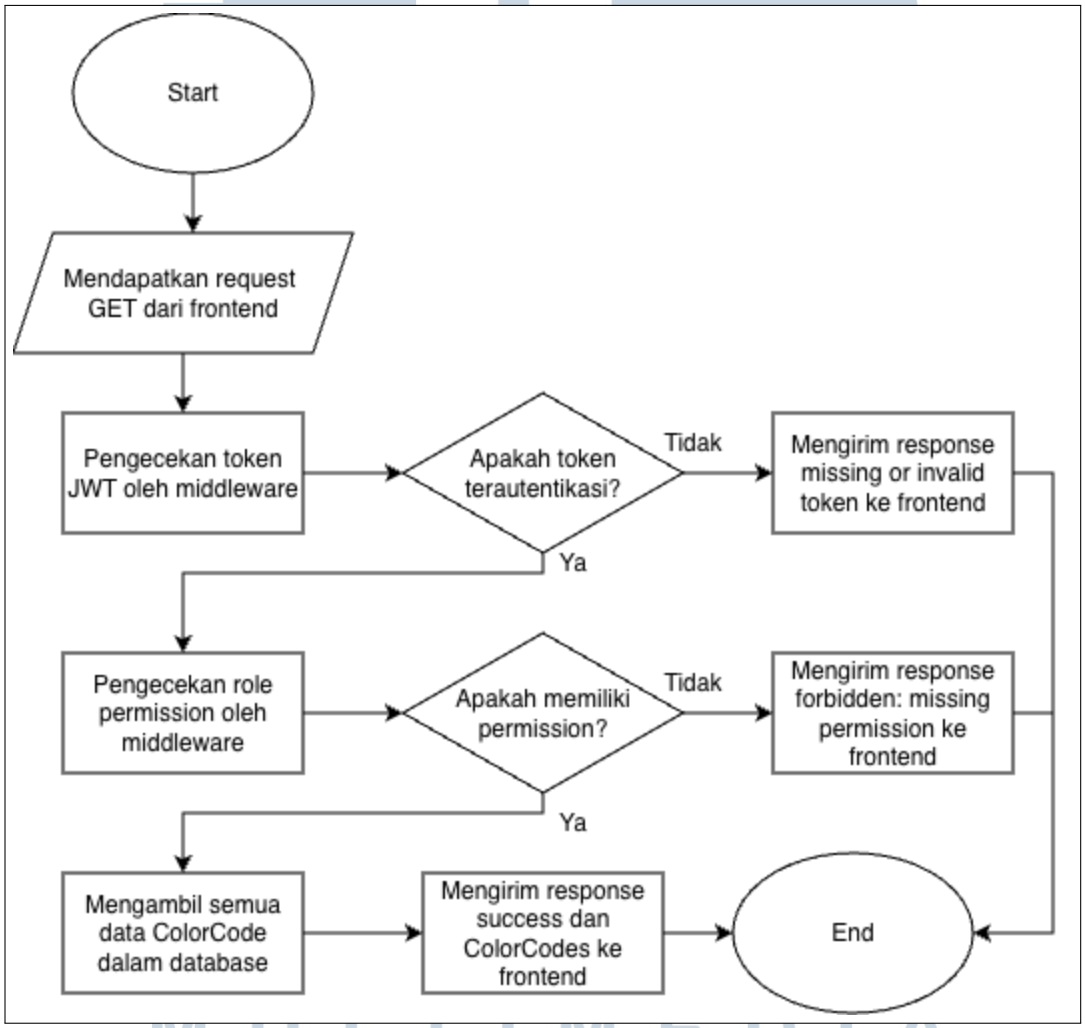
Flowchart proses *login* pengguna menggambarkan alur autentikasi pengguna pada sistem Ritra Logistics, sebagaimana ditunjukkan pada Gambar 3.3. Proses dimulai ketika pengguna memasukkan kredensial *login* berupa email dan kata sandi melalui antarmuka aplikasi. Selanjutnya, sistem *frontend* melakukan validasi awal terhadap *input* yang diberikan, seperti pengecekan format email, panjang kata sandi, serta memastikan tidak ada field yang kosong. Apabila validasi ini tidak terpenuhi, pengguna akan diminta untuk memperbaiki data yang dimasukkan hingga sesuai dengan ketentuan yang berlaku.

Jika data *login* telah lolos validasi pada sisi *frontend*, permintaan *login* akan dikirimkan ke *backend* untuk diproses lebih lanjut. *Backend* kemudian mengambil data pengguna berdasarkan email yang dikirimkan pada *request*. Apabila data pengguna tidak ditemukan dalam basis data, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi kepada *frontend*.

Apabila data pengguna berhasil ditemukan, *backend* akan melakukan proses verifikasi kata sandi dengan mencocokkannya terhadap data yang tersimpan di

dalam basis data. Jika kata sandi tidak sesuai, sistem akan mengirimkan pesan kesalahan autentikasi ke frontend. Sebaliknya, jika proses verifikasi berhasil, *backend* akan mengirimkan respons berupa status *success* beserta data pengguna kepada *frontend*. Dengan alur ini, proses *login* dapat berjalan secara aman dan terkontrol sesuai dengan rancangan sistem.

C Flowchart Get All Color Code



Gambar 3.4. Flowchart *get all color code*

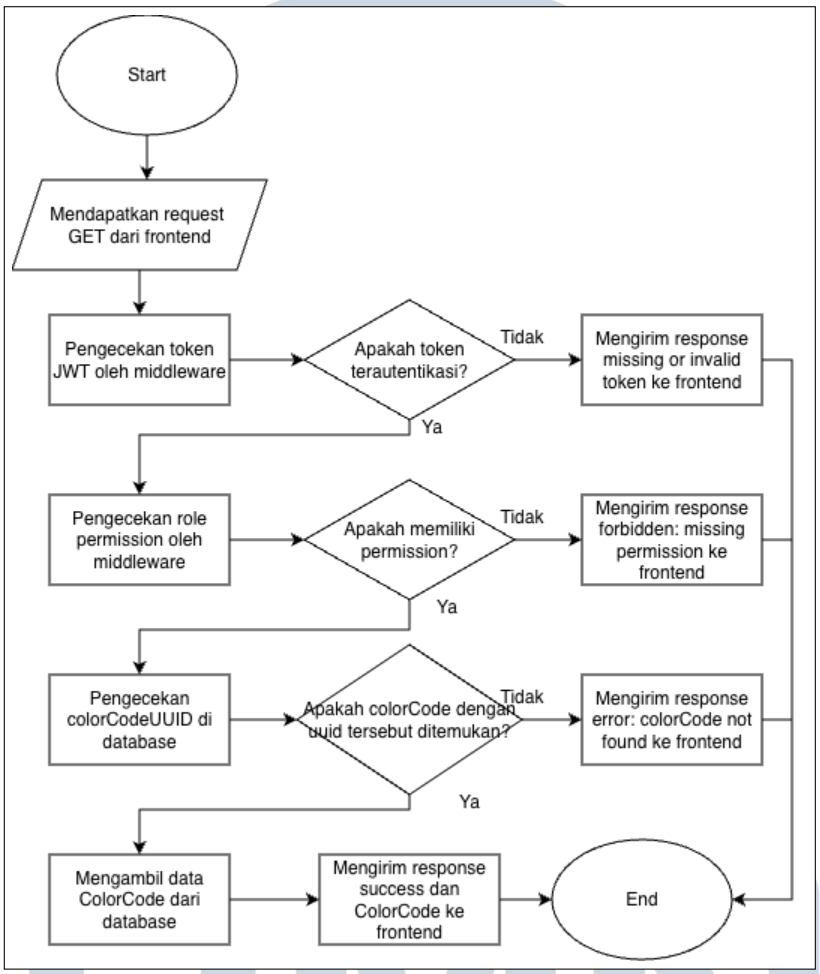
Flowchart proses pengambilan seluruh data *color code* menggambarkan alur pengambilan data kode area yang digunakan dalam sistem Ritra Logistics, sebagaimana ditunjukkan pada Gambar 3.4. Dalam sistem ini, *color code* digunakan sebagai penanda area tertentu yang direpresentasikan dalam bentuk

nama warna. Proses dimulai ketika *frontend* mengirimkan permintaan (request) bertipe GET untuk mengambil seluruh data *color code* yang tersedia pada sistem.

Setelah permintaan diterima, *backend* akan melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan bersama *request*. Apabila token tidak ditemukan atau tidak valid, sistem akan mengembalikan respons berupa pesan kesalahan yang menyatakan bahwa autentikasi gagal. Jika token dinyatakan valid, backend kemudian melanjutkan ke tahap otorisasi dengan memeriksa apakah pengguna memiliki hak akses atau *permission* yang sesuai untuk mengakses data color code. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan pesan kesalahan berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan mengambil seluruh data *color code* yang tersimpan di dalam basis data. Data tersebut kemudian dikirimkan kembali ke *frontend* dalam bentuk respons sukses. Dengan alur ini, sistem memastikan bahwa hanya pengguna yang terautentikasi dan memiliki izin yang sesuai yang dapat mengakses data *color code*, sehingga keamanan dan kontrol akses tetap terjaga sesuai dengan rancangan sistem.

D Flowchart Get Color Code By UUID



Gambar 3.5. Flowchart *get color code by uuid*

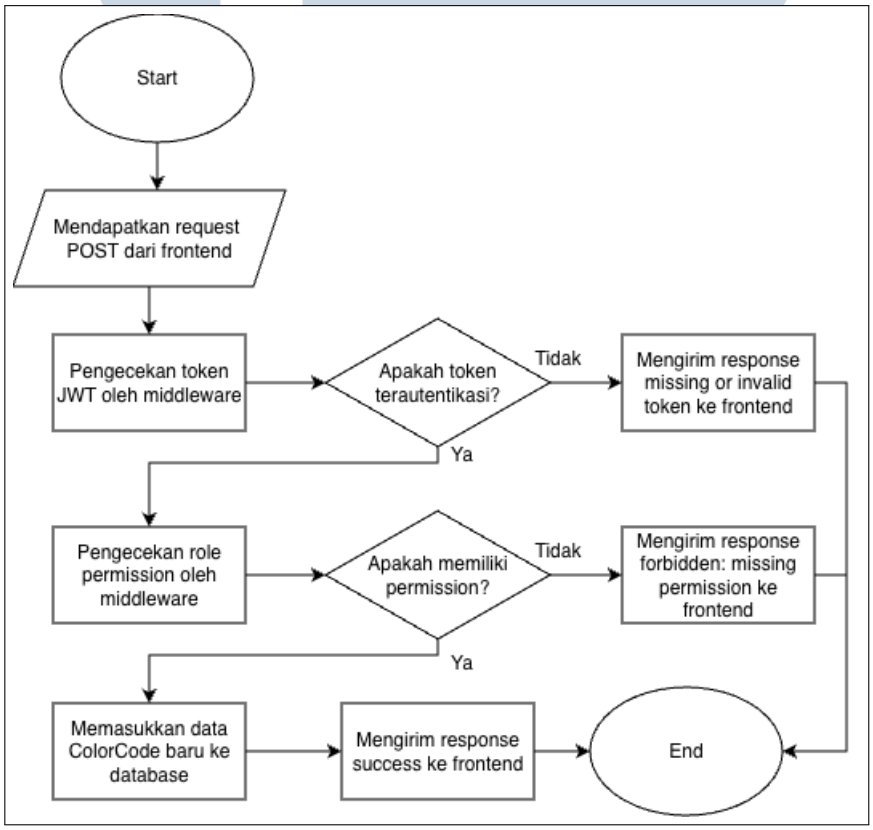
Flowchart proses pengambilan data *color code* berdasarkan UUID menggambarkan alur pengambilan data spesifik dari sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.5. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* dengan menyertakan UUID dari *color code* yang ingin diambil. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons kesalahan berupa *missing or invalid token*. Jika autentikasi berhasil, *backend* melanjutkan dengan proses otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai.

Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengirimkan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *color code* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan di dalam basis data, sistem akan mengembalikan respons berupa pesan *color code not found*. Namun, jika data ditemukan, *backend* akan mengambil data tersebut dan mengirimkan respons keberhasilan beserta detail *color code* yang diminta kepada *frontend*. Dengan alur ini, sistem memastikan pengambilan data dilakukan secara aman dan terkontrol sesuai dengan hak akses pengguna.

E Flowchart Create Color Code



Gambar 3.6. Flowchart *create color code*

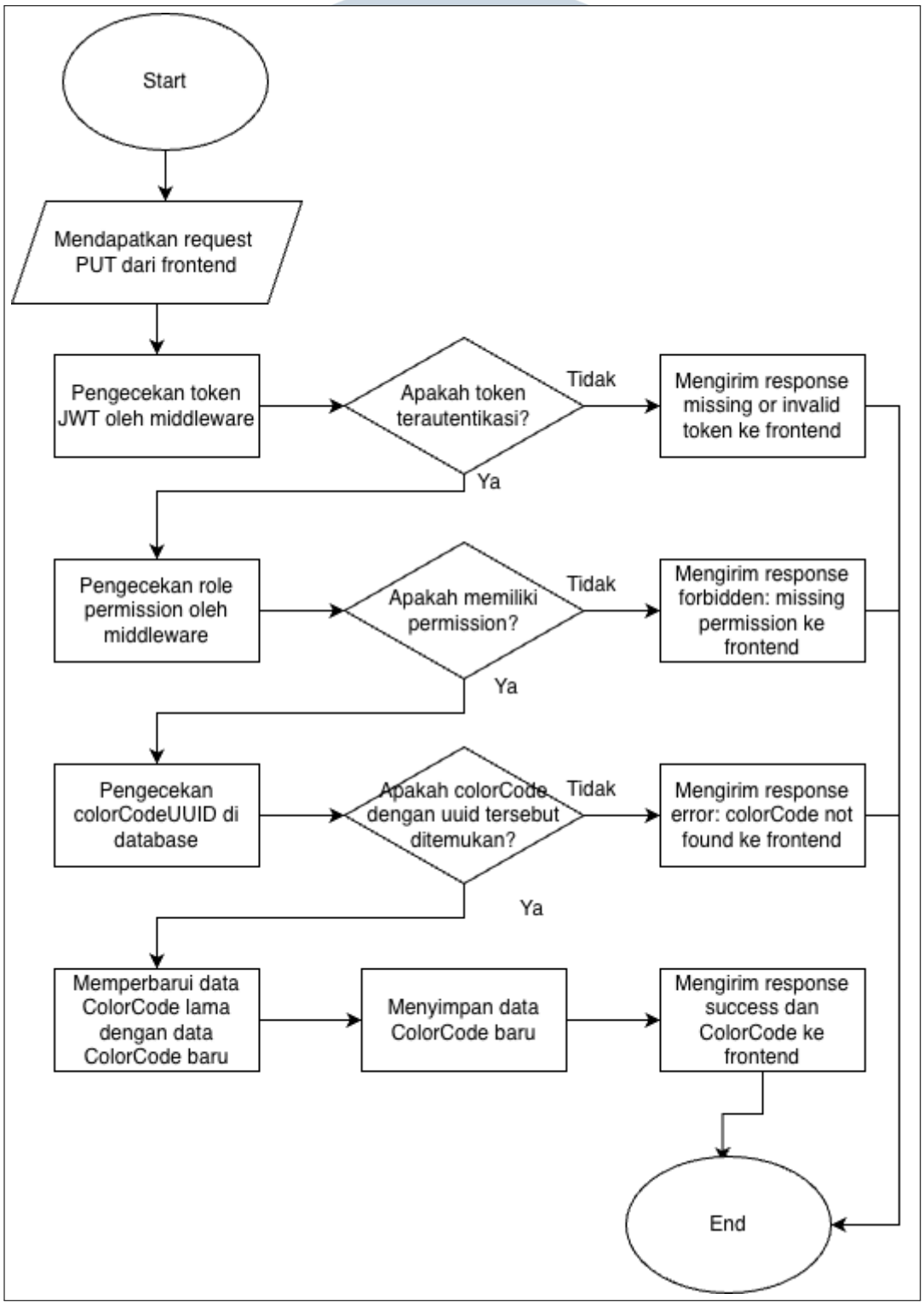
Flowchart proses pembuatan *color code* baru menggambarkan alur penambahan data kode area ke dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.6. Proses dimulai ketika *frontend* mengirimkan

permintaan (*request*) bertipe *POST* yang berisi data *color code* baru yang akan ditambahkan ke dalam sistem. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika proses autentikasi berhasil, *backend* akan melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melanjutkan dengan menyimpan data *color code* baru ke dalam basis data. Setelah proses penyimpanan berhasil dilakukan, sistem akan mengirimkan respons keberhasilan beserta data *color code* yang baru dibuat kepada *frontend*. Dengan alur ini, proses penambahan *color code* dapat dilakukan secara aman dan terkontrol sesuai dengan hak akses pengguna.

F Flowchart Update Color Code



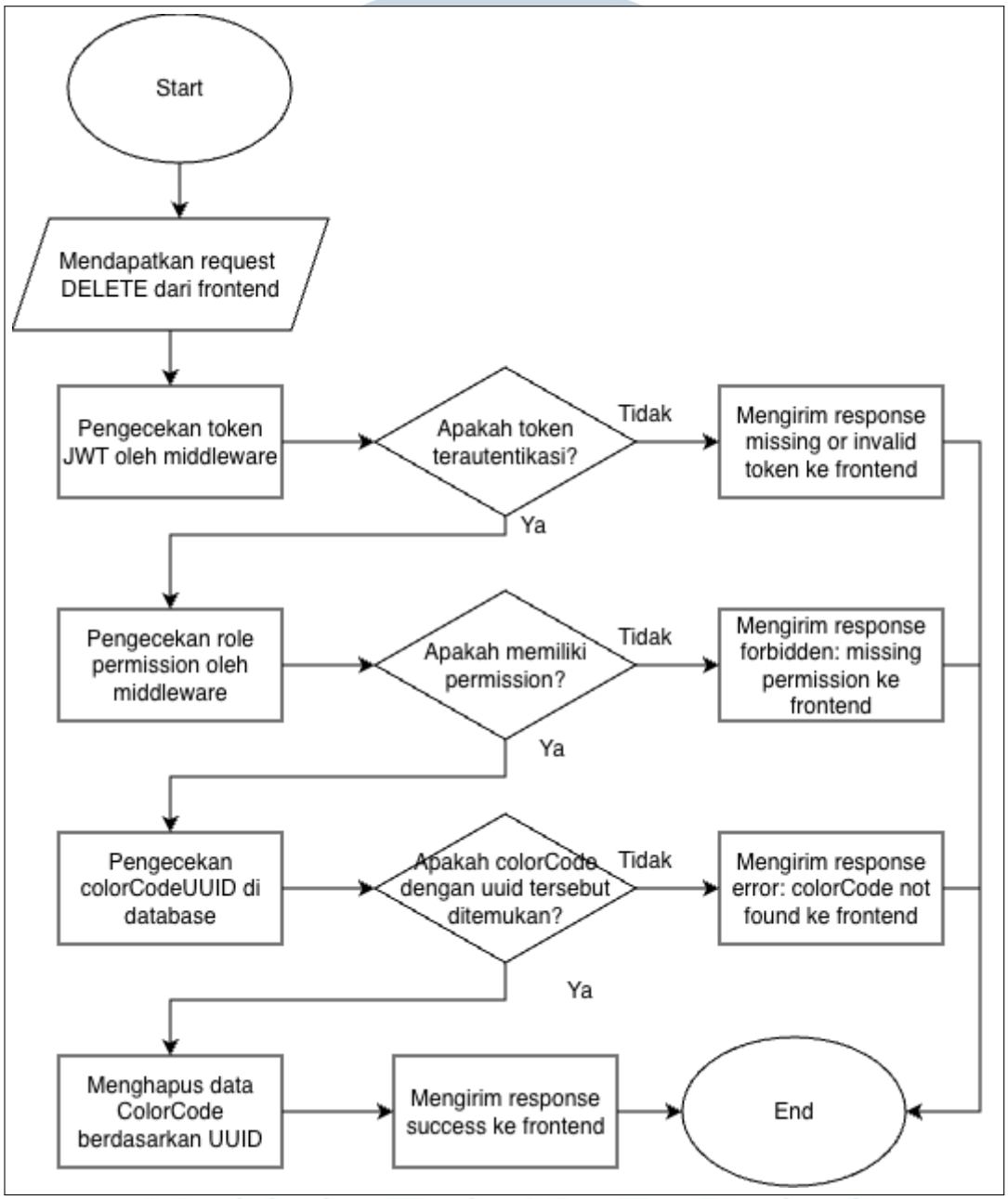
Gambar 3.7. Flowchart *update color code*

Flowchart proses pembaruan *color code* menggambarkan alur perubahan data *color code* yang telah ada pada sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.7. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *PUT* yang berisi data *color code* terbaru yang akan diperbarui. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* akan melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *color code* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan, sistem akan mengembalikan respons berupa pesan *color code not found*. Namun, jika data ditemukan, *backend* akan memperbarui data *color code* yang lama dengan data baru yang dikirimkan dari *frontend*, tanpa menambahkan baris data baru pada basis data. Setelah proses pembaruan berhasil dilakukan, sistem akan mengirimkan respons keberhasilan beserta data *color code* terbaru kepada *frontend*.

