BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama kerja praktek magang di PT Cranium Royal Aditama, segala bentuk penugasan dilakukan di dalam divisi *IT and Software Solutions* sebagai *Full Stack Developer* pada proyek pengembangan *enterprise resource planning software* milik perusahaan. Fokus utama dalam kerja praktek magang ini terdapat pada perbaikan dan pengembangan fitur-fitur yang diperlukan dalam *software* ERP yang ditawarkan Cranium kepada perusahaan klien.

Dalam pelaksanaan kerja praktek magang, terdapat beberapa mentor dan supervisor yang memantau, membantu, serta mengarahkan dalam pelaksanaan pekerjaan, serta mengevaluasi dan memberikan masukan terhadap hasil pekerjaan. Selain mentor dan supervisor, terdapat juga anggota lain dalam tim pengembangan ERP, masing-masing dengan *task* yang berbeda-beda dalam pengembangan ERP.

Kedudukan dalam divisi *IT and Software Solutions* Cranium dapat dijabarkan sebagai berikut:

1. Supervisor: Supervisor berperan penting dalam pelaksanaan kerja magang karena bertanggung jawab mengawasi dan mengarahkan kinerja *developer*, baik dari segi pengerjaan tugas, ketepatan waktu, serta kesesuaian hasil kerja dengan kriteria perusahaan. Selain itu, supervisor juga berperan penting dalam mengajarkan sistem kerja di perusahaan dan ketentuan yang harus diikuti mahasiswa magang di Cranium.

2. Mentor: Sebagai *full stack developer*, terdapat dua mentor yaitu mentor di bidang *backend* ERP dan di bidang *frontend* ERP. Keduanya berperan dalam mengajarkan sistem dan alur pengerjaan baik frontend maupun backend pada ERP Cranium, mulai dari *setup* lingkungan kerja hingga alur koordinasi kerja antar *developer* melalui Github.

3. Mahasiswa Magang: Mahasiswa bertanggung jawab mengerjakan dan menyelesaikan semua tugas yang diberikan oleh supervisor sesuai dengan tenggat waktu dan kriteria yang diberikan. Selama kerja magang, tugas mahasiswa magang sebagai *full stack developer* adalah untuk mengembangkan fitur-fitur pada modul master ERP Cranium.

Selama kerja magang, ada beberapa peraturan kerja dan prosedur yang sudah ditentukan oleh perusahaan dalam pengembangan proyek ERP, untuk memastikan tugas dapat dikerjakan secara efektif dan efisien oleh semua orang yang terlibat di dalamnya. Alur atau *flow* kerja yang diterapkan dalam proyek ERP Cranium dapat dijelaskan sebagai berikut:

1. Distribusi Tugas
    (a) Distribusi tugas dilakukan ketika *developer* sudah menyelesaikan tugas sebelumnya.
    (b) *Developer* akan diberikan tugas baru apabila tugas sebelumnya sudah selesai hingga tahap di-*merge* dengan *branch* utama pada repository proyek.
    (c) Tugas akan diberikan dan di-*brief* secara langsung oleh supervisor kepada *developer*, dan dikomunikasikan secara pribadi maupun dalam meeting dengan keseluruhan tim ERP Cranium.
    (d) Supervisor menjelaskan kebutuhan fitur yang akan dibuat oleh *developer*, baik secara lisan atau dengan menggunakan bantuan *flowchart*, tabel rancangan, maupun rancangan desain pada Figma.
    (e) Tugas yang diberikan dapat berbentuk pengerjaan tampilan pada frontend, pengerjaan fitur pada backend, maupun perbaikan *bug* atau perubahan fitur pada frontend maupun backend.
    (f) Setiap tugas dikerjakan pada *branch* yang berbeda dalam *repository*. Hal ini dilakukan untuk mempermudah penggabungan hasil kerja antar *developer* dan mengantisipasi terjadinya *conflict* antar hasil kerja *developer*.
2. Pengerjaan Tugas
    (a) Tugas langsung dikerjakan setelah *developer* mendapatkan penjelasan mengenai fitur yang perlu dikerjakannya. Tugas harus dikerjakan sesuai dengan arahan supervisor dalam tenggat waktu yang sudah ditentukan.
    (b) Mahasiswa magang diminta untuk mengerjakan tugasnya secara maksimal, namun apabila mengalami kendala atau terkena *blocker*, maka supervisor dan mentor dapat membantu mencari solusi terhadap kendala yang ditemukan.

3. Konsultasi dan Revisi

   (a) Setelah setiap tahap tugas selesai dikerjakan, mahasiswa magang melakukan konsultasi atau asistensi dengan mentor atau supervisor, untuk mendapatkan *feedback* terhadap hasil kerja.

   (b) Mahasiswa juga dapat bertanya atau berkonsultasi dengan mentor dan supervisor di luar keperluan asistensi dan peninjauan hasil kerja.

   (c) Apabila terdapat revisi atau masukan terhadap hasil kerja, maka mahasiswa dapat langsung memperbaikinya. Perbaikan diharapkan dikerjakan tetap dalam tenggat waktu yang ditentukan, namun apabila tidak memungkinkan maka dapat diberikan tambahan waktu pengerjaan.

   (d) Apabila revisi selesai dikerjakan, mahasiswa perlu melakukan konsultasi lagi dengan mentor atau supervisor untuk memastikan kesesuaian hasil revisi.