G Flowchart Delete Color Code



Gambar 3.8. Flowchart *delete color code*

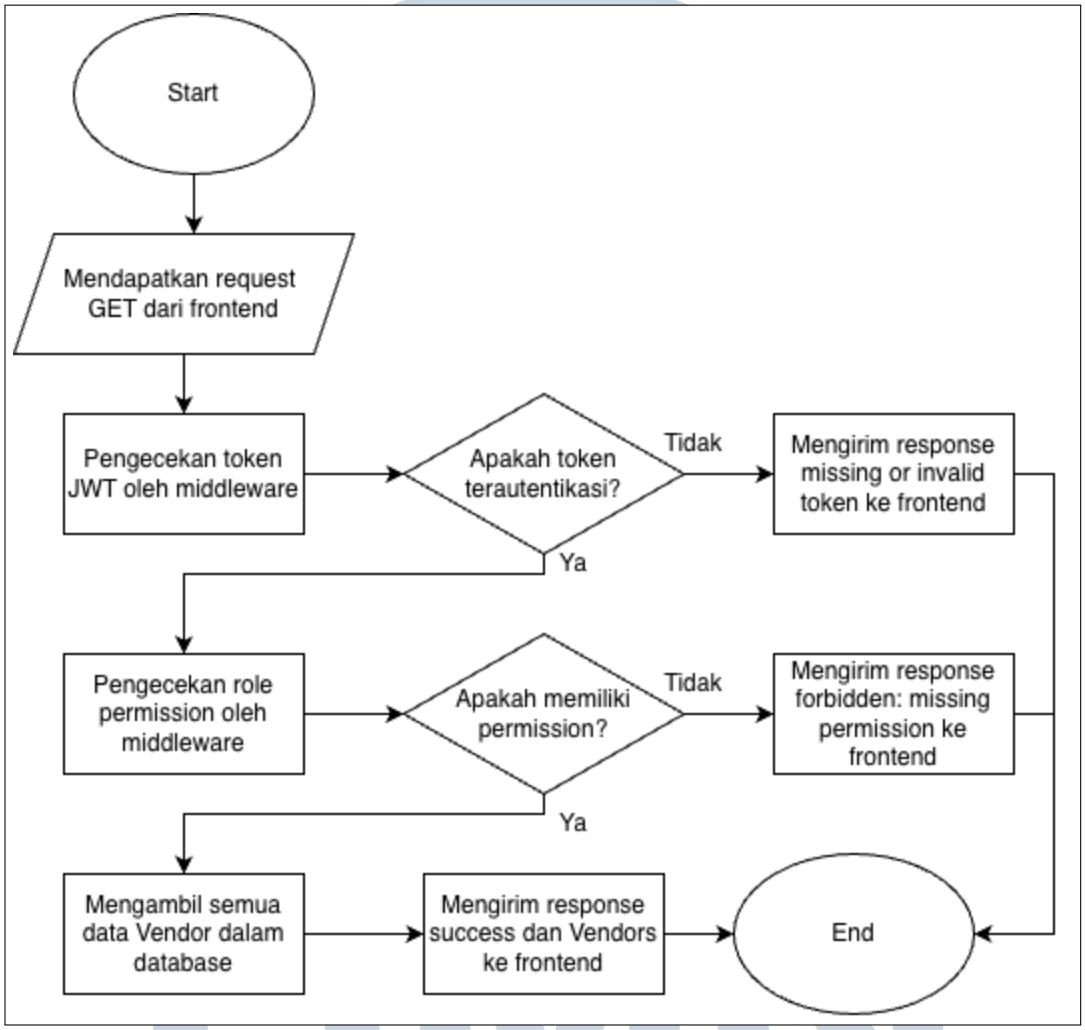
Flowchart proses penghapusan *color code* menggambarkan alur penghapusan data *color code* dari sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.8. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *DELETE* yang berisi UUID dari *color code* yang akan dihapus.

Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *color code* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan, sistem akan mengembalikan respons berupa pesan *color code not found*. Namun, jika data ditemukan, *backend* akan menghapus data *color code* tersebut dari basis data. Setelah proses penghapusan berhasil dilakukan, sistem akan mengirimkan respons keberhasilan kepada *frontend* sebagai penanda bahwa data telah berhasil dihapus.

H Flowchart Get All Vendor



Gambar 3.9. Flowchart *get all vendor*

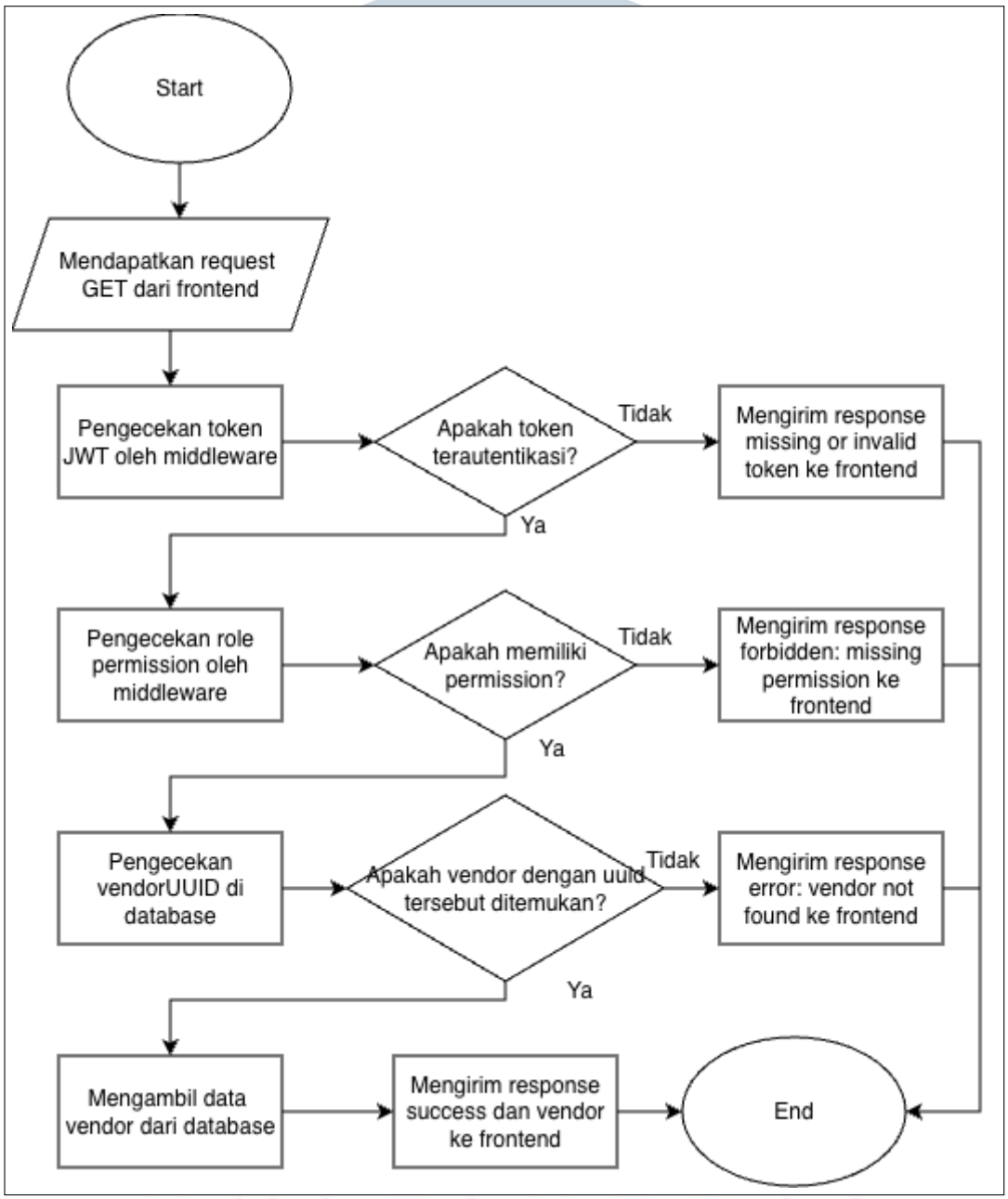
Flowchart proses pengambilan seluruh data *vendor* menggambarkan alur pengambilan data *vendor* yang tersimpan dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.9. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* untuk mengambil seluruh data *vendor* yang tersedia pada sistem. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk

memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan mengambil seluruh data *vendor* yang tersimpan di dalam basis data. Data tersebut kemudian dikirimkan kembali ke *frontend* dalam bentuk respons keberhasilan beserta daftar *vendor* yang tersedia. Dengan alur ini, sistem memastikan bahwa data *vendor* hanya dapat diakses oleh pengguna yang telah terautentikasi dan memiliki izin yang sesuai.

I Flowchart Get Vendor By UUID



Gambar 3.10. Flowchart *get vendor by uuid*

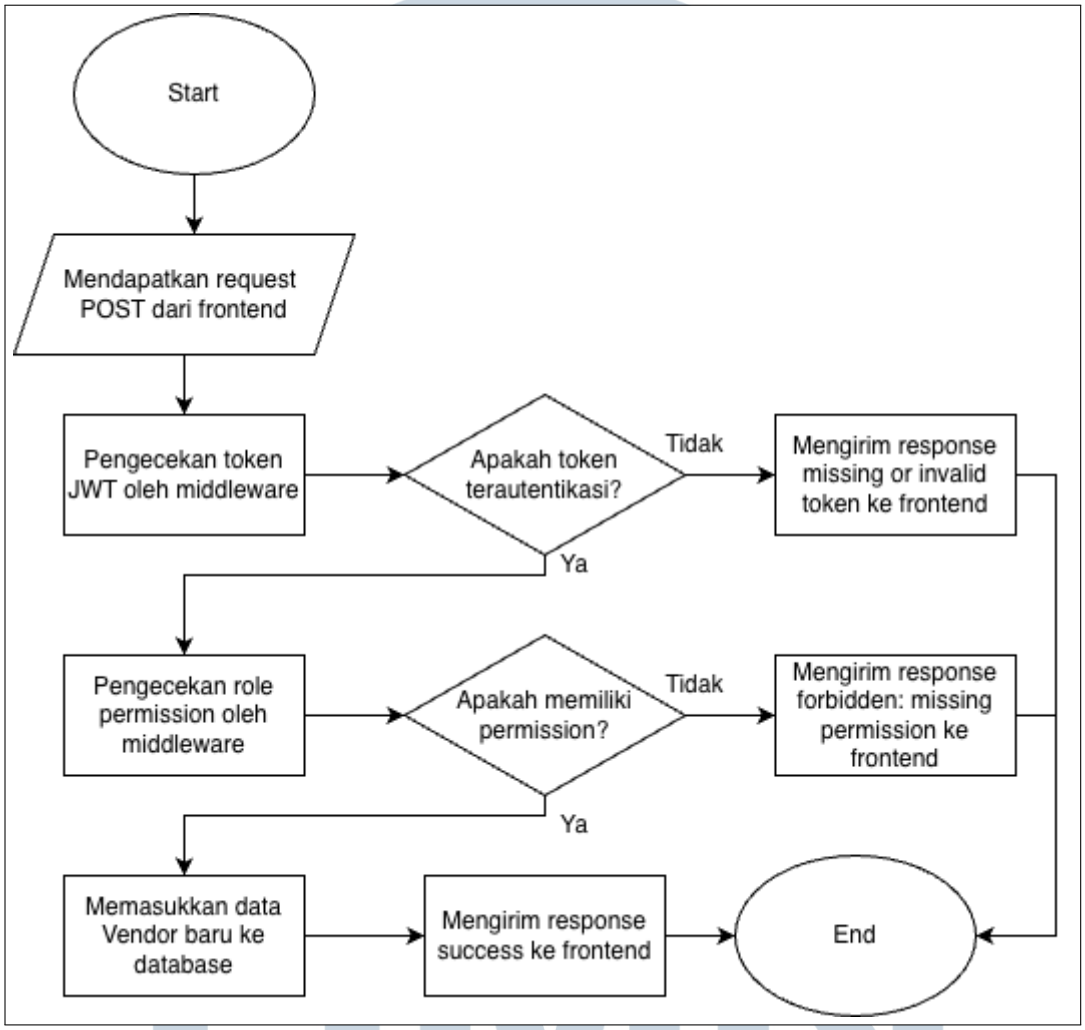
Flowchart proses pengambilan data *vendor* berdasarkan UUID menggambarkan alur pengambilan data *vendor* tertentu dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.10. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* yang berisi UUID dari

vendor yang ingin diambil. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *vendor* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan, sistem akan mengembalikan respons berupa pesan *vendor not found*. Namun, jika data ditemukan, *backend* akan mengambil data *vendor* tersebut dan mengirimkan respons keberhasilan beserta detail data *vendor* kepada *frontend*.

J Flowchart Create Vendor



Gambar 3.11. Flowchart *create vendor*

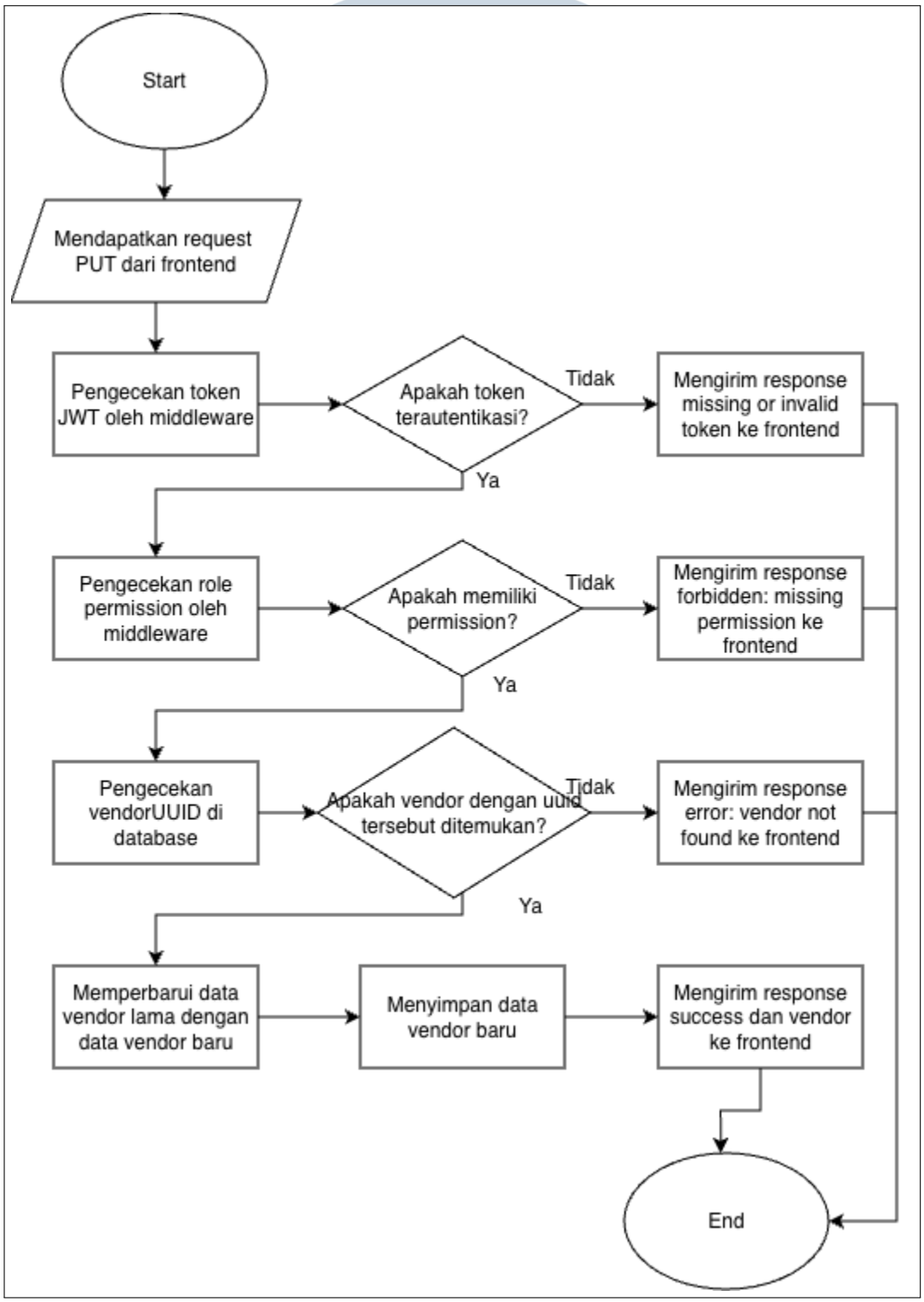
Flowchart proses pembuatan *vendor* baru menggambarkan alur penambahan data *vendor* ke dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.11. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *POST* yang berisi data *vendor* baru yang akan ditambahkan ke dalam sistem. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk

memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan menyimpan data *vendor* baru ke dalam basis data. Setelah proses penyimpanan berhasil dilakukan, sistem akan mengirimkan respons keberhasilan beserta data *vendor* yang baru ditambahkan kepada *frontend*. Dengan alur ini, proses penambahan data *vendor* dapat dilakukan secara aman dan terkontrol sesuai dengan kebijakan akses sistem.

K Flowchart Update Vendor



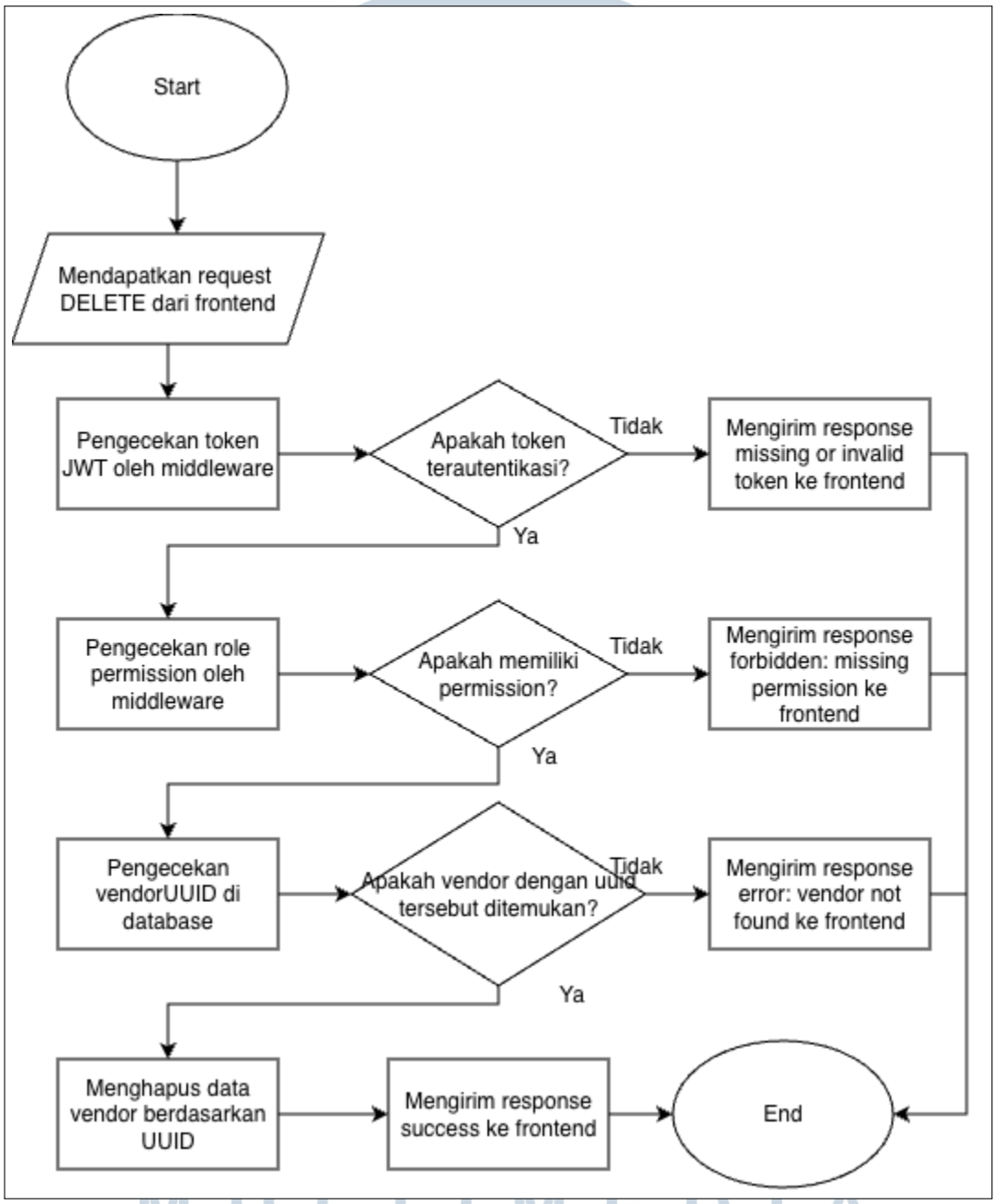
Gambar 3.12. Flowchart *update vendor*

Flowchart proses pembaruan data *vendor* menggambarkan alur perubahan data *vendor* yang telah tersimpan di dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.12. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *PUT* yang berisi data *vendor* terbaru yang akan digunakan untuk memperbarui data sebelumnya. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *vendor* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan, sistem akan mengembalikan respons berupa pesan *vendor not found*. Namun, jika data ditemukan, *backend* akan memperbarui data *vendor* lama dengan data yang baru, kemudian menyimpannya kembali ke dalam basis data. Setelah proses pembaruan berhasil, sistem akan mengirimkan respons keberhasilan beserta data *vendor* terbaru kepada *frontend*.

L Flowchart Delete Vendor



Gambar 3.13. Flowchart *delete vendor*

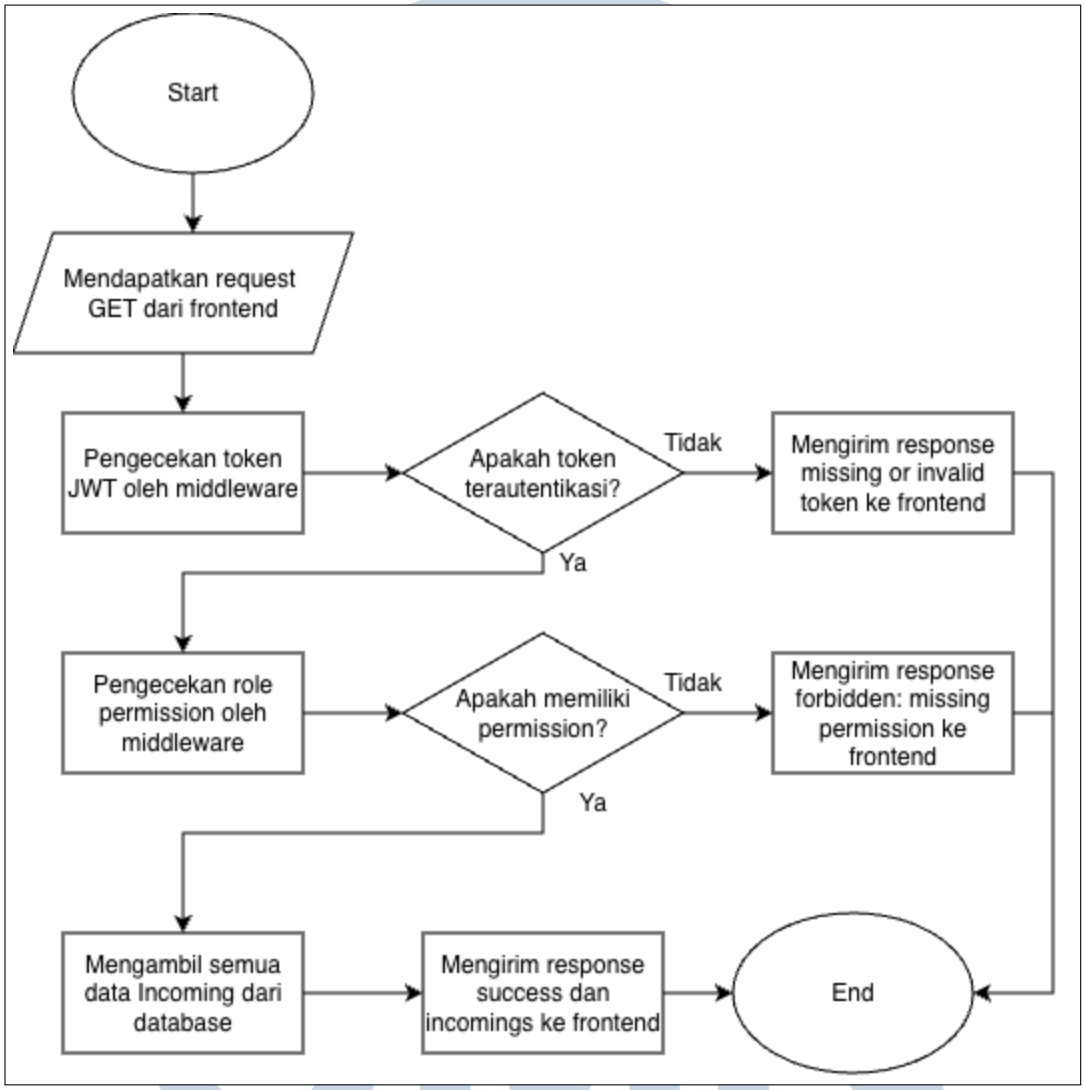
Flowchart proses penghapusan data *vendor* menggambarkan alur penghapusan data *vendor* dari sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.13. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *DELETE* yang berisi UUID dari *vendor* yang akan dihapus.

Permintaan tersebut kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *vendor* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan, sistem akan mengembalikan respons berupa pesan *vendor not found*. Namun, jika data ditemukan, *backend* akan menghapus data *vendor* tersebut dari basis data. Setelah proses penghapusan berhasil dilakukan, sistem akan mengirimkan respons keberhasilan kepada *frontend* sebagai tanda bahwa data telah berhasil dihapus.

M Flowchart Get All Incoming



Gambar 3.14. Flowchart *get all incoming*

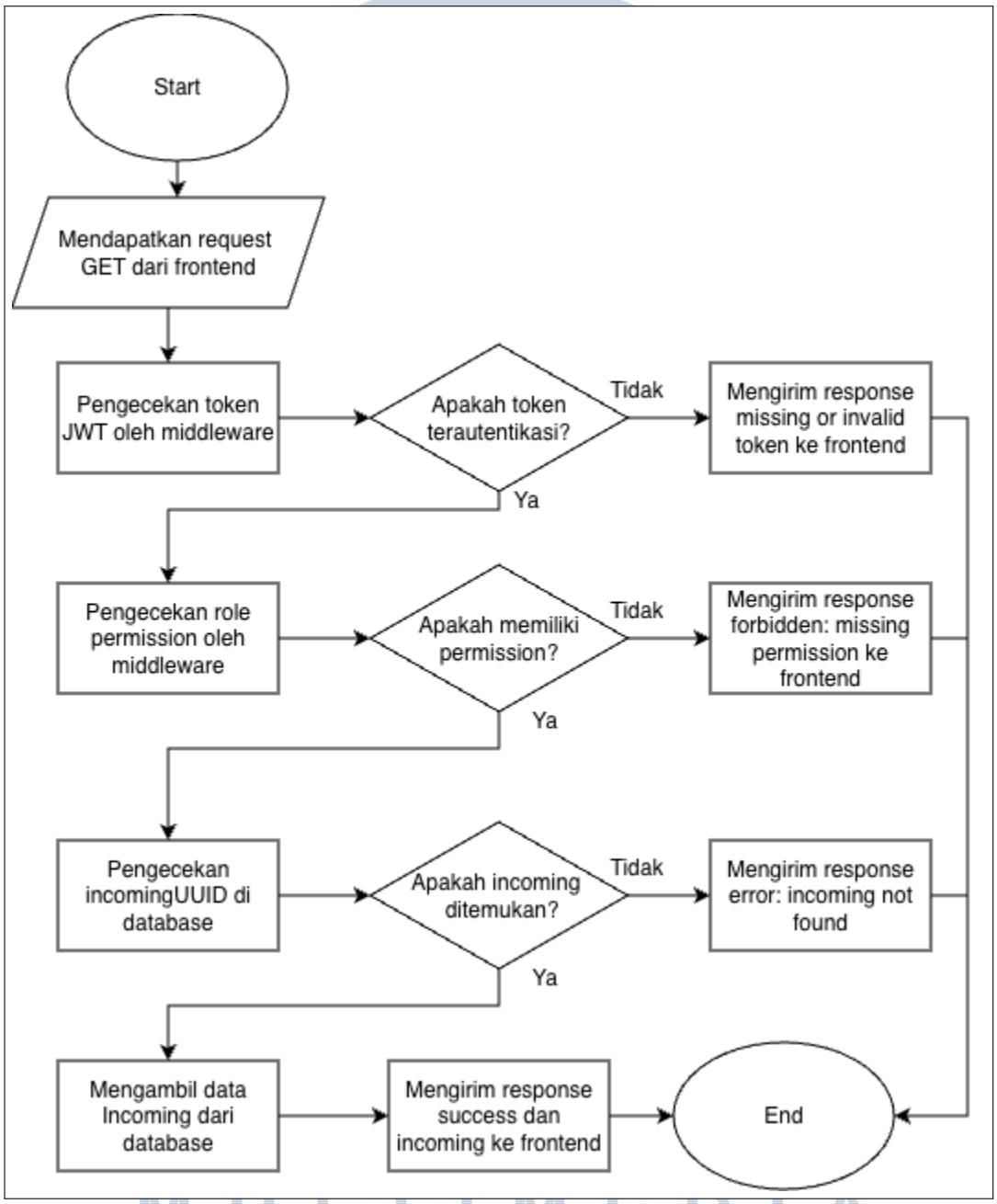
Flowchart proses pengambilan seluruh data *incoming* menggambarkan alur pengambilan data barang masuk yang tersimpan dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.14. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* untuk mengambil seluruh data *incoming* yang tersedia pada sistem. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan

autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan mengambil seluruh data *incoming* yang tersimpan di dalam basis data. Data tersebut kemudian dikirimkan kembali ke *frontend* dalam bentuk respons keberhasilan beserta daftar data *incoming* yang tersedia. Dengan alur ini, sistem memastikan bahwa data *incoming* hanya dapat diakses oleh pengguna yang telah terautentikasi dan memiliki izin yang sesuai.

N Flowchart Get Incoming By UUID



Gambar 3.15. Flowchart *get incoming by uuid*

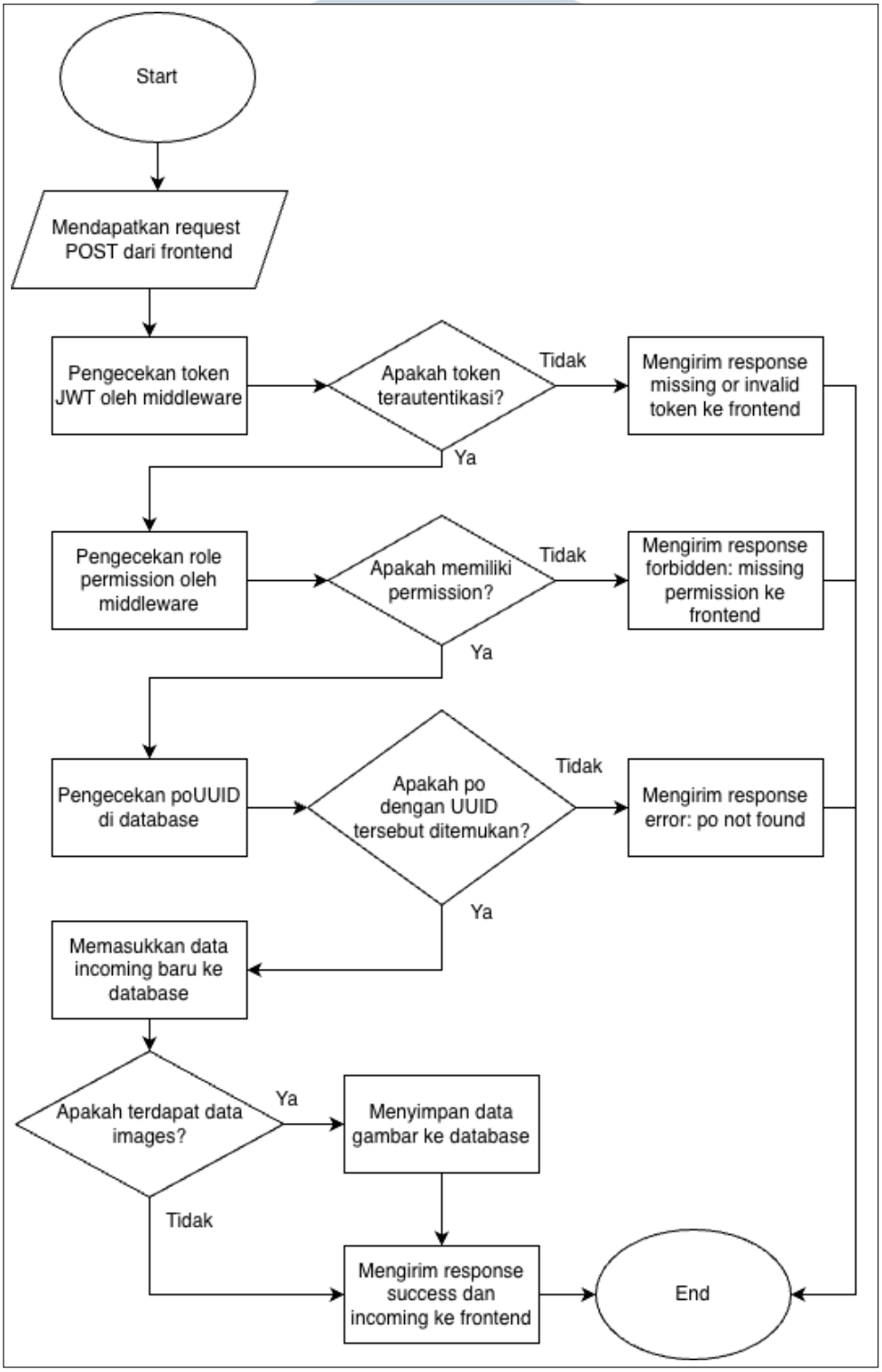
Flowchart proses pengambilan data *incoming* berdasarkan UUID menggambarkan alur pengambilan data barang masuk tertentu dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.15. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* yang berisi UUID dari

data *incoming* yang ingin diambil. Permintaan tersebut kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *incoming* berdasarkan UUID yang diterima dari *request*. Apabila data tidak ditemukan, sistem akan mengembalikan respons berupa pesan *incoming not found*. Namun, jika data ditemukan, *backend* akan mengambil data tersebut dan mengirimkan respons keberhasilan beserta detail data *incoming* kepada *frontend*. Dengan alur ini, sistem memastikan bahwa pengambilan data dilakukan secara aman dan sesuai dengan hak akses pengguna.