4. Pengujian Hasil Kerja

   (a) Hasil kerja mahasiswa perlu melalui pengujian atau pengecekan. Pengujian dilakukan dengan dua metode, yaitu dengan membuat *unit test* dan *contract test* pada backend maupun frontend, serta dengan melakukan pengujian fitur secara manual.

   (b) Pengujian dilakukan dengan menggunakan fitur *testing* pada Springboot dan NextJS, dengan *test case* yang dibuat oleh mahasiswa dan telah disesuaikan dengan fitur yang ada dalam ERP Cranium.

   (c) Pengujian manual dilakukan oleh mahasiswa dan oleh mentor atau supervisor setelah fitur diunggah ke *repository*.

5. Penggabungan Hasil Kerja (*Pull Request*)

   (a) Hasil kerja mahasiswa yang sudah lulus tahap pengujian akan digabungkan ke dalam *branch* utama pada *repository*.

   (b) Hasil *merging* melewati pengujian lagi secara manual dengan menjalankan aplikasi yang sudah digabungkan dan memastikan tidak ada *error* maupun *bug* pada aplikasi.

6. Finalisasi Hasil Kerja

(a) Hasil kerja dianggap final apabila sudah digabungkan dalam *branch* utama *repository*. Setelah digabungkan, *branch* yang digunakan untuk pengembangan fitur dihapus dari *repository* dan *developer* membuat *branch* baru untuk pengerjaan tugas berikutnya.

Alur komunikasi dan koordinasi dalam kerja magang yang dilaksanakan di Cranium dapat dijabarkan sebagai berikut:

1. Komunikasi
   (a) Komunikasi harian dalam proyek ERP dilakukan dengan menggunakan aplikasi *Whatsapp* untuk sarana komunikasi utama dalam tim ERP, serta dengan menggunakan aplikasi *Discord* sebagai sarana komunikasi pembantu. Umumnya *Discord* digunakan untuk diskusi dan konsultasi dalam tim, khususnya ketika sedang WFH.
   (b) Komunikasi harian ketika WFO dilakukan secara langsung tatap muka, baik dalam bentuk rapat maupun komunikasi pribadi.
2. Koordinasi
   (a) Koordinasi pengerjaan tugas dalam proyek ERP dilakukan dengan menggunakan tabel *roadmap* yang dibuat di Figma untuk melacak progres pengerjaan tugas dalam proyek ERP.
   (b) Mahasiswa dapat memperbarui secara mandiri progres kerjanya pada tabel *roadmap* di Figma, sama seperti anggota tim ERP Cranium yang lain.
   (c) Mahasiswa juga diberikan Google Sheets kolaboratif untuk melacak hasil kerja harian serta melihat hasil kerja harian satu sama lain.
3. Kolaborasi
   (a) Kolaborasi pengerjaan tugas dalam proyek ERP dilakukan menggunakan Github sebagai *repository*.

3.2 Tugas yang Dilakukan

Selama kerja magang di PT Cranium Royal Aditama, mahasiswa ditempatkan pada proyek ERP Cranium dengan tugas untuk mengembangkan

dan memperbaiki fitur-fitur dalam ERP sesuai dengan kebutuhan perusahaan dan konsumen.

Pengembangan ERP Cranium pada bagian *backend* menggunakan Java Springboot serta pada bagian *frontend* menggunakan NextJS dengan Typescript, dengan arsitektur aplikasi modular monolitik. Arsitektur modular monolitik merupakan pola pengembangan perangkat lunak yang memadukan kemudahan *deployment* dari sistem monolitik dengan prinsip modularitas yang biasanya ditemukan pada arsitektur *microservices*. Dalam pendekatan ini, sistem dibagi menjadi sejumlah modul yang saling terpisah dan memiliki batas tanggung jawab serta ketergantungan (*dependency*) yang terdefinisi dengan jelas. Setiap modul dirancang untuk menangani fungsi tertentu, dilengkapi dengan antarmuka yang eksplisit, serta memungkinkan proses pengembangan dan pengujian dilakukan secara terpisah. [7, 8]

Jenis-jenis tugas yang diberikan beragam, seperti perbaikan *bugs* minor, penambahan fitur pada *frontend* atau *backend*, penambahan sistem CRUD baru baik untuk *frontend* maupun *backend*, pembuatan *unit test*, dan perbaikan/penyesuaian fitur. Tugas-tugas dikerjakan sesuai dengan arahan supervisor dan menyesuaikan dengan kebutuhan perusahaan.

Selama menjalankan kerja magang di Cranium, mahasiswa magang ditempatkan pada proyek ERP Cranium, dan diberikan tugas untuk mengembangkan fitur-fitur pada ERP, sesuai dengan kebutuhan perusahaan. Tugas yang diberikan memiliki skala yang beragam dan mencakup berbagai aspek dari pengembangan ERP, mulai dari pengembangan tampilan ERP, pengembangan sistem backend ERP, pembuatan *unit test* pada frontend maupun backend, perbaikan *bugs* dan *error* yang ditemukan pada ERP, serta penyesuaian fitur pada ERP, mengikuti arahan dari supervisor dan menyesuaikan dengan kebutuhan perusahaan.

Terdapat beberapa *tools*, *software*, dan *framework* untuk mengerjakan tugas-tugas di Cranium. *Software* dan *framework* yang digunakan dalam pengembangan ERP Cranium serta penjelasannya adalah sebagai berikut:

1. IntelliJ digunakan untuk mengembangkan backend berbasis Java Springboot dari ERP. Integrasi IntelliJ dengan bahasa pemrograman Java mempermudah pengembangan backend ERP. Pengembangan yang dilakukan dengan menggunakan IntelliJ juga mencakup pengembangan *unit/contract test* untuk backend.

2. Java Springboot digunakan sebagai *framework* untuk backend dari ERP Cranium.
3. Visual Studio Code digunakan untuk mengembangkan frontend berbasis NextJS dari ERP. VSCode digunakan untuk mengembangkan keseluruhan frontend serta *unit test* yang dibutuhkan untuk frontend.
4. NextJS dengan TypeScript digunakan sebagai *framework* untuk mengembangkan frontend ERP Cranium. Frontend ERP Cranium dikembangkan dalam bentuk *web application*.
5. PostgreSQL digunakan sebagai database dalam proyek ERP Cranium.
6. DBeaver digunakan sebagai *database tool* dalam pengerjaan ERP Cranium.
7. Postman adalah *tools* API yang digunakan untuk menguji API yang dibuat pada backend Cranium dan menyesuaikan frontend agar dapat menggunakan API terkait.
8. Git digunakan sebagai *version control* untuk melacak perubahan *file* pada proyek ERP.
9. GitHub digunakan sebagai *distributed version control* untuk proyek ERP Cranium.

3.3 Uraian Pelaksanaan Magang

3.3.1 Pelaksanaan Kerja Magang

Pelaksanaan kerja magang dapat dilihat pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	*Onboarding* perusahaan, instalasi *software*, *library* dan *tools* dilakukan. Setup dasar Java Springboot, NextJS, PostgreSQL dilakukan, serta aturan bekerja *developer* di Cranium dipelajari.
	Lanjut di halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu Ke -	Pekerjaan yang dilakukan
2	Pembelajaran Springboot dilanjutkan dan *backend training* ERP Cranium dimulai. Pembelajaran difokuskan pada konsep *Controller-Service-Repository*, *Data Transfer Object*, dan *Validation* serta pembuatan sistem CRUD sederhana pada *backend*.
3	Pembelajaran *backend* dilanjutkan dan struktur *backend* proyek ERP Cranium mulai dipelajari. Pembelajaran difokuskan pada konsep *message* dan *authorization*.
4	Pembelajaran *backend* proyek ERP Cranium dilanjutkan. Pembelajaran difokuskan pada konsep *test suites* (*contract test, unit test*). *Unit test* dan *contract test* sederhana untuk *backend* dibuat.
5	Hasil *backend training* diulas kembali dan dievaluasi supervisor. Pembelajaran *frontend* ERP dipersiapkan.
6	Pembelajaran *frontend* ERP Cranium (ERP *frontend training*) menggunakan NextJS dimulai.
7	Pembelajaran *frontend* ERP Cranium dilanjutkan, difokuskan pada pembuatan halaman untuk *create* dan *read* (POST dan GET), kemudian disambungkan dengan *backend* melalui *endpoint* yang sudah dibuat.
8	Pembelajaran *frontend* ERP Cranium dilanjutkan, pembelajaran difokuskan pada pembuatan halaman untuk *update* dan fungsionalitas *delete*. *Unit test* untuk *frontend* juga dibuat.
9	*Training backend* dan *frontend* diulas kembali serta evaluasi oleh supervisor.
10	Persiapan pengembangan ERP Cranium dilakukan dengan *clone* dari *repository* Github, kemudian struktur dan alur aplikasi dipelajari.
11	Tugas pada modul *Master* submodul *Salesman* dikerjakan. Perubahan tampilan *field* pada *frontend* dan perbaikan pada *backend* dilakukan.