O Flowchart Create Incoming



Gambar 3.16. Flowchart *create incoming*

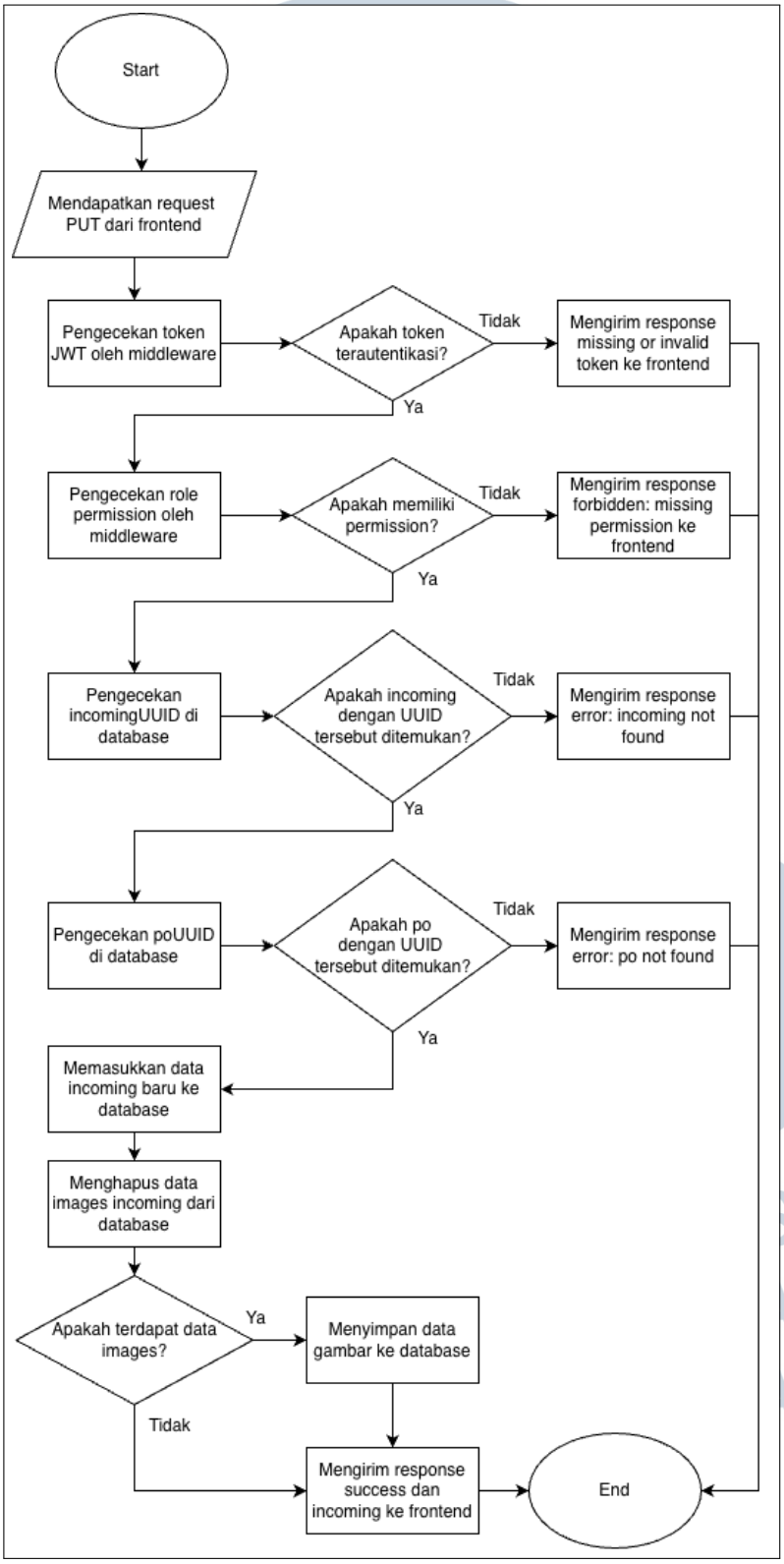
Flowchart proses pembuatan data *incoming* menggambarkan alur penambahan data barang masuk ke dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.16. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *POST* yang berisi data *incoming* baru. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pengecekan terhadap *purchase order* (*PO*) berdasarkan *PO UUID* yang dikirimkan pada *request*. Hal ini dilakukan karena setiap data *incoming* harus memiliki relasi dengan data *purchase order*. Apabila data *PO* tidak ditemukan, sistem akan mengembalikan respons berupa pesan *PO not found*. Jika data *PO* ditemukan, *backend* akan menyimpan data *incoming* baru ke dalam basis data.

Setelah data *incoming* berhasil disimpan, *backend* akan melakukan pengecekan terhadap *field images*. Apabila *field* tersebut tidak berisi data, sistem akan langsung mengirimkan respons keberhasilan beserta data *incoming* ke *frontend*. Namun, apabila *field images* tersedia, *backend* akan terlebih dahulu menyimpan data gambar tersebut ke dalam basis data, kemudian mengirimkan respons keberhasilan beserta data *incoming* yang telah tersimpan. Dengan alur ini, proses pencatatan data barang masuk dapat berjalan secara fleksibel dan terintegrasi sesuai kebutuhan sistem.

P Flowchart Update Incoming



Gambar 3.17. Flowchart *update incoming*

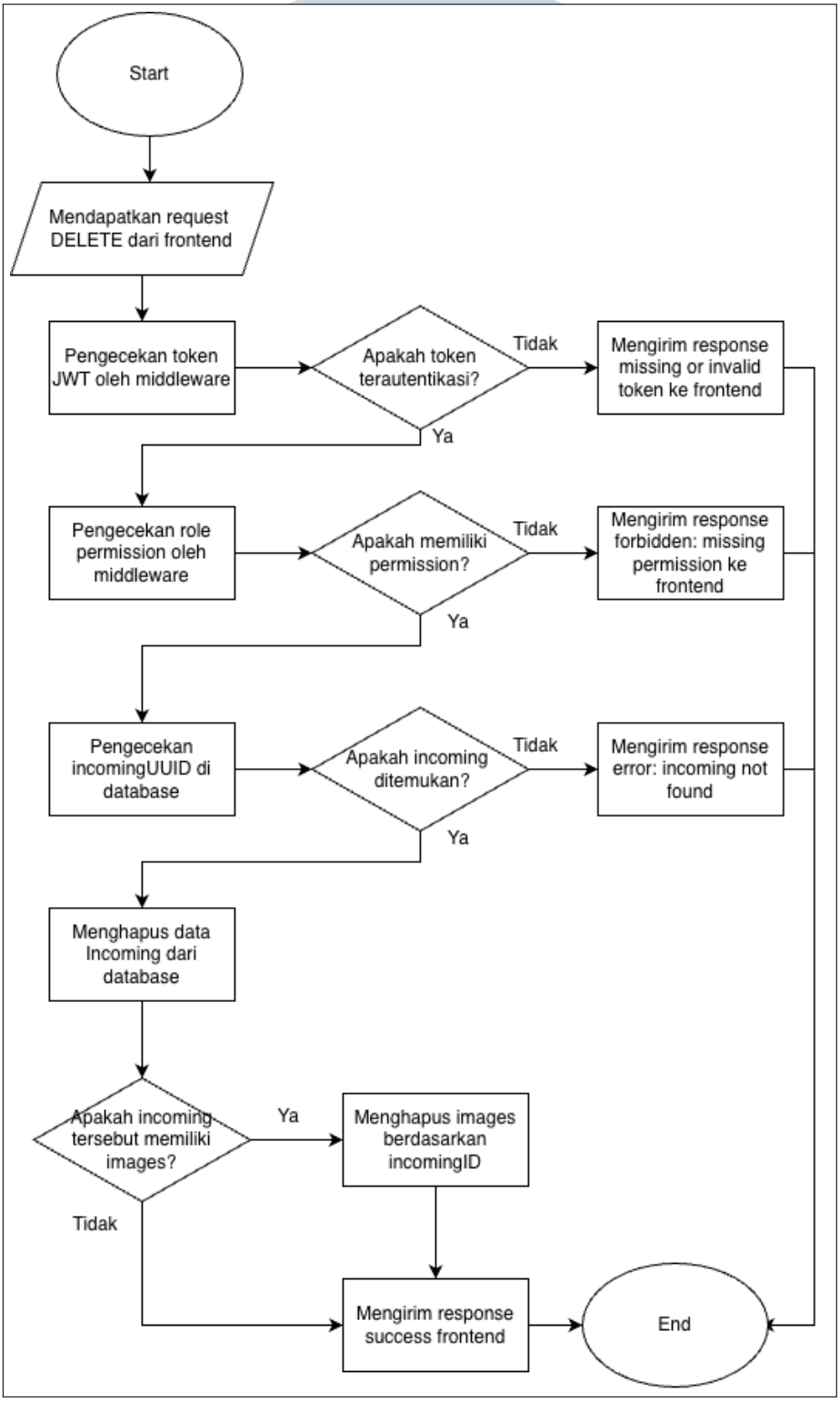
Flowchart proses pembaruan data *incoming* menggambarkan alur perubahan data barang masuk yang telah tersimpan dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.17. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *PUT* yang berisi data *incoming* terbaru untuk dilakukan pembaruan. Permintaan ini kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *incoming* berdasarkan UUID yang diterima dari *request*. Apabila data *incoming* tidak ditemukan, sistem akan mengembalikan respons berupa pesan *incoming not found*. Selanjutnya, *backend* akan melakukan pengecekan terhadap *purchase order* (*PO*) berdasarkan *PO UUID* yang dikirimkan. Apabila data *PO* tidak ditemukan, sistem akan mengembalikan respons berupa pesan *PO not found*.

Jika seluruh data valid, *backend* akan memperbarui data *incoming* dengan informasi terbaru yang diterima dari *request*. Setelah itu, *backend* akan menghapus data *images* lama yang sebelumnya terasosiasi dengan *incoming* tersebut. Selanjutnya, sistem akan melakukan pengecekan terhadap *field images*. Apabila tidak terdapat data gambar, sistem akan langsung mengirimkan respons keberhasilan beserta data *incoming* terbaru kepada *frontend*. Namun, apabila *field images* tersedia, *backend* akan menyimpan data gambar tersebut ke dalam basis data dan kemudian mengirimkan respons keberhasilan beserta data *incoming* terbaru ke *frontend*.

Q Flowchart Delete Incoming



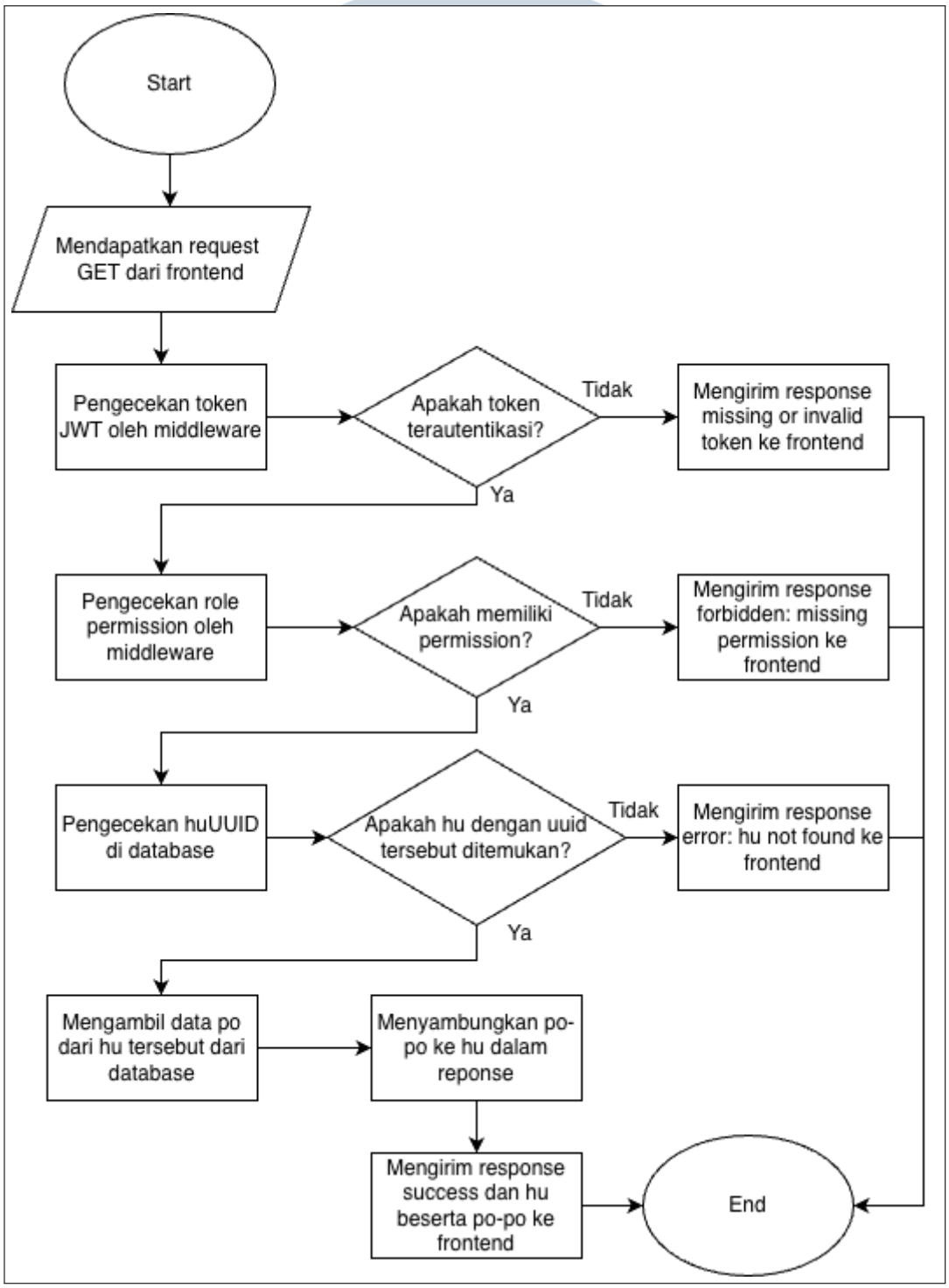
Gambar 3.18. Flowchart *delete incoming*

Flowchart proses penghapusan data *incoming* menggambarkan alur penghapusan data barang masuk dari sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.18. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *DELETE* yang berisi UUID dari data *incoming* yang akan dihapus. Permintaan tersebut kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *incoming* berdasarkan UUID yang diterima dari *request*. Apabila data *incoming* tidak ditemukan, sistem akan mengembalikan respons berupa pesan *incoming not found*. Namun, jika data ditemukan, *backend* akan menghapus data *incoming* tersebut dari basis data. Selanjutnya, sistem juga akan menghapus data *images* yang terasosiasi dengan *incoming* apabila data tersebut tersedia. Setelah seluruh proses penghapusan berhasil dilakukan, sistem akan mengirimkan respons keberhasilan kepada *frontend* sebagai tanda bahwa data telah berhasil dihapus.

R Flowchart Get HU By UUID



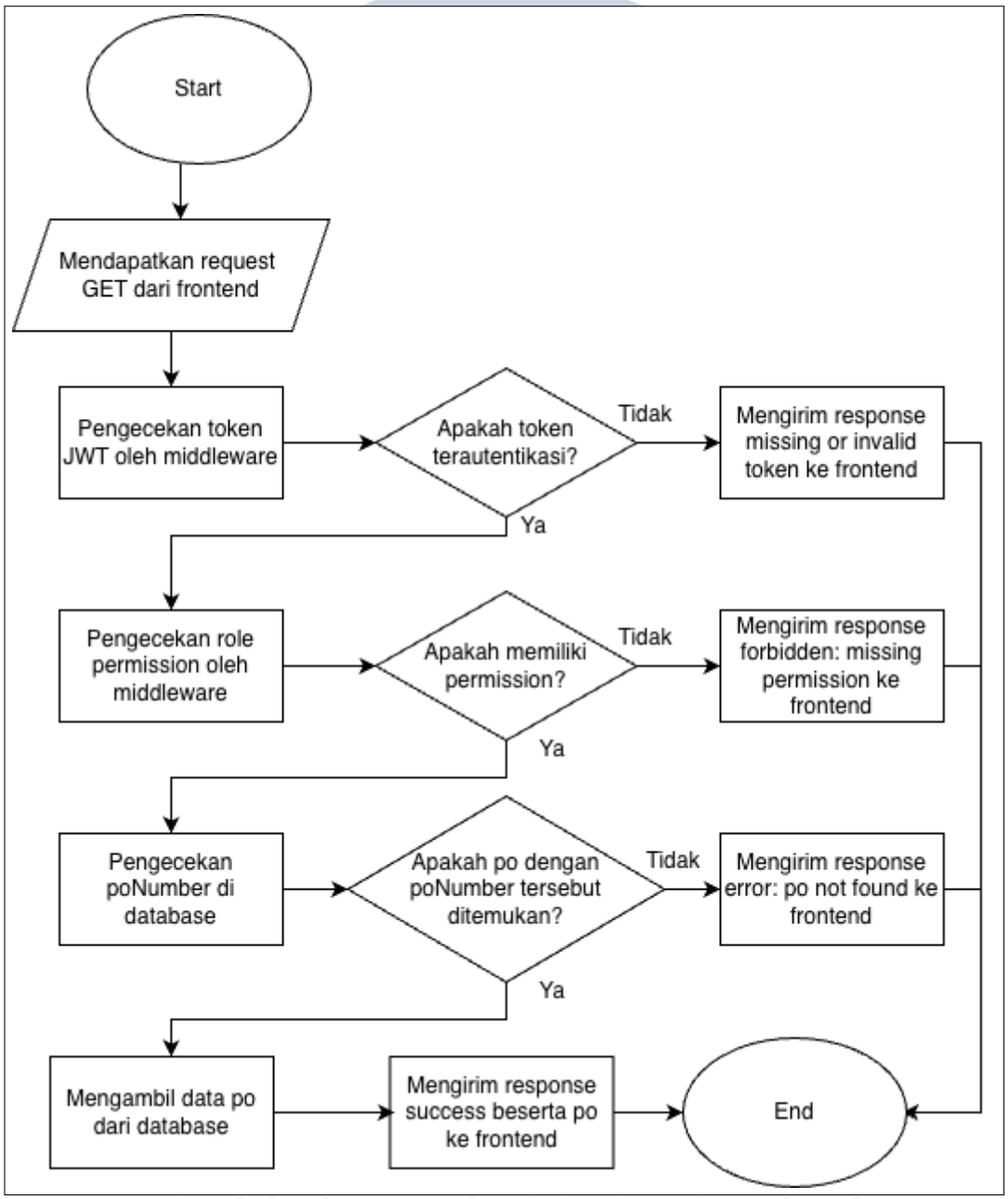
Gambar 3.19. Flowchart *get HU by uuid*

Flowchart proses pengambilan data *handling unit* (*HU*) berdasarkan UUID menggambarkan alur pengambilan informasi *HU* beserta data *purchase order* (*PO*) yang terkait di dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.19. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* yang berisi UUID dari *HU* yang ingin diambil. Permintaan tersebut kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *handling unit* berdasarkan UUID yang diterima dari *request*. Apabila data *HU* tidak ditemukan, sistem akan mengembalikan respons berupa pesan *HU not found*. Namun, jika data *HU* ditemukan, *backend* akan mengambil seluruh data *purchase order* (*PO*) yang terhubung dengan *HU* tersebut. Seluruh data tersebut kemudian digabungkan menjadi satu struktur respons yang berisi informasi *HU* beserta daftar *PO* yang terkait. Setelah itu, sistem mengirimkan respons keberhasilan beserta data *HU* dan *PO* kepada *frontend*.

S Flowchart Get PO by PO Number



Gambar 3.20. Flowchart *get PO by PO number*

Flowchart proses pengambilan data *purchase order* (*PO*) berdasarkan nomor *PO* menggambarkan alur pengambilan data *PO* tertentu dalam sistem *Ritra Logistics*, sebagaimana ditunjukkan pada Gambar 3.20. Proses dimulai ketika *frontend* mengirimkan permintaan (*request*) bertipe *GET* yang berisi nomor *PO*

yang ingin dicari. Permintaan tersebut kemudian diterima oleh *backend* untuk diproses lebih lanjut.

Setelah menerima *request*, *backend* melakukan proses autentikasi dengan memeriksa token JWT yang dikirimkan. Apabila token tidak valid atau tidak ditemukan, sistem akan mengembalikan respons berupa pesan kesalahan autentikasi. Jika autentikasi berhasil, *backend* melanjutkan ke tahap otorisasi untuk memastikan bahwa pengguna memiliki hak akses yang sesuai. Apabila pengguna tidak memiliki izin yang diperlukan, sistem akan mengembalikan respons berupa *missing permission* kepada *frontend*.