Lanjut di halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

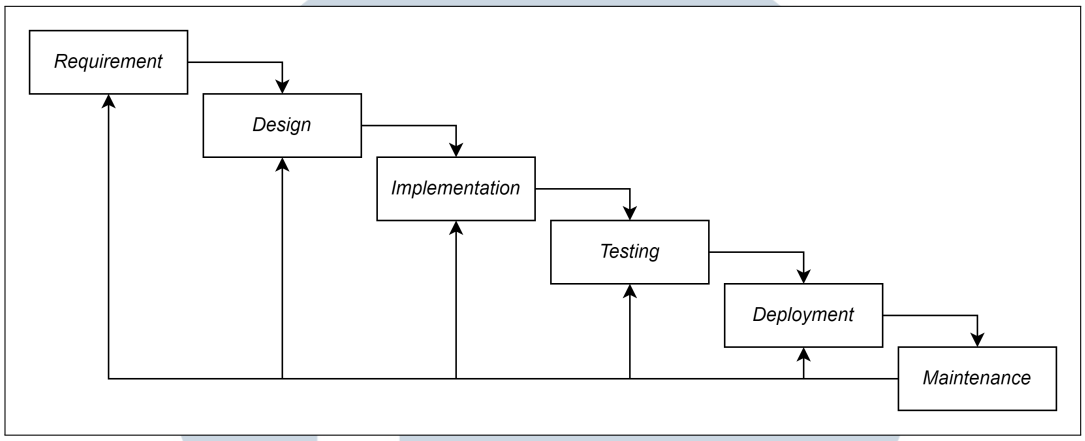
Minggu Ke -	Pekerjaan yang dilakukan
12	Pengerjaan perbaikan pada submodul *Salesman* dimulai. Pengerjaan difokuskan pada revisi yang masuk melalui *comment* dalam *Pull Request*.
13	Revisi terakhir untuk *Salesman* dikerjakan (menghilangkan kolom data pada *Salesman*). Tugas pada submodul *Supplier* untuk penambahan fitur *generate CoA* mulai dikerjakan.
14	Pengerjaan submodul *Supplier* dilanjutkan, difokuskan pada penyesuaian *unit test* dan revisi pekerjaan berdasarkan *comment* dalam *Pull Request*.
15	CRUD pada *backend* untuk submodul baru yaitu *Plugin Group* dikerjakan.
16	*Unit test* dan *contract test* pada *backend* untuk submodul *Plugin Group* dikerjakan.
17	Penambahan fitur VAT (*Value Added Tax*) pada submodul *Quotation* dalam modul *Selling* dikerjakan. VAT merupakan salah satu submodul dalam *Master* yang berfungsi merepresentasikan nilai persentase PPn dalam perhitungan. Pengerjaan difokuskan pada pembuatan *webclient* untuk komunikasi antara modul Selling dengan Master.
18	Pengerjaan submodul *Quotation* dilanjutkan, difokuskan pada penambahan kolom, modifikasi perhitungan, dan penyesuaian *unit test* serta *contract test*.
19	Penambahan fitur *fulfillment* pada submodul *Delivery Order* dalam modul *Selling* dikerjakan.
20	Penambahan komponen VAT pada perhitungan submodul *Order Return* dalam modul *Selling* dikerjakan. Fitur *fulfillment* pada submodul *Delivery Order Return* dikerjakan.

3.3.2 Konsep *Sofware Development Life Cycle* ERP Cranium

Software Development Life Cycle (SDLC), atau Siklus Hidup Pengembangan Perangkat Lunak, adalah sebuah kerangka kerja struktural yang menggambarkan tahapan-tahapan yang terlibat dalam pengembangan,

pengelolaan, dan pemeliharaan suatu *software*. [9]

Tahapan SDLC yang digunakan dalam pengembangan ERP Cranium adalah sebagai berikut:



Gambar 3.1. Alur SDLC ERP Cranium

Pengembangan *software* ERP Cranium menggunakan pendekatan Model Air Terjun atau *Waterfall Model* seperti pada gambar 3.1 [10]. *Waterfall Model* adalah metodologi pengembangan perangkat lunak sekuensial dan linear di mana kemajuan mengalir secara dominan ke satu arah melalui berbagai fase SDLC. Model ini mewajibkan setiap fase diselesaikan sepenuhnya sebelum melanjutkan ke fase pengembangan berikutnya. [11, 10]

Penjelasan masing-masing tahapan pada *waterfall* yang digunakan proyek ERP Cranium adalah sebagai berikut:

- *Requirement*

  Tahap ini berfokus pada identifikasi kebutuhan sistem, mencakup spesifikasi perangkat lunak, analisis kebutuhan/persyaratan bisnis, dan aspek fungsional sistem ERP.

- *Design*

  Berdasarkan kebutuhan yang telah ditetapkan, tahap ini fokus pada perancangan arsitektur sistem secara keseluruhan. Ini mencakup penentuan struktur komponen, interaksi antar komponen, dan detail teknis seperti skema *database* dan *interface*.

- *Implementation*

Pada tahap ini, kode program ditulis sesuai dengan desain yang telah disetujui. Pengembangan kode dibagi per modul, dengan fokus awal pada fungsionalitas dasar CRUD (*Create, Read, Update, Delete*). Integrasi proses bisnis antarmodul akan dilakukan pada fase pengembangan selanjutnya.

- *Testing*

  Setelah tahap implementasi, *software* melalui tahap pengujian untuk memastikan fungsi-fungsi berjalan sesuai desain yang dibuat dan untuk memastikan kualitas hasil implementasi, termasuk mengidentifikasi *bugs*.

- *Deployment*

  Software yang sudah lulus pengujian dirilis ke lingkungan produksi. Tahap ini mencakup instalasi *software* ke server dan aktivitas lainnya untuk memastikan aplikasi dapat berjalan dengan baik dan dapat digunakan oleh *user*.

- *Maintenance*

  Pada tahap ini, *software* yang sudah dikembangkan dan sudah melalui pengujian melalui tahap pemeliharaan yang meliputi perbaikan *bug*, optimalisasi, dan penambahan fitur baru untuk menyempurnakan *software* di masa mendatang.

3.3.3 Ruang Lingkup dan Konsep Pengembangan

Pengembangan yang dilakukan dalam ERP Cranium dapat dikategorikan menjadi hal-hal berikut:

1. Perbaikan *Bugs* Perbaikan *bugs* dilakukan pada modul Master dan Selling. Perbaikan dilakukan terhadap *bugs* yang sudah ditemukan dan dilaporkan oleh supervisor maupun terhadap *bugs* yang ditemukan ketika mengerjakan modul. Perbaikan dapat dilakukan di *frontend* maupun *backend*, dengan skala *bugs* yang berbeda-beda. Beberapa jenis *bugs* yang diperbaiki adalah:
   - Perbaikan tampilan data pada *web*. Terdapat beberapa tampilan pada *web* ERP yang memerlukan perbaikan, seperti pelabelan dan *translation* kolom atau *field* yang tidak tepat, bentuk data yang ditampilkan tidak sesuai, jumlah kolom data yang tidak sesuai, dan data yang tidak muncul.