Jika proses autentikasi dan otorisasi berhasil, *backend* akan melakukan pencarian data *purchase order* berdasarkan nomor *PO* yang diterima dari *request*. Apabila data *PO* tidak ditemukan, sistem akan mengembalikan respons berupa pesan *PO not found*. Namun, jika data *PO* ditemukan, *backend* akan mengambil data tersebut dan mengirimkan respons keberhasilan beserta informasi *PO* kepada *frontend*. Dengan alur ini, proses pencarian data *PO* dapat dilakukan secara aman dan terkontrol sesuai dengan ketentuan sistem.

3.3.3 Pembuatan *Model* dan *Data Transfer Object* (DTO)

Model dan *Data Transfer Object* (DTO) merupakan komponen penting dalam perancangan backend sistem Ritra Logistics. Model digunakan sebagai representasi struktur tabel pada basis data, sedangkan DTO berfungsi sebagai perantara dalam pertukaran data antara client dan server. Penggunaan DTO bertujuan untuk memastikan data yang dikirim dan diterima sesuai dengan kebutuhan sistem, sekaligus menjaga keamanan dan konsistensi data yang diproses.

Setiap model dalam sistem memiliki sejumlah *field* utama yang secara otomatis dihasilkan oleh *backend* tanpa memerlukan *input* dari sisi *frontend*. *Field* tersebut meliputi ID, UUID, CreatedAt, CreatedBy, UpdatedAt, dan UpdatedBy. Keberadaan *field-field* ini berperan penting dalam proses identifikasi data, pelacakan perubahan, serta pengelolaan relasi antar entitas dalam sistem.

A Model dan DTO User

Dalam model user, field yang disimpan ke database adalah nama user, email, password, dan role. Cuplikan kode model user dapat dilihat pada 3.1.

```
1 package models
2
```

```
import (
  "time"
)

type User struct {
  ID        uint    `gorm:"primaryKey;autoIncrement"`
  UUID      string  `gorm:"type:char(36);uniqueIndex"`
  Name      string  `gorm:"size:50;not null"`
  Email     string  `gorm:"size:100;uniqueIndex;not null"`
  Password  string  `gorm:"size:255;not null"`
  RoleID    *uint
  Role      *Role   `gorm:"foreignKey:RoleID;constraint:OnUpdate:
    CASCADE,OnDelete:SET NULL;"`

  CreatedAt time.Time `gorm:"autoCreateTime"`
  CreatedBy *uint
  UpdatedAt time.Time `gorm:"autoUpdateTime"`
  UpdatedBy *uint
}
```

Kode 3.1: Cuplikan kode model User

Untuk kebutuhan komunikasi data antara frontend dan backend, digunakan Data Transfer Object (DTO) yang dibedakan berdasarkan jenis permintaan. Pada proses autentikasi, terdapat dua jenis DTO request, yaitu *RegisterRequest* dan *LoginRequest*. DTO *RegisterRequest* digunakan saat proses pendaftaran pengguna baru dan memerlukan data nama, email, serta kata sandi. Sementara itu, *LoginRequest* hanya membutuhkan email dan kata sandi. Cuplikan kode DTO request dapat dilihat pada Kode 3.2.

```
package requests

type RegisterRequest struct {
    Name string json:"name" validate:"required,min=3,max=50"
    Email string json:"email" validate:"required,email"
    Password string json:"password" validate:"required,min=6"
}

type LoginRequest struct {
    Email string json:"email" validate:"required,email"
    Password string json:"password" validate:"required"
}
```

Kode 3.2: Cuplikan kode DTO request user

Selain request, sistem juga memiliki DTO response yang digunakan untuk mengirimkan data pengguna kembali ke frontend setelah proses autentikasi berhasil. DTO ini hanya memuat informasi yang diperlukan oleh sisi klien, yaitu UUID, nama, dan email pengguna. Cuplikan kode DTO response dapat dilihat pada Kode 3.3.

```
package responses

type AuthResponse struct {
    UUID string json:"uuid"
    Name string json:"name"
    Email string json:"email"
}
```

Kode 3.3: Cuplikan kode DTO response user

B Model dan DTO ColorCode

Model Color Code digunakan untuk merepresentasikan data kode area yang digunakan dalam sistem Ritra Logistics. Model ini menyimpan informasi terkait identitas warna, area, serta atribut tambahan yang berkaitan dengan lokasi dan klasifikasi barang. Selain itu, model ini juga memiliki atribut audit seperti waktu pembuatan dan pembaruan data yang dikelola secara otomatis oleh sistem. Cuplikan kode model Color Code dapat dilihat pada Kode 3.4.

```
package models

import (
"time"

"github.com/google/uuid"
"gorm.io/gorm"


)

type ColorCode struct {
    ID uint json:"id" gorm:"primaryKey;autoIncrement"
    UUID string json:"uuid" gorm:"type:char(36);uniqueIndex;not
    null"
    Name string json:"name" gorm:"type:varchar(100);not null"
    Code string json:"code" gorm:"type:varchar(50);not null"
    Highland bool json:"highland" gorm:"not null;default:false"
```

```
    Description string json:"description" gorm:"type:text"
    Bloc string json:"bloc" gorm:"type:varchar(100)"
    CreatedAt time.Time json:"created_at" gorm:"autoCreateTime"
    CreatedBy *uint json:"created_by"
    UpdatedAt time.Time json:"updated_at" gorm:"autoUpdateTime"
    UpdatedBy *uint json:"updated_by"
    DeletedAt gorm.DeletedAt json:"-" gorm:"index"
}
```

Kode 3.4: Cuplikan kode model Color Code

Untuk kebutuhan komunikasi data antara frontend dan backend, digunakan Data Transfer Object (DTO) sebagai perantara pengiriman data. DTO request digunakan saat proses pembuatan atau pembaruan data color code. Struktur DTO ini hanya memuat field yang diperlukan dari sisi klien, sehingga dapat menjaga validasi dan keamanan data. Cuplikan kode DTO request dapat dilihat pada Kode 3.5.

```
package requests

type ColorCodeRequest struct {
    ColorCode string json:"color_code" validate:"required"
    Area string json:"area" validate:"required"
    Highland bool json:"highland"
    Description string json:"description"
    Sloc string json:"sloc"
}
```

Kode 3.5: Cuplikan kode DTO request Color Code

Selain DTO request, sistem juga menggunakan DTO response untuk mengirimkan data color code kembali ke frontend. DTO ini berisi informasi utama yang diperlukan untuk ditampilkan pada sisi klien, tanpa menyertakan atribut internal yang tidak diperlukan. Cuplikan kode DTO response dapat dilihat pada Kode 3.6.

```
package responses

type ColorCodeResponse struct {
    UUID string json:"uuid"
    ColorCode string json:"color_code"
    Area string json:"area"
    Highland bool json:"highland"
    Description string json:"description"
    Sloc string json:"sloc"
```

```
}
```

Kode 3.6: Cuplikan kode DTO response Color Code

C Model dan DTO Vendor

Model Vendor digunakan untuk merepresentasikan data pemasok atau pihak penyedia barang dalam sistem Ritra Logistics. Model ini menyimpan informasi identitas vendor seperti nama, alamat, nomor telepon, nomor faks, serta alamat email. Selain itu, model Vendor juga memiliki atribut pendukung berupa informasi waktu pembuatan dan pembaruan data yang dikelola secara otomatis oleh sistem. Cuplikan kode model Vendor dapat dilihat pada Kode 3.7.

```
package models

import (
"time"
"gorm.io/gorm"
)

type Vendor struct {
    ID uint gorm:"primaryKey" json:"id"
    UUID string gorm:"type:char(36);uniqueIndex" json:"uuid"
    Name string gorm:"type:varchar(255);not null" json:"name"
    Address string gorm:"type:text" json:"address"
    Phone string gorm:"type:varchar(50)" json:"phone"
    FaxNumber string gorm:"type:varchar(50)" json:"fax_number"
    Email string gorm:"type:varchar(255)" json:"email"

    CreatedAt time.Time      `json:"created_at" gorm:"autoCreateTime"`
    CreatedBy uint           `json:"created_by" gorm:"column:created_by"`
    UpdatedAt time.Time      `json:"updated_at" gorm:"autoUpdateTime"`
    UpdatedBy uint           `json:"updated_by" gorm:"column:updated_by"`

    DeletedAt gorm.DeletedAt `json:"-" gorm:"index"`
}
```

Kode 3.7: Cuplikan kode model Vendor

Untuk kebutuhan pertukaran data antara frontend dan backend, digunakan Data Transfer Object (DTO) yang berfungsi membatasi serta memvalidasi data yang diterima dan dikirimkan oleh sistem. DTO request digunakan pada proses pembuatan maupun pembaruan data vendor, sedangkan DTO response digunakan untuk mengirimkan informasi vendor yang telah diproses kepada frontend. Cuplikan kode DTO request dapat dilihat pada Kode 3.8.

```
package requests

type VendorRequest struct {
    Name string json:"name" validate:"required,min=3,max=255"
    Address string json:"address" validate:"required"
    Phone string json:"phone" validate:"required"
    FaxNumber string json:"fax_number"
    Email string json:"email" validate:"required,email"
}
```

Kode 3.8: Cuplikan kode DTO request Vendor

Selain DTO request, sistem juga menggunakan DTO response untuk mengirimkan data vendor ke sisi frontend setelah proses berhasil dilakukan. DTO ini hanya memuat informasi yang relevan untuk ditampilkan, tanpa menyertakan atribut internal sistem. Cuplikan kode DTO response dapat dilihat pada Kode 3.9.

```
package responses

type VendorResponse struct {
    UUID string json:"uuid"
    Name string json:"name"
    Address string json:"address"
    Phone string json:"phone"
    FaxNumber string json:"fax_number"
    Email string json:"email"
}
```

Kode 3.9: Cuplikan kode DTO response Vendor

D Model dan DTO Incoming

Model Incoming digunakan untuk merepresentasikan data barang masuk (incoming goods) dalam sistem Ritra Logistics. Data ini merekam informasi penerimaan barang yang berkaitan dengan Purchase Order (PO), termasuk status pemeriksaan, catatan tambahan, serta dokumentasi berupa gambar. Model ini juga

memiliki relasi dengan beberapa entitas lain seperti PO, User, dan IncomingImage. Cuplikan kode model Incoming dapat dilihat pada Kode 3.10.

```
package models

import (
"time"
"github.com/google/uuid"
"gorm.io/gorm"
)

type Incoming struct {
    ID uint json:"id" gorm:"primaryKey;autoIncrement"
    UUID string json:"uuid" gorm:"type:char(36);uniqueIndex;not null"
    POID *uint json:"po_id"
    CheckBy *uint json:"check_by"
    Status string json:"status" gorm:"type:varchar(10);not null;default:'pass'"
    Notes *string json:"notes" gorm:"type:text"

    PO PO 'json:"po" gorm:"foreignKey:POID;constraint:OnUpdate:CASCADE,OnDelete:SET NULL;"'
    IncomingImages []IncomingImage 'json:"incoming_images" gorm:"foreignKey:IncomingID;constraint:OnUpdate:CASCADE,OnDelete:CASCADE;"'
    CheckByUser *User 'json:"check_by_user,omitempty" gorm:"foreignKey:CheckBy;references:ID;constraint:OnUpdate:CASCADE,OnDelete:SET NULL;"'

    CreatedAt time.Time 'json:"created_at" gorm:"autoCreateTime"'
    CreatedBy *uint 'json:"created_by"'
    UpdatedAt time.Time 'json:"updated_at" gorm:"autoUpdateTime"'
    UpdatedBy *uint 'json:"updated_by"'
    DeletedAt gorm.DeletedAt 'json:"-" gorm:"index"'
}
```

Kode 3.10: Cuplikan kode model Incoming

Untuk menerima data dari frontend, digunakan DTO request yang berfungsi sebagai validasi awal terhadap data yang dikirimkan. DTO ini mencakup UUID dari Purchase Order, catatan tambahan, serta daftar gambar yang berkaitan dengan

proses incoming. Cuplikan kode DTO request dapat dilihat pada Kode 3.11.

```
package requests

type IncomingRequest struct {
    POUUID string json:"po_uuid" validate:"required,uuid4"
    Notes *string json:"notes"
    Images []IncomingImageRequest json:"images"
}
```

Kode 3.11: Cuplikan kode DTO request Incoming

Sebagai respons terhadap permintaan dari frontend, sistem mengembalikan data dalam bentuk DTO response. DTO ini berisi informasi utama incoming beserta relasi terkait seperti data PO, pemeriksa (checker), dan daftar gambar jika tersedia. Cuplikan kode DTO response dapat dilihat pada Kode 3.12.

```
package responses

import "time"

type IncomingResponse struct {
    UUID string json:"uuid"
    POUUID string json:"po_uuid"
    PO *POResponse json:"po,omitempty"
    CheckBy *CheckByResponse json:"check_by,omitempty"
    Status string json:"status"
    Notes *string json:"notes,omitempty"
    Images []IncomingImageResponse json:"images,omitempty"
    CreatedAt time.Time json:"created_at"
    UpdatedAt time.Time json:"updated_at"
}
```

Kode 3.12: Cuplikan kode DTO response Incoming

E Model dan DTO HU

Handling Unit (HU) digunakan untuk merepresentasikan satuan kemasan atau unit logistik yang digunakan dalam proses distribusi barang. Dalam sistem Ritra Logistics, struktur HU dibagi menjadi dua bagian utama, yaitu HU sebagai entitas induk dan HU Detail sebagai rincian isi dari HU tersebut. Pendekatan ini digunakan untuk memisahkan informasi umum HU dengan detail dimensi atau karakteristik fisik dari setiap unit yang terkandung di dalamnya.

Model HU menyimpan informasi utama seperti nomor seri, kode klien, nomor OF, total berat, serta relasi terhadap Color Code dan Purchase Order. Selain itu, model ini juga memiliki relasi satu-ke-banyak dengan model HU Detail, yang memungkinkan satu HU memiliki beberapa detail item di dalamnya. Cuplikan kode model HU dapat dilihat pada Kode 3.13.

```
package models

import (
"time"
"github.com/google/uuid"
"gorm.io/gorm"


)

type HU struct {
    ID uint json:"id" gorm:"primaryKey;autoIncrement"
    UUID string json:"uuid" gorm:"type:char(36);uniqueIndex;not
    null"
    Series string json:"series" gorm:"type:varchar(100);not null"
    ClientSeries string json:"client_series" gorm:"type:varchar
    (100);not null"
    OF string json:"of" gorm:"type:varchar(50);not null"
    TotalWeight float64 json:"total_weight" gorm:"type:decimal
    (10,2)"

    IsFM          bool    `json:"is_fm" gorm:"not null;default:false"
    `
    ColorCodeID *uint   `json:"color_code_id" gorm:"index"`

    Details []HUDetail `json:"details" gorm:"foreignKey:HUID;
    constraint:OnUpdate:CASCADE, OnDelete:CASCADE"`
    POs       []PO         `json:"po" gorm:"foreignKey:HUID;constraint
    :OnUpdate:CASCADE, OnDelete:SET NULL;"`

    CreatedAt time.Time       `json:"created_at" gorm:"
    autoCreateTime"`
    CreatedBy *uint           `json:"created_by"`
    UpdatedAt time.Time       `json:"updated_at" gorm:"
    autoUpdateTime"`
    UpdatedBy *uint           `json:"updated_by"`
    DeletedAt gorm.DeletedAt `json:"-" gorm:"index"`
```

```
}
```

Kode 3.13: Cuplikan kode model HU

HU Detail berfungsi untuk menyimpan informasi rinci dari setiap unit barang yang terdapat di dalam satu HU. Data yang disimpan meliputi dimensi fisik, berat, jenis satuan, serta deskripsi tambahan. Setiap HU Detail terhubung langsung dengan satu HU melalui foreign key. Cuplikan kode model HU Detail dapat dilihat pada Kode 3.14.

```
type HUDetail struct {
    ID uint json:"id" gorm:"primaryKey;autoIncrement"
    HUID uint json:"hu_id" gorm:"not null;index"
    UUID string json:"uuid" gorm:"type:char(36);uniqueIndex;not null"

    Length float64 `json:"length" gorm:"type:decimal(10,2);not null"`
    Width float64 `json:"width" gorm:"type:decimal(10,2);not null"`
    Height float64 `json:"height" gorm:"type:decimal(10,2);not null"`
    Weight *float64 `json:"weight" gorm:"type:decimal(10,2)"`

    Units string `json:"units" gorm:"type:varchar(20);not null"`
    HU string `json:"hu" gorm:"type:varchar(50);not null"`
    Dimension float64 `json:"dimension" gorm:"type:decimal(10,2);not null"`
    HUDescription string `json:"hu_description" gorm:"type:varchar(255)"`

    CreatedAt time.Time `json:"created_at" gorm:"autoCreateTime"`
    UpdatedAt time.Time `json:"updated_at" gorm:"autoUpdateTime"`
    DeletedAt gorm.DeletedAt `json:"-" gorm:"index"`
}
```

Kode 3.14: Cuplikan kode model HU Detail

Untuk pertukaran data antara frontend dan backend, digunakan DTO yang memisahkan kebutuhan input dan output. DTO request digunakan saat pembuatan HU baru, yang berisi data utama HU serta daftar detailnya. Sementara itu, DTO

response digunakan untuk mengembalikan data HU beserta detail dan relasi lain seperti PO dan informasi pembaruan. Cuplikan kode DTO request dapat dilihat pada Kode 3.15.

```
package requests

type HUDetailRequest struct {
    Length float64 json:"length" validate:"required"
    Width float64 json:"width" validate:"required"
    Height float64 json:"height" validate:"required"
    Weight *float64 json:"weight"

    Units          string  `json:"units" validate:"required"`
    HU             string  `json:"hu" validate:"required"`
    Dimension      float64 `json:"dimension" validate:"required"`
    HUDescription *string `json:"hu_description"`
}

type HURequest struct {
    Series string json:"series" validate:"required"
    ClientSeries *string json:"client_series" validate:"required"
    OF string json:"of" validate:"required"
    Details []HUDetailRequest json:"details" validate:"required,
    dive"
}
```

Kode 3.15: Cuplikan kode DTO request HU

Sebagai hasil akhir, sistem akan mengembalikan DTO response yang memuat informasi lengkap mengenai HU, termasuk detail barang dan relasi terhadap PO. DTO ini digunakan oleh frontend untuk menampilkan data secara terstruktur kepada pengguna. Cuplikan kode DTO response dapat dilihat pada Kode 3.16.

```
package responses

type HUDetailResponse struct {
    UUID string json:"uuid"
    Length float64 json:"length"
    Width float64 json:"width"
    Height float64 json:"height"
    Weight *float64 json:"weight"
    Units string json:"units"
    HU string json:"hu"
    Dimension float64 json:"dimension"
```

```
    HUDescription string json:"hu_description"
}

type HUResponse struct {
    UUID string json:"uuid"
    Series string json:"series"
    ClientSeries string json:"client_series"
    OF string json:"of"
    IsFM bool json:"is_fm"
    ColorCode string json:"color_code"

    Details []HUDetailResponse `json:"details"`
    POs     []POResponse       `json:"po"`

    UpdatedBy *UpdateHUResponse `json:"updatedBy"`
}
```

Kode 3.16: Cuplikan kode DTO response HU

F Model dan DTO PO

Purchase Order (PO) merupakan entitas utama dalam proses operasional Ritra Logistics yang merepresentasikan permintaan atau pesanan barang dari vendor. PO menjadi pusat keterkaitan berbagai proses lanjutan seperti incoming, pemeriksaan barang (article check), wrapping, hingga stuffing. Oleh karena itu, model PO memiliki banyak relasi dengan entitas lain seperti Vendor, Color Code, HU, dan berbagai proses operasional lainnya.