- Perbaikan fitur pada *web*. Beberapa fitur pada *web* ERP membutuhkan perbaikan, di antaranya adalah *dropdown* yang tidak menampilkan data pilihan, kolom isian yang belum ada, *form* pembuatan atau perubahan data yang tidak dapat dikirim, dan *filtering* data yang belum diimplementasi.
- Perbaikan CRUD. Beberapa fitur CRUD tidak sesuai implementasinya atau tidak dapat dijalankan. Perbaikan CRUD yang dilakukan meliputi perbaikan *endpoint*, perbaikan metode CRUD pada *backend*, serta perbaikan transaksi *database*.
- Perbaikan struktur data *frontend* maupun *backend*. Terdapat beberapa data yang mengalami perbedaan antara data yang dikirimkan oleh *web* dan data yang dibutuhkan oleh *backend*

2. Pengembangan Submodul Baru

   Plugin Group merupakan salah satu submodul baru dalam modul *Master* yang dibutuhkan untuk fungsionalitas plugin pada ERP Cranium, bersama dengan *Plugin* dan *Plugin Install*. Pengembangan submodul baru yang dilakukan mencakup pembuatan CRUD untuk *Plugin Group* disertai dengan *endpoint, validation,* dan *unit test*. Integrasi dengan *frontend* dikerjakan secara terpisah untuk *Plugin Group*.

3. Penambahan Fitur Submodul

   Terdapat beberapa fitur baru yang ditambahkan ke submodul yang sudah ada. Salah satu penambahan fitur yang paling banyak dilakukan adalah penambahan *VAT* ke dalam perhitungan PPn modul *Selling* dan penambahan fitur *fulfillment* di beberapa submodul dalam *Selling*.

Beberapa konsep yang umum digunakan dalam pengembangan ERP Cranium adalah sebagai berikut:

1. *Data Transfer Object*, *Entity, & Mapper*
   Data Transfer Object atau DTO adalah objek sederhana yang dirancang khusus untuk membawa data antara proses atau *layer* yang berbeda dalam sebuah aplikasi. DTO tidak mengandung logika bisnis apapun dan hanya berisi properti/*field* dan metode *getter/setter* untuk data yang dibawanya. DTO bertujuan untuk mengurangi jumlah panggilan/*call* yang mahal antara *client* dan *server* maupun antar proses aplikasi, dengan

mengelompokkan data menjadi satu objek untuk dikirimkan dalam satu kali operasi. Keberadaan DTO juga memastikan data yang ditampilkan ke *client* hanya data yang dibutuhkan atau properti yang relevan saja, sehingga informasi sensitif terkait struktur database dan sejenisnya tidak akan terekspos ke *client*. [12, 13]

Entity atau entitas adalah POJO (*Plain Old Java Object*) yang merepresentasikan data dalam sebuah tabel database. *Entity* adalah representasi objek dari data yang persisten, dan mendefinisikan kolom data database menjadi bentuk yang bisa digunakan oleh aplikasi, serta mendefinisikan hubungan antar tabel dalam database. *Java Persistence API* (JPA, sekarang disebut *Jakarta Persistence*) digunakan dalam ERP Cranium untuk manajemen data relasional dalam bentuk *entity*. Keith, M., Schincariol, M., & Jensen, P. (2018). [14]

Mapper adalah komponen yang berfungsi untuk mengubah objek dari satu jenis ke jenis lainnya. Dalam ERP Cranium atau arsitektur perangkat lunak yang menggunakan DTO dan *entity*, *mapper* digunakan untuk mengubah DTO menjadi bentuk *entity* ataupun sebaliknya.

2. *Controller, Service, & Repository*

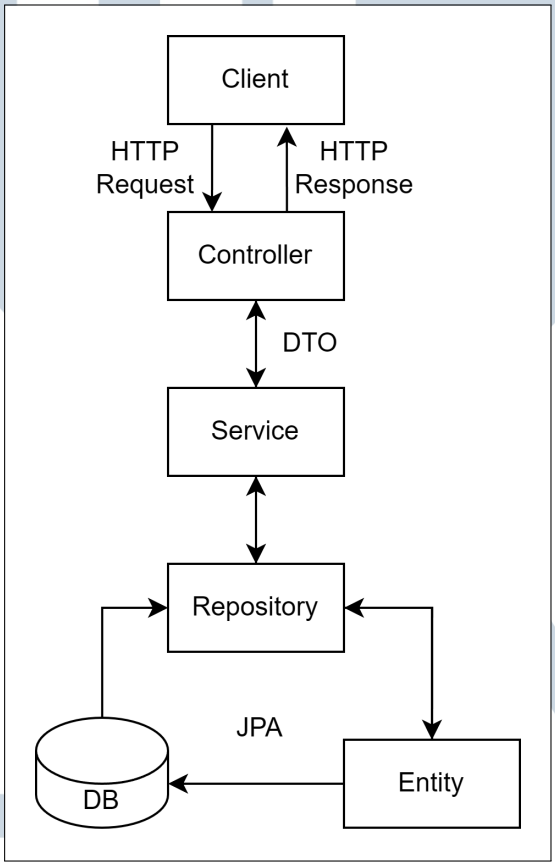
   Controller, *Service*, & *Repository* merupakan tiga komponen utama yang membentuk *layered architecture* dalam pengembangan aplikasi Spring Boot. Arsitektur ini memisahkan fungsi dalam aplikasi, sehingga kode lebih terstruktur, mudah diuji, dan mudah dipelihara.

   Controller merupakan lapisan presentasi atau *web layer* yang bertanggung jawab menangani *HTTP Request* dari *client* dan mengembalikan *HTTP Response*. *Controller* adalah titik masuk pertama untuk aplikasi eksternal dan tidak mengandung logika bisnis sama sekali, melainkan menghantarkan data ke *service*. Validasi data sederhana dapat dilakukan dalam *controller* untuk memastikan data yang dikirim ke *service* layak. *Controller* biasanya mengkomunikasikan data dalam bentuk DTO.

   Service merupakan *business logic layer* yang menangani semua logika, aturan bisnis, dan seluruh proses yang dibutuhkan untuk memenuhi permintaan/*request*. Lapisan *service* dapat diartikan sebagai "otak" aplikasi dan mengolah data-data yang diterima dari *controller* untuk diolah dan ditransformasi sehingga dapat digunakan sebagai *entity* oleh *repository*, dan

sebaliknya mengubah *entity* untuk dikirimkan lagi ke *controller* dalam bentuk DTO. *Service* juga melakukan mayoritas proses validasi data. *Mapper* banyak digunakan dalam *service*.

Repository bertanggung jawab dalam interaksi dengan database, menjadi jembatan antara aplikasi Java dengan database. *JPA Repository* digunakan dalam pengembangan ERP Cranium dan sudah menyediakan beberapa operasi CRUD dasar, namun *repository* juga dapat dikembangkan untuk melakukan CRUD yang lebih spesifik menggunakan *custom query*.



Gambar 3.2. Hubungan Controller-Service-Repository dalam arsitektur aplikasi ERP Cranium (Sumber: Dokumen internal)

Gambar 3.2 menjelaskan hubungan antara *controller, service, dan repository* serta alur datanya. Proses dimulai ketika *client* menggunakan *API* untuk melakukan *HTTP request*, mengirimkan suatu informasi atau data ke sistem. Setelah data tersebut dikirim, *controller* akan menerimanya dan memproses *request* sesuai dengan metode *HTTP* yang dikirimkan. Data kemudian diteruskan ke *service* dalam bentuk DTO. *Service* akan

menjalankan pengolahan data sesuai dengan permintaan dan mengubah DTO menjadi *entity*, kemudian meneruskan data ke *repository* yang berinteraksi dengan database. *Repository* melakukan operasi CRUD pada *entity* dengan menggunakan *JPA Repository* sebagai ORM.

3. *Unit Test & Contract Test*

   Unit test merupakan pengujian *software* yang berfungsi menguji komponen terkecil yang dapat diuji dalam sebuah aplikasi, biasanya berupa metode, prosedur, atau *class* tertentu. Tujuan dari *unit test* adalah untuk memastikan setiap unit kode secara individual atau terisolasi sesuai dengan spesifikasi yang dibutuhkan.

   Contract test adalah pengujian yang memastikan bahwa bagian-bagian dalam suatu aplikasi berkomunikasi dengan benar sesuai dengan "kontrak" yang ditentukan.

   Terdapat *unit test* dan *contract test* untuk *backend*, dan *unit test* untuk *frontend*. Pada *backend*, *unit test* digunakan untuk memastikan bahwa setiap metode dalam *service* berjalan sesuai dengan logika bisnis dan spesifikasi yang ditentukan, dengan menggunakan *library* JUnit, Mockito, dan Spring Boot Test. *Contract test* pada *backend* menguji komunikasi *controller* dan memastikan *controller* memenuhi permintaan *client* terkait dengan *endpoint* dan format responnya, dengan menggunakan *library* Spring Cloud Contract, MockMVC, Mockito, dan JUnit. Pada *frontend*, *unit test* memverifikasi bahwa komponen UI di-*render* dan me-*render* data dengan benar dan berperilaku sesuai ekspektasi. Pengujian pada *frontend* menggunakan *library* Jest dan React Testing Library.

3.3.4 Pengembangan Modul Master

A Submodul Salesman

Submodul *Salesman* atau penjual mendefinisikan struktur dan fungsi dari *Salesman*. Dalam proses bisnis ERP Cranium, *Salesman* digunakan untuk merepresentasikan seorang *sales* yang berperan dalam suatu proses penjualan. Suatu objek *Salesman* akan digunakan di beberapa modul dan submodul lainnya untuk menandakan *sales* yang menangani suatu transaksi.