Model PO menyimpan informasi dasar seperti nomor PO, tanggal, vendor, serta berbagai atribut operasional seperti status FM, posisi barang, dan estimasi waktu proses. Selain itu, model ini juga menyimpan relasi ke beberapa tabel turunan seperti POProduct, Incoming, ArticleCheck, Wrapping, dan Stuffing. Cuplikan kode model PO dapat dilihat pada Kode 3.17.

```
type PO struct {
    ID uint json:"id" gorm:"primaryKey;autoIncrement"
    UUID string json:"uuid" gorm:"type:char(36);uniqueIndex;not
    null"
    PONumber string json:"po_number" gorm:"type:varchar(100);not
    null"

    VendorID uint     `json:"vendor_id" gorm:"not null"`
```

```
    Vendor    *Vendor `json:"vendor" gorm:"foreignKey:VendorID;
constraint:OnUpdate:CASCADE, OnDelete:RESTRICT"`

    ColorCodeID uint         `json:"color_code_id"`
    ColorCode   *ColorCode `json:"color_code" gorm:"foreignKey:
ColorCodeID;constraint:OnUpdate:CASCADE, OnDelete:SET NULL"`

    IncomingDuration      *float64 `json:"incoming_duration" gorm:"
type:double precision"`
    ArticleCheckDuration *float64 `json:"article_check_duration"
gorm:"type:double precision"`
    WrappingDuration      *float64 `json:"wrapping_duration" gorm:"
type:double precision"`
    StorageDuration       *float64 `json:"storage_duration" gorm:"
type:double precision"`
    StuffingDuration      *float64 `json:"stuffing_duration" gorm:"
type:double precision"`
    TotalLeadTime         *float64 `json:"total_lead_time" gorm:"
type:double precision"`

    StorageEndAt *time.Time `json:"storage_end_at"`

    HUID *uint `json:"hu_id"`
    HU   *HU   `json:"hu" gorm:"foreignKey:HUID;constraint:
OnUpdate:CASCADE, OnDelete:SET NULL"`

    POProducts     []POProduct     `json:"po_products" gorm:"
foreignKey:POID;constraint:OnUpdate:CASCADE, OnDelete:CASCADE"`
    Incomings      []Incoming      `json:"incomings" gorm:"
foreignKey:POID;constraint:OnUpdate:CASCADE, OnDelete:SET NULL"`
    ArticleChecks []ArticleCheck `json:"article_checks" gorm:"
foreignKey:POID;constraint:OnUpdate:CASCADE, OnDelete:SET NULL"`
    Wrappings      []Wrapping      `json:"wrappings" gorm:"
foreignKey:POID;constraint:OnUpdate:CASCADE, OnDelete:SET NULL"`
    Stuffings      []Stuffing      `json:"stuffings" gorm:"
foreignKey:POID;constraint:OnUpdate:CASCADE, OnDelete:SET NULL"`

    Date          time.Time `json:"date" gorm:"not null"`
    FM            bool      `json:"fm" gorm:"default:false"`
    DG            string    `json:"dg" gorm:"type:varchar(50)"`
    VerifyBy      *uint     `json:"verify_by"`
    Position      string    `json:"position" gorm:"type:varchar(100)
"`
```

```
    Note          string     `json:"note" gorm:"type:text"`
    AvgLeadTime   float64    `json:"avg_lead_time" gorm:"type:decimal
    (10,2);default:0"`
    QueueNumber   int        `json:"queue_number" gorm:"not null"`

    CreatedAt     time.Time      `json:"created_at" gorm:"
    autoCreateTime"`
    CreatedBy     *uint          `json:"created_by"`
    UpdatedAt     time.Time      `json:"updated_at" gorm:"
    autoUpdateTime"`
    UpdatedBy     *uint          `json:"updated_by"`
    DeletedAt     gorm.DeletedAt `json:"-" gorm:"index"`
}
```

Kode 3.17: Cuplikan kode model PO

Untuk menerima data dari frontend, digunakan DTO request yang berfungsi sebagai representasi data input saat pembuatan atau pembaruan PO. DTO ini memastikan bahwa data yang masuk telah tervalidasi sesuai kebutuhan sistem, seperti validasi UUID dan format tanggal. Cuplikan kode DTO request dapat dilihat pada Kode 3.18.

```
package request

type PORequest struct {
    PONumber string json:"po_number" validate:"required"
    VendorUUID string json:"vendor_uuid" validate:"required,uuid4"
    Date string json:"date" validate:"required,datetime=2006-01-02
    "
    ColorCodeUUID string json:"color_code_uuid" validate:"
    omitempty,uuid4"
    FM bool json:"fm"
    DG string json:"dg" validate:"omitempty"
    VerifyBy *uint json:"verify_by" validate:"omitempty"
    Position string json:"position" validate:"omitempty"
    Note string json:"note" validate:"omitempty"
    HUUUID string json:"hu_uuid" validate:"omitempty,uuid4"
    Products []POProductRequest json:"products" validate:"
    omitempty,dive"
}
```

Kode 3.18: Cuplikan kode DTO request PO

Sebagai respons, sistem mengembalikan DTO yang berisi informasi lengkap terkait PO, termasuk relasi terhadap vendor, HU, serta seluruh proses lanjutan yang

terhubung. DTO ini digunakan oleh frontend untuk menampilkan status dan detail PO secara komprehensif. Cuplikan kode DTO response dapat dilihat pada Kode 3.19.

```go
package responses

type POResponse struct {
    UUID string json:"uuid"
    PONumber string json:"po_number"
    Date string json:"date"
    ColorCodeUUID string json:"color_code_uuid,omitempty"
    ColorCode *ColorCodeResponse json:"color_code,omitempty"
    FM bool json:"fm"
    DG string json:"dg,omitempty"
    VerifyBy *uint json:"verify_by,omitempty"
    Position string json:"position,omitempty"
    PositionString string json:"position_string,omitempty"
    Note string json:"note,omitempty"
    VendorUUID string json:"vendor_uuid,omitempty"
    Vendor *VendorResponse json:"vendor,omitempty"
    HUUUID string json:"hu_uuid,omitempty"
    HU *HUResponse json:"hu,omitempty"
    QueueNumber int json:"queue_number"

    Products []POProductResponse `json:"products,omitempty"`

    Incoming *POIncomingResponse `json:"incoming,omitempty"`
    ArticleCheck *POArticleCheckResponse `json:"article_check,omitempty"`
    Wrapping *POWrappingResponse `json:"wrapping,omitempty"`
    Storage *StorageResponse `json:"storage,omitempty"`
    Stuffing *POStuffingResponse `json:"stuffing,omitempty"`

    TotalLeadTime string `json:"total_lead_time,omitempty"`
}
```

Kode 3.19: Cuplikan kode DTO response PO

3.3.4 Pembuatan Rute

A Rute *Authentication*

Route autentikasi digunakan untuk menangani proses pendaftaran dan autentikasi pengguna dalam sistem Ritra Logistics. Seluruh endpoint autentikasi dikelompokkan dalam satu route group dengan prefix /auth untuk memudahkan pengelolaan dan menjaga konsistensi struktur API. Pada bagian ini, tersedia dua endpoint utama, yaitu register dan login, yang masing-masing menangani proses pembuatan akun pengguna baru serta proses autentikasi pengguna yang telah terdaftar.

Cuplikan kode pendefinisian route autentikasi dapat dilihat pada Kode 3.20.

```
auth := router.Group("/auth")

auth.Post("/register", authHandler.Register)
auth.Post("/login", authHandler.Login)
```

Kode 3.20: Cuplikan kode route autentikasi

B Rute *ColorCode*

Route Color Code digunakan untuk mengelola data kode warna yang merepresentasikan area atau klasifikasi tertentu dalam sistem Ritra Logistics. Seluruh endpoint pada modul ini berada dalam satu grup dengan prefix /color-codes dan dilindungi oleh authentication middleware untuk memastikan hanya pengguna yang telah terautentikasi yang dapat mengaksesnya. Selain itu, setiap endpoint juga dilengkapi dengan pemeriksaan permission sesuai dengan jenis operasi yang dilakukan.

Cuplikan kode pendefinisian route Color Code dapat dilihat pada Kode 3.21.

```
color := router.Group("/color-codes", middlewares.AuthMiddleware()
    )

color.Get("/get-all",
middlewares.RequirePermission(db, "color_code.view"),
colorHandler.GetAll,
)

color.Get("/get/:uuid",
middlewares.RequirePermission(db, "color_code.view"),
```

```
10 colorHandler.GetByUUID,
11 )
12
13 color.Post("/create",
14 middlewares.RequirePermission(db, "color_code.create"),
15 colorHandler.Create,
16 )
17
18 color.Put("/update/:uuid",
19 middlewares.RequirePermission(db, "color_code.update"),
20 colorHandler.Update,
21 )
22
23 color.Delete("/delete/:uuid",
24 middlewares.RequirePermission(db, "color_code.delete"),
25 colorHandler.Delete,
26 )
```

Kode 3.21: Cuplikan kode route Color Code

C Rute *Vendor*

Route Vendor digunakan untuk mengelola data pemasok yang terdaftar dalam sistem Ritra Logistics. Seluruh endpoint pada modul ini dikelompokkan dalam satu route group dengan prefix /vendors dan dilindungi oleh authentication middleware untuk memastikan bahwa hanya pengguna yang telah terautentikasi yang dapat mengakses layanan ini. Setiap operasi juga dibatasi menggunakan mekanisme permission sesuai dengan hak akses pengguna.

Cuplikan kode pendefinisian route Vendor dapat dilihat pada Kode 3.22.

```
1 vendor := router.Group("/vendors", middlewares.AuthMiddleware())
2
3 vendor.Get("/get-all",
4 middlewares.RequirePermission(db, "vendor.view"),
5 vendorHandler.GetAll,
6 )
7
8 vendor.Get("/get/:uuid",
9 middlewares.RequirePermission(db, "vendor.view"),
10 vendorHandler.GetByUUID,
11 )
12
13 vendor.Post("/create",
```

```
middlewares.RequirePermission(db, "vendor.create"),
vendorHandler.Create,
)

vendor.Put("/update/:uuid",
middlewares.RequirePermission(db, "vendor.update"),
vendorHandler.Update,
)

vendor.Delete("/delete/:uuid",
middlewares.RequirePermission(db, "vendor.delete"),
vendorHandler.Delete,
)
```

Kode 3.22: Cuplikan kode route Vendor

D Rute *Incoming*

Route Incoming digunakan untuk menangani proses pencatatan dan pengelolaan data barang masuk (incoming goods) dalam sistem Ritra Logistics. Seluruh endpoint pada modul ini berada dalam satu grup dengan prefix /incoming dan dilindungi oleh authentication middleware untuk memastikan hanya pengguna yang telah terautentikasi yang dapat mengaksesnya. Setiap endpoint juga dilengkapi dengan pengecekan permission sesuai dengan jenis operasi yang dilakukan.

Cuplikan kode pendefinisian route Incoming dapat dilihat pada Kode 3.23.

```
incoming := router.Group("/incoming", middlewares.AuthMiddleware()
    )

incoming.Get("/get-all",
middlewares.RequirePermission(db, "incoming.view"),
incomingHandler.GetAll,
)

incoming.Get("/get/:uuid",
middlewares.RequirePermission(db, "incoming.view"),
incomingHandler.GetByUUID,
)

incoming.Post("/create",
middlewares.RequirePermission(db, "incoming.create"),
```

```
15 incomingHandler.Create,
16 )
17
18 incoming.Put("/update/:uuid",
19 middlewares.RequirePermission(db, "incoming.update"),
20 incomingHandler.Update,
21 )
22
23 incoming.Delete("/delete/:uuid",
24 middlewares.RequirePermission(db, "incoming.delete"),
25 incomingHandler.Delete,
26 )
```

Kode 3.23: Cuplikan kode route Incoming

E Rute *HU*

Route Handling Unit (HU) digunakan untuk mengambil data detail HU berdasarkan UUID tertentu. Endpoint ini digunakan untuk menampilkan informasi lengkap terkait HU, termasuk detail unit dan relasi dengan data lain seperti PO. Seluruh akses ke route ini dilindungi oleh authentication middleware dan pengecekan permission agar hanya pengguna dengan hak akses yang sesuai yang dapat mengakses data tersebut.

Cuplikan kode pendefinisian route HU dapat dilihat pada Kode 3.24.

```
1 hu := router.Group("/hu", middlewares.AuthMiddleware())
2
3 hu.Get("/get/:uuid",
4 middlewares.RequirePermission(db, "hu.view"),
5 huHandler.GetByUUID,
6 )
```

Kode 3.24: Cuplikan kode route HU

F Rute *PO*

Route Purchase Order (PO) digunakan untuk mengambil data PO berdasarkan nomor PO yang diberikan. Endpoint ini berfungsi untuk menampilkan informasi detail terkait satu PO beserta relasi yang dimilikinya, seperti vendor, HU, dan proses operasional lainnya. Akses terhadap route ini dilindungi oleh authentication middleware serta pengecekan permission untuk memastikan hanya pengguna dengan hak akses yang sesuai yang dapat menggunakannya.

Cuplikan kode pendefinisian route PO dapat dilihat pada Kode 3.25.

```
po := router.Group("/po", middlewares.AuthMiddleware())

po.Get("/get-by-number/:po_number",
middlewares.RequirePermission(db, "po.view"),
poHandler.GetByPONumber,
)
```

Kode 3.25: Cuplikan kode route PO

3.3.5 Pembuatan *Handlers*, *Services*, dan *Repositories*

Dalam pengembangan sistem backend pada aplikasi Ritra Logistics, struktur kode dibagi ke dalam tiga lapisan utama, yaitu handler, service, dan repository. Pembagian ini bertujuan untuk menerapkan prinsip pemisahan tanggung jawab (separation of concerns) sehingga kode menjadi lebih terstruktur, mudah dipelihara, serta lebih mudah dikembangkan di kemudian hari.

Lapisan handler berfungsi sebagai titik masuk pertama ketika sebuah endpoint diakses oleh pengguna. Pada bagian ini, sistem menangani proses yang berkaitan langsung dengan HTTP request, seperti membaca parameter URL, mengambil data dari body request, melakukan validasi awal, serta membentuk respons HTTP yang akan dikirimkan kembali ke klien. Handler tidak berisi logika bisnis yang kompleks, melainkan hanya berperan sebagai penghubung antara permintaan dari client dan proses internal sistem.

Selanjutnya, lapisan service bertanggung jawab terhadap seluruh logika bisnis aplikasi. Pada bagian ini dilakukan pengolahan data, validasi lanjutan, pengambilan keputusan bisnis, serta pengaturan alur proses sesuai kebutuhan sistem. Oleh karena itu, kode pada layer service umumnya lebih kompleks dan menjadi pusat dari aturan bisnis aplikasi.

Adapun lapisan repository berfungsi sebagai penghubung langsung dengan basis data. Layer ini bertugas melakukan operasi penyimpanan, pembacaan, pembaruan, dan penghapusan data tanpa melibatkan logika bisnis. Dengan pemisahan ini, proses akses data menjadi lebih terorganisir dan mudah untuk diuji maupun dikembangkan di kemudian hari.

Struktur pemisahan antara handler, service, dan repository ini diterapkan secara konsisten pada seluruh modul sistem, seperti autentikasi, color code, vendor, incoming, handling unit, dan purchase order, sehingga arsitektur backend menjadi lebih modular, terstruktur, dan mudah dipelihara.

A Register

Pada bagian handler, fungsi Register akan dijalankan ketika endpoint registrasi diakses. Fungsi ini bertugas membaca data dari request body, mengambil informasi tambahan seperti alamat IP dan user agent, lalu meneruskan data tersebut ke service untuk diproses lebih lanjut.

```
func (h *AuthHandler) Register(c *fiber.Ctx) error {
  var request requests.RegisterRequest

  if err := c.BodyParser(&request); err != nil {
    return c.Status(http.StatusBadRequest).JSON(fiber.Map{
      "error": "invalid request body",
    })
  }

  ip := c.IP()
  userAgent := c.Get("User-Agent")

  _, userDTO, err := h.authService.Register(request, ip, userAgent
    )
  if err != nil {
    return c.Status(http.StatusInternalServerError).JSON(fiber.Map
    {
      "error": err.Error(),
    })
  }

  return c.Status(http.StatusCreated).JSON(responses.
    MessageResponse{
    Message: "registration successful",
    Data:    userDTO,
  })
}
```

Setelah data diterima oleh handler, proses dilanjutkan ke lapisan service. Pada tahap ini dilakukan berbagai proses inti seperti hashing password, pembuatan UUID, penyimpanan data pengguna, pembuatan token JWT, serta pencatatan aktivitas pengguna.

```
func (s *authService) Register(request requests.RegisterRequest,
    ip, userAgent string)
  (string, responses.AuthResponse, error) {

```

```
  hashedPassword, err := bcrypt.GenerateFromPassword(
    []byte(request.Password), bcrypt.DefaultCost,
  )
  if err != nil {
    return "", responses.AuthResponse{}, err
  }

  createdBy := uint(0)

  newUser := models.User{
    UUID:      uuid.New().String(),
    Name:      request.Name,
    Email:     request.Email,
    Password:  string(hashedPassword),
    CreatedBy: &createdBy,
  }

  err = s.authRepo.Create(&newUser)
  if err != nil {
    return "", responses.AuthResponse{}, err
  }

  token, err := pkg.GenerateJWT(newUser.ID, newUser.UUID)
  if err != nil {
    return "", responses.AuthResponse{}, err
  }

  _ = s.logService.CreateLog(
    newUser.ID, "REGISTER", "auth", ip, userAgent, "", "",
  )

  return token, responses.AuthResponse{
    UUID:  newUser.UUID,
    Name:  newUser.Name,
    Email: newUser.Email,
  }, nil
}
```

Lapisan service berfungsi sebagai pusat pengambilan keputusan dan memastikan seluruh proses bisnis berjalan sesuai aturan yang telah ditetapkan.

Lapisan repository merupakan lapisan paling bawah yang berinteraksi langsung dengan database. Fungsi ini bertanggung jawab menyimpan data pengguna menggunakan ORM GORM serta mengelola context dan timeout untuk

keamanan eksekusi query.

```
func (r *authRepository) Create(user *models.User) error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  return r.db.WithContext(ctx).Create(user).Error
}
```

Repository tidak memiliki logika bisnis dan hanya berfokus pada operasi database, sehingga struktur kode menjadi lebih bersih dan mudah dipelihara.

B Login

Handler login bertanggung jawab untuk menerima permintaan dari klien, melakukan parsing data request, serta mengatur respons HTTP yang dikembalikan. Selain itu, handler juga menyimpan token hasil autentikasi ke dalam cookie agar dapat digunakan pada permintaan berikutnya.

```
func (h *AuthHandler) Login(c *fiber.Ctx) error {
  var request requests.LoginRequest

  if err := c.BodyParser(&request); err != nil {
    return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
      "error": "Invalid request body",
    })
  }

  ip := c.IP()
  userAgent := c.Get("User-Agent")

  token, user, err := h.authService.Login(
    request.Email, request.Password, ip, userAgent,
  )
  if err != nil {
    return c.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
      "error": err.Error(),
    })
  }

  c.Cookie(&fiber.Cookie{
    Name:     "token",
    Value:    token,
```

```
    Expires:  time.Now().Add(24 * time.Hour),
    HTTPOnly: true,
    Secure:   true,
    SameSite: "None",
    Path:     "/",
  })

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "login successful",
    Data:    user,
  })
}
```

Handler hanya bertugas sebagai penghubung antara klien dan service, tanpa menyentuh proses validasi bisnis maupun akses database secara langsung.

Lapisan service menangani proses autentikasi utama, termasuk pencarian data pengguna, validasi kata sandi menggunakan bcrypt, pembuatan token JWT, serta pencatatan aktivitas login ke dalam sistem log.

```
func (s *authService) Login(email, password, ip, userAgent string)
  (string, responses.AuthResponse, error) {

  user, err := s.authRepo.GetByEmail(email)
  if err != nil {
    return "", responses.AuthResponse{}, errors.New("invalid email
    or password")
  }

  if err := bcrypt.CompareHashAndPassword(
    []byte(user.Password), []byte(password),
  ); err != nil {
    return "", responses.AuthResponse{}, errors.New("invalid email
    or password")
  }

  token, err := pkg.GenerateJWT(user.ID, user.UUID)
  if err != nil {
    return "", responses.AuthResponse{}, err
  }

  authResponse := responses.AuthResponse{
    UUID:  user.UUID,
    Name:  user.Name,
```

```
    Email: user.Email,
  }

  updatedBy := uint(0)
  if user.UpdatedBy != nil {
    updatedBy = *user.UpdatedBy
  }

  _ = s.logService.CreateLog(
    user.ID, "LOGIN", "auth", ip, userAgent,
    fmt.Sprintf("%d", updatedBy), "",
  )

  return token, authResponse, nil
}
```

Service berfungsi sebagai pusat logika autentikasi, termasuk verifikasi kredensial dan pembuatan token akses yang akan digunakan pada permintaan berikutnya.

Repository bertugas melakukan komunikasi langsung dengan basis data. Pada proses login, repository hanya mengambil data pengguna berdasarkan email yang diberikan.

```
func (r *authRepository) GetByEmail(email string) (*models.User,
    error) {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  var user models.User
  if err := r.db.WithContext(ctx).
    Where("email = ?", email).
    First(&user).Error; err != nil {
    return nil, err
  }

  return &user, nil
}
```

C Create Color Code

Handler bertugas menerima data dari request, melakukan validasi awal, serta mengambil informasi pengguna yang sedang login dari konteks. Jika seluruh data valid, handler akan memanggil service untuk membuat data color code baru.

```
func (h *ColorCodeHandler) Create(c *fiber.Ctx) error {
  var req requests.ColorCodeRequest
  if err := c.BodyParser(&req); err != nil {
    return c.Status(fiber.StatusBadRequest).JSON(responses.
    ErrorResponse{
      Error: "invalid request body",
    })
  }

  userID, ok := c.Locals("user_id").(uint)
  if !ok || userID == 0 {
    return c.Status(fiber.StatusUnauthorized).JSON(responses.
    ErrorResponse{
      Error: "unauthorized",
    })
  }

  colorCode, err := h.service.Create(req, &userID)
  if err != nil {
    return c.Status(http.StatusInternalServerError).JSON(responses
    .ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusCreated).JSON(responses.
    MessageResponse{
    Message: "color code created successfully",
    Data:    colorCode,
  })
}
```

Service bertanggung jawab membentuk entitas ColorCode berdasarkan data request, sekaligus menetapkan nilai CreatedBy. Setelah itu, data diteruskan ke repository untuk disimpan ke database.

```
func (s *colorCodeService) Create(
  req requests.ColorCodeRequest,
```

```
  createdBy *uint,
) (responses.ColorCodeResponse, error) {

  colorCode := models.ColorCode{
    Name:        req.Area,
    Code:        req.ColorCode,
    Highland:    req.Highland,
    Description: req.Description,
    Bloc:        req.Sloc,
    CreatedBy:   createdBy,
  }

  if err := s.repo.Create(&colorCode); err != nil {
    return responses.ColorCodeResponse{}, err
  }

  return responses.ColorCodeResponse{
    UUID:        colorCode.UUID,
    ColorCode:   colorCode.Code,
    Area:        colorCode.Name,
    Highland:    colorCode.Highland,
    Description: colorCode.Description,
    Sloc:        colorCode.Bloc,
  }, nil
}
```

Repository menangani proses penyimpanan data color code ke dalam database menggunakan ORM. Proses ini dilakukan dalam konteks dengan batas waktu tertentu untuk menjaga stabilitas koneksi.

```
func (r *colorCodeRepository) Create(colorCode *models.ColorCode)
    error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  return r.db.WithContext(ctx).Create(colorCode).Error
}
```


D Get All Color Code

Handler bertugas menerima permintaan dari endpoint dan memanggil fungsi service untuk mengambil seluruh data color code. Jika terjadi kesalahan selama

proses, handler akan mengembalikan respons error, sedangkan jika berhasil, data akan dikembalikan dalam format respons yang telah ditentukan.

```
func (h *ColorCodeHandler) GetAll(c *fiber.Ctx) error {
  colorCodes, err := h.service.GetAll()
  if err != nil {
    return c.Status(http.StatusInternalServerError).JSON(responses
    .ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "color codes retrieved successfully",
    Data:    colorCodes,
  })
}
```

Pada bagian service, data color code diambil dari repository, kemudian dilakukan proses transformasi dari model database menjadi bentuk response yang akan dikirim ke frontend. Proses ini memastikan hanya data yang diperlukan saja yang dikirimkan.

```
func (s *colorCodeService) GetAll() ([]responses.ColorCodeResponse
   , error) {
  colorCodes, err := s.repo.GetAll()
  if err != nil {
    return nil, err
  }

  var res []responses.ColorCodeResponse
  for _, c := range colorCodes {
    res = append(res, responses.ColorCodeResponse{
      UUID:        c.UUID,
      ColorCode:   c.Code,
      Area:        c.Name,
      Highland:    c.Highland,
      Description: c.Description,
      Sloc:        c.Bloc,
    })
  }

  return res, nil
}
```

Repository bertanggung jawab mengambil seluruh data color code dari database menggunakan ORM. Hasil query kemudian dikembalikan ke service untuk diproses lebih lanjut.

```
func (r *colorCodeRepository) GetAll() ([]models.ColorCode, error) {
  ctx, cancel := context.WithTimeout(context.Background(), dbTimeout)
  defer cancel()

  var colorCodes []models.ColorCode
  if err := r.db.WithContext(ctx).Find(&colorCodes).Error; err != nil {
    return nil, err
  }
  return colorCodes, nil
}
```

E Get Color Code By UUID

Handler bertugas mengambil parameter uuid dari URL dan meneruskannya ke service. Jika service mengembalikan error, handler akan mengirimkan respons not found ke frontend.

```
func (h *ColorCodeHandler) GetByUUID(c *fiber.Ctx) error {
  uuid := c.Params("uuid")
  colorCode, err := h.service.GetByUUID(uuid)
  if err != nil {
    return c.Status(http.StatusNotFound).JSON(responses.ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "color code retrieved successfully",
    Data:    colorCode,
  })
}
```

Service bertugas memanggil repository untuk mengambil data color code berdasarkan UUID. Jika data ditemukan, service akan memetakan hasilnya ke dalam bentuk response yang dikirimkan ke frontend.

```go
func (s *colorCodeService) GetByUUID(uuid string) (responses.
    ColorCodeResponse, error) {
  c, err := s.repo.GetByUUID(uuid)
  if err != nil {
    return responses.ColorCodeResponse{}, err
  }

  return responses.ColorCodeResponse{
    UUID:        c.UUID,
    ColorCode:   c.Code,
    Area:        c.Name,
    Highland:    c.Highland,
    Description: c.Description,
    Sloc:        c.Bloc,
  }, nil
}
```

Repository melakukan pencarian data color code berdasarkan UUID di dalam database. Jika data tidak ditemukan, repository akan mengembalikan error yang menandakan bahwa data tidak tersedia.

```go
func (r *colorCodeRepository) GetByUUID(uuid string) (models.
    ColorCode, error) {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  var colorCode models.ColorCode
  if err := r.db.WithContext(ctx).Where("uuid = ?", uuid).First(&
    colorCode).Error; err != nil {
    if errors.Is(err, gorm.ErrRecordNotFound) {
      return colorCode, errors.New("color code not found")
    }
    return colorCode, err
  }
  return colorCode, nil
}
```

F Update Color Code

Handler bertugas mengambil parameter UUID, membaca data pembaruan dari request body, serta mengambil informasi pengguna yang sedang login. Setelah itu, handler memanggil service untuk melakukan proses pembaruan data.

```go
func (h *ColorCodeHandler) Update(c *fiber.Ctx) error {
  uuid := c.Params("uuid")

  var req requests.ColorCodeRequest
  if err := c.BodyParser(&req); err != nil {
    return c.Status(fiber.StatusBadRequest).JSON(responses.
    ErrorResponse{
      Error: "invalid request body",
    })
  }

  userID, ok := c.Locals("user_id").(uint)
  if !ok || userID == 0 {
    return c.Status(fiber.StatusUnauthorized).JSON(responses.
    ErrorResponse{
      Error: "unauthorized",
    })
  }

  colorCode, err := h.service.Update(uuid, req, &userID)
  if err != nil {
    return c.Status(fiber.StatusNotFound).JSON(responses.
    ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "color code updated successfully",
    Data:    colorCode,
  })
}
```

Service melakukan pencarian data color code berdasarkan UUID untuk memastikan data tersedia. Jika ditemukan, data lama diperbarui dengan nilai baru dari request, kemudian disimpan kembali melalui repository.

```go
func (s *colorCodeService) Update(
  uuid string,
  req requests.ColorCodeRequest,
  updatedBy *uint,
) (responses.ColorCodeResponse, error) {

  existing, err := s.repo.GetByUUID(uuid)
```

```
  if err != nil {
    return responses.ColorCodeResponse{}, errors.New("color code
     not found")
  }

  existing.Name = req.Area
  existing.Code = req.ColorCode
  existing.Highland = req.Highland
  existing.Description = req.Description
  existing.Bloc = req.Sloc
  existing.UpdatedBy = updatedBy

  if err := s.repo.Update(&existing); err != nil {
    return responses.ColorCodeResponse{}, err
  }

  return responses.ColorCodeResponse{
    UUID:        existing.UUID,
    ColorCode:   existing.Code,
    Area:        existing.Name,
    Highland:    existing.Highland,
    Description: existing.Description,
    Sloc:        existing.Bloc,
  }, nil
}
```

Repository melakukan pembaruan data color code di database berdasarkan UUID. Proses ini memastikan bahwa hanya data dengan UUID yang sesuai yang diperbarui.

```
func (r *colorCodeRepository) Update(colorCode *models.ColorCode)
    error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  return r.db.WithContext(ctx).
    Model(&models.ColorCode{}).
    Where("uuid = ?", colorCode.UUID).
    Updates(colorCode).Error
}
```

G Delete Color Code

Handler bertugas mengambil parameter UUID dari URL dan meneruskannya ke service untuk diproses. Jika terjadi kesalahan, seperti data tidak ditemukan, handler akan mengembalikan respons error ke frontend.

```
func (h *ColorCodeHandler) Delete(c *fiber.Ctx) error {
  uuid := c.Params("uuid")
  if err := h.service.Delete(uuid); err != nil {
    return c.Status(fiber.StatusNotFound).JSON(responses.
    ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.
    MessageOnlyResponse{
    Message: "color code deleted successfully",
  })
}
```

Service berfungsi sebagai perantara untuk meneruskan permintaan penghapusan ke repository. Pada tahap ini tidak dilakukan proses tambahan selain pemanggilan fungsi repository.

```
func (s *colorCodeService) Delete(uuid string) error {
  return s.repo.Delete(uuid)
}
```

Repository bertanggung jawab untuk menghapus data color code dari database berdasarkan UUID. Jika tidak ada data yang terhapus, repository akan mengembalikan error yang menandakan bahwa data tidak ditemukan.

```
func (r *colorCodeRepository) Delete(uuid string) error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  result := r.db.WithContext(ctx).Where("uuid = ?", uuid).Delete(&
    models.ColorCode{})
  if result.RowsAffected == 0 {
    return errors.New("color code not found")
  }
  return result.Error
}
```

H Create Vendor

Handler menerima data vendor dari request body, mengambil informasi pengguna yang sedang login, lalu meneruskan data tersebut ke service. Jika terjadi kesalahan pada proses validasi atau penyimpanan, handler akan mengembalikan respons error ke frontend.

```
func (h *VendorHandler) Create(c *fiber.Ctx) error {
  fmt.Println("Handler Create START")

  var req requests.VendorRequest
  if err := c.BodyParser(&req); err != nil {
    fmt.Println("BodyParser error:", err)
    return c.Status(fiber.StatusBadRequest).JSON(responses.
   ErrorResponse{
      Error: "invalid request body",
    })
  }

  userID, ok := c.Locals("user_id").(uint)
  fmt.Println("userID:", userID, "OK?", ok)

  fmt.Println("Request:", req)

  vendor, err := h.service.Create(req, userID)
  if err != nil {
    fmt.Println("Service.Create error:", err)
    return c.Status(http.StatusInternalServerError).JSON(responses
   .ErrorResponse{
      Error: err.Error(),
    })
  }

  fmt.Println("Vendor created:", vendor)

  return c.Status(http.StatusCreated).JSON(responses.
   MessageResponse{
    Message: "vendor created successfully",
    Data:    vendor,
  })
}
```

Service bertanggung jawab membentuk objek vendor berdasarkan data dari

request, termasuk menetapkan nilai UUID dan informasi pembuat data. Setelah itu, data diteruskan ke repository untuk disimpan ke database.

```
func (s *vendorService) Create(req requests.VendorRequest,
    createdBy uint) (responses.VendorResponse, error) {

  fmt.Println("Create Vendor by user:", createdBy)

  vendor := models.Vendor{
    UUID:      uuid.NewString(),
    Name:      req.Name,
    Address:   req.Address,
    Phone:     req.Phone,
    FaxNumber: req.FaxNumber,
    Email:     req.Email,
    CreatedBy: createdBy,
  }

  if err := s.repo.Create(&vendor); err != nil {
    return responses.VendorResponse{}, err
  }

  return responses.VendorResponse{
    UUID:      vendor.UUID,
    Name:      vendor.Name,
    Address:   vendor.Address,
    Phone:     vendor.Phone,
    FaxNumber: vendor.FaxNumber,
    Email:     vendor.Email,
  }, nil
}
```

Repository bertugas menyimpan data vendor ke dalam database menggunakan ORM. Proses ini dilakukan dalam konteks yang dibatasi waktu untuk menjaga stabilitas koneksi.

```
func (r *vendorRepository) Create(vendor *models.Vendor) error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  return r.db.WithContext(ctx).Create(vendor).Error
}
```

I Get All Vendor

Handler bertugas menerima request GET, memanggil service untuk mengambil semua data vendor, dan mengembalikan hasilnya ke frontend. Jika terjadi error, handler akan mengirimkan respons error.

```
func (h *VendorHandler) GetAll(c *fiber.Ctx) error {
  vendors, err := h.service.GetAll()
  if err != nil {
    return c.Status(http.StatusInternalServerError).JSON(responses
    .ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "vendors retrieved successfully",
    Data:    vendors,
  })
}
```

Service memanggil repository untuk mengambil semua data vendor dari database, lalu mengubahnya menjadi format respons yang sesuai agar bisa dikirim ke frontend.

```
func (s *vendorService) GetAll() ([]responses.VendorResponse,
   error) {
  vendors, err := s.repo.GetAll()
  if err != nil {
    return nil, err
  }

  var res []responses.VendorResponse
  for _, v := range vendors {
    res = append(res, responses.VendorResponse{
      UUID:       v.UUID,
      Name:       v.Name,
      Address:    v.Address,
      Phone:      v.Phone,
      FaxNumber: v.FaxNumber,
      Email:      v.Email,
    })
  }
  return res, nil
```

```
}
```

Repository melakukan query ke database untuk mengambil semua data vendor. Proses ini menggunakan konteks dengan batas waktu agar koneksi tetap stabil.

```
func (r *vendorRepository) GetAll() ([]models.Vendor, error) {
  ctx, cancel := context.WithTimeout(context.Background(),
   dbTimeout)
  defer cancel()

  var vendors []models.Vendor
  if err := r.db.WithContext(ctx).Find(&vendors).Error; err != nil
    {
    return nil, err
  }
  return vendors, nil
}
```

J Get Vendor By UUID

Handler menerima uuid dari parameter URL, lalu memanggil service. Jika vendor tidak ditemukan, handler mengirimkan respons error Not Found. Jika berhasil, data vendor dikirimkan ke frontend.

```
func (h *VendorHandler) GetByUUID(c *fiber.Ctx) error {
  uuid := c.Params("uuid")
  vendor, err := h.service.GetByUUID(uuid)
  if err != nil {
    return c.Status(http.StatusNotFound).JSON(responses.
   ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "vendor retrieved successfully",
    Data:    vendor,
  })
}
```

Service memanggil repository untuk mendapatkan data vendor berdasarkan UUID. Setelah data diperoleh, service membungkusnya menjadi respons yang sesuai dengan format DTO.

```
func (s *vendorService) GetByUUID(uuid string) (responses.
    VendorResponse, error) {
  v, err := s.repo.GetByUUID(uuid)
  if err != nil {
    return responses.VendorResponse{}, err
  }

  return responses.VendorResponse{
    UUID:       v.UUID,
    Name:       v.Name,
    Address:    v.Address,
    Phone:      v.Phone,
    FaxNumber: v.FaxNumber,
    Email:      v.Email,
  }, nil
}
```

Repository mengeksekusi query ke database untuk mencari vendor berdasarkan UUID. Jika tidak ditemukan, akan mengembalikan error yang diteruskan ke service.

```
func (r *vendorRepository) GetByUUID(uuid string) (*models.Vendor,
    error) {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  var vendor models.Vendor
  if err := r.db.WithContext(ctx).Where("uuid = ?", uuid).First(&
    vendor).Error; err != nil {
    return nil, err
  }
  return &vendor, nil
}
```

K Update Vendor

Handler mem-parsing body request, mengambil UUID dari parameter URL, dan memvalidasi user yang melakukan update. Jika valid, handler memanggil service untuk melakukan update. Hasilnya dikirim ke frontend.

```
func (h *VendorHandler) Update(c *fiber.Ctx) error {
  uuid := c.Params("uuid")
```

```
3
4   var req requests.VendorRequest
5   if err := c.BodyParser(&req); err != nil {
6     return c.Status(fiber.StatusBadRequest).JSON(responses.
      ErrorResponse{
7       Error: "invalid request body",
8     })
9   }
10
11  userID, ok := c.Locals("user_id").(uint)
12  if !ok || userID == 0 {
13    return c.Status(fiber.StatusUnauthorized).JSON(responses.
      ErrorResponse{
14      Error: "unauthorized",
15    })
16  }
17
18  vendor, err := h.service.Update(uuid, req, userID)
19  if err != nil {
20    return c.Status(http.StatusNotFound).JSON(responses.
      ErrorResponse{
21      Error: err.Error(),
22    })
23  }
24
25  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
26    Message: "vendor updated successfully",
27    Data:    vendor,
28  })
29 }
```