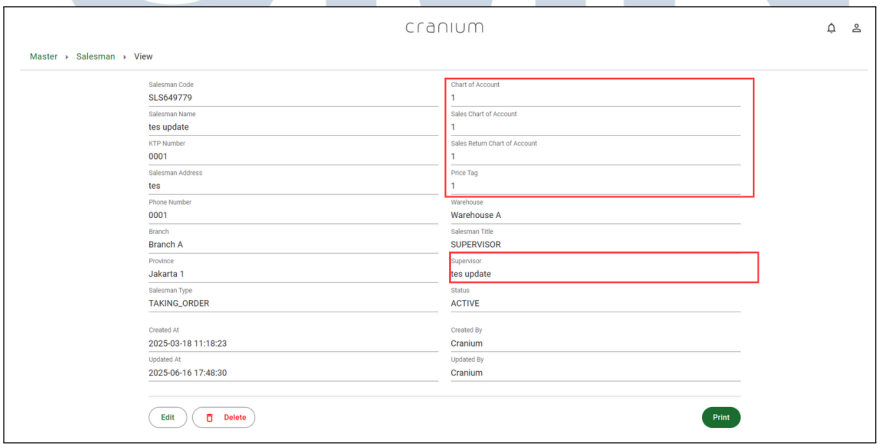
A.1 Analisis Masalah

Masalah yang ditemukan dalam *Salesman* setelah melalui proses *quality assurance* adalah sebagai berikut:

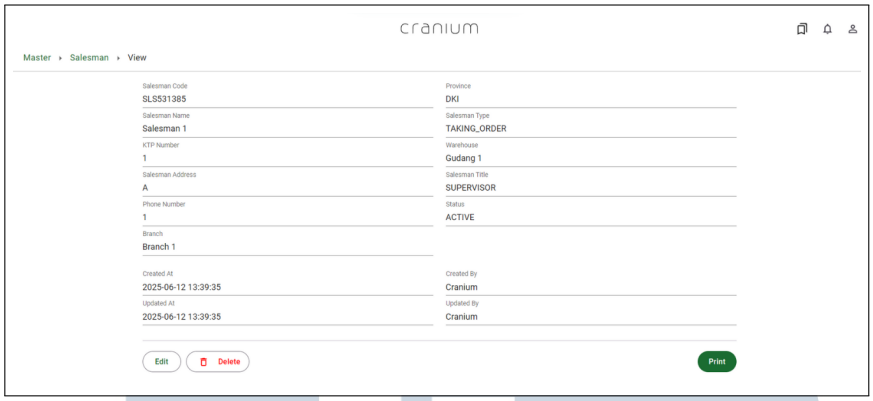
1. Kolom *Chart of Account* dan *Price Tag* pada tampilan data *Salesman* seharusnya tidak perlu ditampilkan, baik di form *create, update,* dan *detail view*.
2. Kolom *superior* sudah tidak dibutuhkan lagi, sehingga dihapus dari BE maupun FE.
3. Fitur pencarian data pada *Salesman* tidak dapat digunakan untuk kolom gudang dan provinsi.
4. Kolom *salesman title* tidak dapat di-*update*.

A.2 Solusi

Penghapusan kolom *CoA* dan *Price Tag* pada tampilan *view detail* hanya membutuhkan penghapusan kolom data yang dimaksud dari kode *frontend* serta penyesuaian pada *unit test*. Kolom *superior* sudah tidak dibutuhkan lagi dari proses bisnis, maka dihapus sepenuhnya dari *backend* maupun *frontend*. Pada *backend*, kolom tersebut dihapus dari *migration*, *entity*, DTO, *controller*, *service*, *mapper*, dan *specification* serta dihapus dari file-file *unit test* dan *contract test*. Pada *frontend*, *superior* dihapus dari *type*, file-file tampilan, dan file-file *unit test*. Gambar 3.3 menunjukkan tampilan awal halaman detail *Salesman* dan gambar 3.4 menunjukkan tampilan detail *Salesman* yang sudah diperbaiki.