Service memeriksa apakah vendor dengan UUID yang dimaksud ada di database. Jika ada, service memperbarui seluruh field dengan data baru dan memanggil repository untuk menyimpan perubahan. Data vendor yang diperbarui dikembalikan dalam format DTO.

```
1 func (s *vendorService) Update(uuid string, req requests.
    VendorRequest, updatedBy uint) (responses.VendorResponse, error
    ) {
2   existing, err := s.repo.GetByUUID(uuid)
3   if err != nil {
4     return responses.VendorResponse{}, errors.New("vendor not
      found")
5   }
```

```
6
7   existing.Name = req.Name
8   existing.Address = req.Address
9   existing.Phone = req.Phone
10  existing.FaxNumber = req.FaxNumber
11  existing.Email = req.Email
12  existing.UpdatedBy = updatedBy
13
14  if err := s.repo.Update(existing); err != nil {
15    return responses.VendorResponse{}, err
16  }
17
18  return responses.VendorResponse{
19    UUID:      existing.UUID,
20    Name:      existing.Name,
21    Address:   existing.Address,
22    Phone:     existing.Phone,
23    FaxNumber: existing.FaxNumber,
24    Email:     existing.Email,
25  }, nil
26 }
```

Repository mengeksekusi query untuk menyimpan data vendor yang sudah diperbarui. Fungsi ini memanfaatkan GORM Save() sehingga seluruh field yang diubah akan tersimpan.

```
1 func (r *vendorRepository) Update(vendor *models.Vendor) error {
2   ctx, cancel := context.WithTimeout(context.Background(),
      dbTimeout)
3   defer cancel()
4   return r.db.WithContext(ctx).Save(vendor).Error
5 }
```

L Delete Vendor

Handler mengambil UUID vendor dari parameter URL dan memanggil service untuk melakukan penghapusan. Jika UUID tidak ditemukan, respons error dikembalikan, jika berhasil, pesan sukses dikirim ke frontend.

```
1 func (h *VendorHandler) Delete(c *fiber.Ctx) error {
2   uuid := c.Params("uuid")
3   if err := h.service.Delete(uuid); err != nil {
4     return c.Status(http.StatusNotFound).JSON(responses.
      ErrorResponse{
```

```
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.
   MessageOnlyResponse{
    Message: "vendor deleted successfully",
  })
}
```

Service memeriksa apakah vendor dengan UUID yang dimaksud ada di database. Jika ada, service memanggil repository untuk menghapus berdasarkan ID vendor. Error dikembalikan jika vendor tidak ditemukan.

```
func (s *vendorService) Delete(uuid string) error {
  vendor, err := s.repo.GetByUUID(uuid)
  if err != nil {
    return errors.New("vendor not found")
  }
  return s.repo.Delete(vendor.ID)
}
```

Repository mengeksekusi query hapus menggunakan GORM berdasarkan ID vendor. Fungsi ini hanya berfokus pada interaksi database dan mengembalikan error jika terjadi kegagalan.

```
func (r *vendorRepository) Delete(id uint) error {
  ctx, cancel := context.WithTimeout(context.Background(),
   dbTimeout)
  defer cancel()
  return r.db.WithContext(ctx).Delete(&models.Vendor{}, id).Error
}
```

M Create Incoming

Handler bertugas parsing request body dan memanggil service. Jika request valid dan user terotentikasi, service akan dijalankan, dan respons success dikirim ke frontend.

```
func (h *IncomingHandler) Create(c *fiber.Ctx) error {
  var req requests.IncomingRequest
  if err := c.BodyParser(&req); err != nil {
    return c.Status(fiber.StatusBadRequest).JSON(responses.
   ErrorResponse{
```

```
      Error: "invalid request body",
    })
  }

  userID := c.Locals("user_id").(uint)
  if userID == 0 {
    return c.Status(fiber.StatusUnauthorized).JSON(responses.
    ErrorResponse{
      Error: "unauthorized",
    })
  }
  inc, err := h.service.Create(req, &userID)
  if err != nil {
    return c.Status(http.StatusInternalServerError).JSON(responses
    .ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusCreated).JSON(responses.
    MessageResponse{
    Message: "Incoming created successfully",
    Data:    inc,
  })
}
```

Service menangani seluruh logika bisnis: memvalidasi keberadaan PO, membuat data incoming, menyimpan optional images jika ada, mengubah posisi PO, dan melakukan commit transaksi database. Jika terjadi kesalahan di tengah proses, transaksi di-rollback.

```
func (s *incomingService) Create(req requests.IncomingRequest,
    createdBy *uint) (responses.IncomingResponse, error) {
  tx := s.db.Begin()

  po, err := s.poRepo.GetByUUID(req.POUUID)
  if err != nil {
    tx.Rollback()
    return responses.IncomingResponse{}, errors.New("po not found"
    )
  }

  incoming := models.Incoming{
    POID:      &po.ID,
```

```
    CheckBy:   createdBy,
    Status:    "pass",
    Notes:     req.Notes,
    CreatedBy: createdBy,
  }

  if err := tx.Create(&incoming).Error; err != nil {
    tx.Rollback()
    return responses.IncomingResponse{}, err
  }

  if len(req.Images) > 0 {
    var images []models.IncomingImage
    for _, img := range req.Images {
      images = append(images, models.IncomingImage{
        ImagePath:  img.ImagePath,
        IncomingID: incoming.ID,
        CreatedBy:  createdBy,
      })
    }

    if err := s.imageRepo.CreateBulkTx(tx, images); err != nil {
      tx.Rollback()
      return responses.IncomingResponse{}, err
    }
  }

  if err := s.poRepo.UpdatePositionTx(tx, po.UUID, 2, *createdBy);
    err != nil {
    return responses.IncomingResponse{}, err
  }

  if err := tx.Commit().Error; err != nil {
    return responses.IncomingResponse{}, err
  }

  created, _ := s.repo.GetByUUID(incoming.UUID)
  return mapToIncomingResponse(*created), nil
}
```

Repository berfokus menyimpan data incoming ke database menggunakan GORM. Fungsi ini hanya menangani transaksi database tanpa logika bisnis.

```
func (r *incomingRepository) Create(incoming *models.Incoming)
```

```
    error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  return r.db.WithContext(ctx).Create(incoming).Error
}
```

N Get All Incoming

Handler berfungsi menerima request GET dan memanggil service. Jika terjadi error saat pengambilan data, handler akan mengembalikan status internal server error, sebaliknya akan mengirim data incoming beserta pesan sukses.

```
func (h *IncomingHandler) GetAll(c *fiber.Ctx) error {
  incomings, err := h.service.GetAll()
  if err != nil {
    return c.Status(http.StatusInternalServerError).JSON(responses
    .ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "Incomings retrieved successfully",
    Data:    incomings,
  })
}
```

Service memanggil repository untuk mendapatkan seluruh data incoming, kemudian memetakan setiap record ke bentuk response yang telah ditentukan agar sesuai dengan DTO.

```
func (s *incomingService) GetAll() ([]responses.IncomingResponse,
    error) {
  incomings, err := s.repo.GetAll()
  if err != nil {
    return nil, err
  }

  var res []responses.IncomingResponse
  for _, inc := range incomings {
    res = append(res, mapToIncomingResponse(inc))
```

```
  }
  return res, nil
}
```

Repository bertugas mengeksekusi query untuk mengambil seluruh data incoming dari database. Fungsi ini menggunakan query dasar yang sudah termasuk relasi terkait jika diperlukan.

```
func (r *incomingRepository) GetAll() ([]models.Incoming, error) {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  var incomings []models.Incoming
  err := r.baseQuery(ctx).Find(&incomings).Error
  return incomings, err
}
```

O Get Incoming By UUID

Handler menerima request GET dengan parameter UUID, lalu memanggil service untuk mengambil data incoming. Jika terjadi error, handler merespon dengan error not found.

```
func (h *IncomingHandler) GetByUUID(c *fiber.Ctx) error {
  uuid := c.Params("uuid")
  inc, err := h.service.GetByUUID(uuid)
  if err != nil {
    return c.Status(http.StatusNotFound).JSON(responses.
    ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "Incoming retrieved successfully",
    Data:    inc,
  })
}
```

Service memanggil repository untuk mengambil data berdasarkan UUID. Setelah data didapat, service memetakan data tersebut ke bentuk response sesuai DTO.

```
func (s *incomingService) GetByUUID(uuid string) (responses.
    IncomingResponse, error) {
  inc, err := s.repo.GetByUUID(uuid)
  if err != nil {
    return responses.IncomingResponse{}, err
  }
  return mapToIncomingResponse(*inc), nil
}
```

Repository mengeksekusi query untuk mencari incoming berdasarkan UUID. Jika record tidak ditemukan, repository mengembalikan error dengan pesan ”incoming not found”.

```
func (r *incomingRepository) GetByUUID(uuid string) (*models.
    Incoming, error) {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  var incoming models.Incoming
  err := r.baseQuery(ctx).
    Where("uuid = ?", uuid).
    First(&incoming).Error

  if errors.Is(err, gorm.ErrRecordNotFound) {
    return nil, errors.New("incoming not found")
  }

  return &incoming, err
}
```

P Update Incoming

Handler menerima parameter UUID dan body request. Selanjutnya, handler memanggil service Update dan merespon hasilnya ke frontend. Error seperti body invalid atau user tidak authorized langsung dikembalikan dari handler.

```
func (h *IncomingHandler) Update(c *fiber.Ctx) error {
  uuid := c.Params("uuid")

  var req requests.IncomingRequest
  if err := c.BodyParser(&req); err != nil {
    return c.Status(fiber.StatusBadRequest).JSON(responses.
    ErrorResponse{
```

```
      Error: "invalid request body",
    })
  }

  userID := c.Locals("user_id").(uint)
  if userID == 0 {
    return c.Status(fiber.StatusUnauthorized).JSON(responses.
    ErrorResponse{
      Error: "unauthorized",
    })
  }
  inc, err := h.service.Update(uuid, req, &userID)
  if err != nil {
    return c.Status(http.StatusNotFound).JSON(responses.
    ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "Incoming updated successfully",
    Data:    inc,
  })
}
```

Service melakukan transaksi database (tx.Begin()) untuk memastikan integritas data. Service memvalidasi apakah incoming dan PO terkait ada, memperbarui field utama, menghapus image lama, menambahkan image baru, dan akhirnya commit transaksi. Jika terjadi error di tengah proses, transaksi di-rollback.

```
func (s *incomingService) Update(uuid string, req requests.
    IncomingRequest, updatedBy *uint) (responses.IncomingResponse,
    error) {
  tx := s.db.Begin()
  if tx.Error != nil {
    return responses.IncomingResponse{}, tx.Error
  }

  defer func() {
    if r := recover(); r != nil {
      tx.Rollback()
    }
  }()

```

```
13  existing, err := s.repo.GetByUUIDTx(tx, uuid)
14  if err != nil {
15    tx.Rollback()
16    return responses.IncomingResponse{}, errors.New("incoming not
     found")
17  }
18
19  po, err := s.poRepo.GetByUUID(req.POUUID)
20  if err != nil {
21    tx.Rollback()
22    return responses.IncomingResponse{}, errors.New("po not found"
     )
23  }
24
25  existing.POID = &po.ID
26  existing.CheckBy = updatedBy
27  existing.Notes = req.Notes
28  existing.UpdatedBy = updatedBy
29
30  if err := tx.Save(&existing).Error; err != nil {
31    tx.Rollback()
32    return responses.IncomingResponse{}, err
33  }
34
35  if err := s.imageRepo.DeleteByIncomingIDTx(tx, existing.ID); err
     != nil {
36    tx.Rollback()
37    return responses.IncomingResponse{}, err
38  }
39
40  if len(req.Images) > 0 {
41    var newImages []models.IncomingImage
42    for _, img := range req.Images {
43      newImages = append(newImages, models.IncomingImage{
44        ImagePath:  img.ImagePath,
45        IncomingID: existing.ID,
46        CreatedBy:  updatedBy,
47      })
48    }
49
50    if err := s.imageRepo.CreateBulkTx(tx, newImages); err != nil
     {
51      tx.Rollback()
```

```
      return responses.IncomingResponse{}, err
    }
  }

  if err := tx.Commit().Error; err != nil {
    return responses.IncomingResponse{}, err
  }

  updated, _ := s.repo.GetByUUID(existing.UUID)
  return mapToIncomingResponse(*updated), nil
}
```

Repository melakukan eksekusi update di database dengan menggunakan UUID sebagai filter. Semua perubahan field incoming dikirim ke database melalui method Updates.

```
func (r *incomingRepository) Update(incoming *models.Incoming)
    error {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  return r.db.WithContext(ctx).
    Model(&models.Incoming{}).
    Where("uuid = ?", incoming.UUID).
    Updates(incoming).Error
}
```

Q Delete Incoming

Fungsi delete incoming digunakan untuk menghapus data *existing* berdasarkan UUID. *Handler* menerima *request* dari *frontend*, memanggil *service*, lalu *service* meneruskan ke *repository* untuk melakukan eksekusi *delete* di *database*.

Handler menerima parameter UUID dan langsung memanggil *service* Delete. Jika UUID tidak ditemukan atau terjadi *error*, handler mengembalikan respons *error*.

```
func (h *IncomingHandler) Delete(c *fiber.Ctx) error {
  uuid := c.Params("uuid")
  if err := h.service.Delete(uuid); err != nil {
    return c.Status(http.StatusNotFound).JSON(responses.
    ErrorResponse{
```

```
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.
   MessageOnlyResponse{
    Message: "Incoming deleted successfully",
  })
}
```

Service bertugas meneruskan UUID ke *repository* untuk melakukan penghapusan. Tidak ada logika tambahan selain validasi sederhana dari *repository*.

```
func (s *incomingService) Delete(uuid string) error {
  return s.repo.Delete(uuid)
}
```

Repository melakukan *query delete* di *database* menggunakan UUID sebagai filter. Jika UUID tidak ditemukan, *repository* mengembalikan *error* ”*incoming not found*”.

```
func (r *incomingRepository) Delete(uuid string) error {
  ctx, cancel := context.WithTimeout(context.Background(),
   dbTimeout)
  defer cancel()

  result := r.db.WithContext(ctx).
    Where("uuid = ?", uuid).
    Delete(&models.Incoming{})

  if result.RowsAffected == 0 {
    return errors.New("incoming not found")
  }

  return result.Error
}
```


R Get HU By UUID

Handler membaca parameter UUID dari URL, memanggil service GetByUUID, dan mengembalikan response ke frontend. Jika UUID tidak ditemukan, handler mengembalikan error 404 Not Found.

```
func (h *HUHandler) GetByUUID(c *fiber.Ctx) error {
```

```
  uuid := c.Params("uuid")
  hu, err := h.service.GetByUUID(uuid)
  if err != nil {
    return c.Status(http.StatusNotFound).JSON(responses.
    ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(http.StatusOK).JSON(responses.MessageResponse{
    Message: "HU retrieved successfully",
    Data:    hu,
  })
}
```

Service memanggil repository untuk mengambil HU berdasarkan UUID, lalu melakukan mapping ke response DTO (HUResponse) melalui helper mapHUToResponse.

```
func (s *huService) GetByUUID(uuid string) (responses.HUResponse,
    error) {
  h, err := s.repo.GetByUUID(uuid)
  if err != nil {
    return responses.HUResponse{}, err
  }
  return mapHUToResponse(h, s.userRepo, s.poRepo)
}
```

Repository mengeksekusi query database menggunakan GORM. Selain memfilter berdasarkan UUID, repository juga melakukan Preload untuk memuat relasi Details dan POs. Jika record tidak ditemukan, akan mengembalikan error khusus.

```
func (r *huRepository) GetByUUID(uuid string) (models.HU, error) {
  var hu models.HU
  if err := r.db.
    Preload("Details").
    Preload("POs").
    Where("uuid = ?", uuid).
    First(&hu).Error; err != nil {

    if errors.Is(err, gorm.ErrRecordNotFound) {
      return hu, errors.New("HU not found")
    }
```

```
    return hu, err
  }
  return hu, nil
}
```

S Get PO By PO Number

Handler memeriksa apakah `po_number` disediakan, memanggil service GetByPONumber, dan mengembalikan response atau error yang sesuai.

```
func (h *POHandler) GetByPONumber(c *fiber.Ctx) error {
  poNumber := c.Params("po_number")
  if poNumber == "" {
    return c.Status(fiber.StatusBadRequest).JSON(responses.
     ErrorResponse{
      Error: "PO number is required",
    })
  }

  po, err := h.service.GetByPONumber(poNumber)
  if err != nil {
    status := fiber.StatusNotFound
    if !strings.Contains(err.Error(), "not found") {
      status = fiber.StatusInternalServerError
    }
    return c.Status(status).JSON(responses.ErrorResponse{
      Error: err.Error(),
    })
  }

  return c.Status(fiber.StatusOK).JSON(responses.MessageResponse{
    Message: "PO retrieved successfully",
    Data:    po,
  })
}
```

Service memanggil repository untuk mencari PO berdasarkan `po_number`. Setelah data diperoleh, service memetakan model PO ke response DTO (POResponse) menggunakan helper mapToPOResponse.

```
func (s *poService) GetByPONumber(poNumber string) (responses.
    POResponse, error) {
  po, err := s.repo.GetByPONumber(poNumber)
```

```
  if err != nil {
    return responses.POResponse{}, err
  }

  return mapToPOResponse(po), nil
}
```

Repository mengeksekusi query ke database dengan filter `po_number`. Jika PO tidak ditemukan, akan mengembalikan error khusus ”PO not found”.

```
func (r *poRepository) GetByPONumber(poNumber string) (models.PO,
    error) {
  ctx, cancel := context.WithTimeout(context.Background(),
    dbTimeout)
  defer cancel()

  var po models.PO
  err := r.baseQuery(ctx).Where("po_number = ?", poNumber).First(&
    po).Error

  if err != nil {
    if errors.Is(err, gorm.ErrRecordNotFound) {
      return po, errors.New("PO not found")
    }
    return po, err
  }
  return po, nil
}
```

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Dalam proses pengembangan sistem, berbagai kendala sering muncul baik dari sisi teknis maupun non-teknis. Bagian ini akan menjelaskan kendala-kendala yang dihadapi selama implementasi dan langkah-langkah solusi yang diterapkan untuk mengatasinya.

1. Perubahan flow dari klien yang sering terjadi, sehingga beberapa modul dan fitur harus dirombak berkali-kali dan mempengaruhi progres pengembangan.

2. Kurangnya konsistensi antar modul, yang menimbulkan kebingungan dalam implementasi karena perbedaan pendekatan atau standar pengembangan antar

file modul.

3.4.2 Solusi

Untuk mengatasi kendala-kendala tersebut, beberapa langkah solusi diterapkan agar proses pengembangan tetap efisien dan konsisten.

1. Mengadakan *daily meeting* dengan tim dan pihak klien untuk memastikan setiap perubahan yang diminta sudah jelas, tercatat, dan disepakati sebelum diterapkan pada sistem.
2. Menjaga konsistensi antar modul dengan melakukan koordinasi rutin bersama supervisor, serta meminta supervisor melakukan pengecekan setiap kali terjadi perubahan pada modul sebelum diintegrasikan ke sistem utama.