Gambar 3.3. Tampilan awal halaman detail *Salesman*

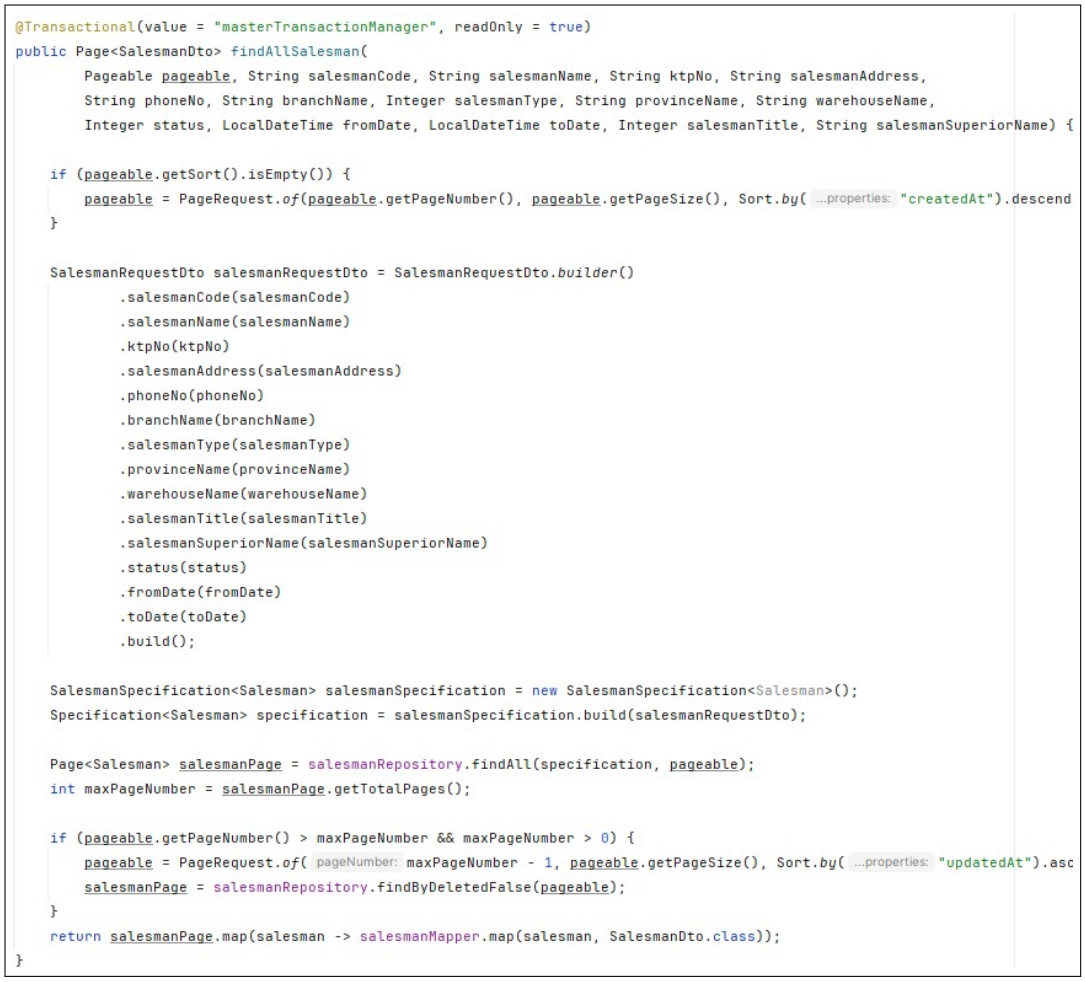


Gambar 3.4. Tampilan halaman detail *Salesman* yang sudah diperbaiki

Bug yang ditemukan pada fitur pencarian untuk kolom gudang dan provinsi disebabkan oleh ketidaksesuaian parameter antara *endpoint* yang disediakan *backend* dengan data yang dikirimkan oleh *frontend*. Perbaikan dilakukan pada *SalesmanController* di *backend* untuk menyesuaikan data yang diterima. Bentuk data yang digunakan untuk mengirimkan data dari *controller* ke *service* diubah dari menggunakan parameter/argumen individual menjadi *SalesmanRequestDTO* untuk mengurangi jumlah parameter yang dikirimkan ke *service*. Hal ini meningkatkan efisiensi kode dengan mengemas data yang dikirimkan menjadi satu DTO saja dibandingkan dengan menggunakan parameter individual untuk setiap kolom data, serta mengurangi *cluttered code*. Perubahan juga dilakukan di *SalesmanService* dengan mengubah parameter individual yang diterima menjadi bentuk DTO. Gambar 3.5 dan 3.6 menunjukkan potongan kode dalam *controller*dan *service Salesman* sebelum perbaikan, sementara potongan kode dalam3.1 dan 3.2 menunjukkan kode yang telah diperbaiki.

```
⚠ AmoryApk +1
@GetMapping(value = "/salesmans", headers = "X-Api-Version=1")
@ResponseStatus(value = HttpStatus.OK)
@IsMasterSalesmanRead
@LogExecutionTime
public Page<SalesmanDto> findAllSalesman(Pageable pageable,
                                         @RequestParam(value = "salesmanCode", required = false) String salesmanCode,
                                         @RequestParam(value = "salesmanName", required = false) String salesmanName,
                                         @RequestParam(value = "ktpNo", required = false) String ktpNo,
                                         @RequestParam(value = "salesmanAddress", required = false) String salesmanAddress,
                                         @RequestParam(value = "phoneNo", required = false) String phoneNo,
                                         @RequestParam(value = "branchName", required = false) String branchName,
                                         @RequestParam(value = "salesmanType", required = false) Integer salesmanType,
                                         @RequestParam(value = "warehouseName", required = false) String warehouseName,
                                         @RequestParam(value = "provinceName", required = false) String provinceName,
                                         @RequestParam(value = "status", required = false) Integer status,
                                         @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime fromDate,
                                         @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime toDate,
                                         @RequestParam(required = false, value = "salesmanTitle") Integer salesmanTitle,
                                         @RequestParam(required = false, value = "salesmanSuperiorName") String salesmanSuperiorName) throws DataNotFoundException {
    return salesmanService.findAllSalesman(pageable, salesmanCode, salesmanName, ktpNo, salesmanAddress, phoneNo,
            branchName, salesmanType, warehouseName, provinceName, status, fromDate, toDate, salesmanTitle, salesmanSuperiorName);
}
```

Gambar 3.5. Tangkapan layar potongan kode *SalesmanController* sebelum perbaikan



```
@Transactional(value = "masterTransactionManager", readOnly = true)
public Page<SalesmanDto> findAllSalesman(
        Pageable pageable, String salesmanCode, String salesmanName, String ktpNo, String salesmanAddress,
        String phoneNo, String branchName, Integer salesmanType, String provinceName, String warehouseName,
        Integer status, LocalDateTime fromDate, LocalDateTime toDate, Integer salesmanTitle, String salesmanSuperiorName) {

    if (pageable.getSort().isEmpty()) {
        pageable = PageRequest.of(pageable.getPageNumber(), pageable.getPageSize(), Sort.by( ...properties: "createdAt").descend
    }

    SalesmanRequestDto salesmanRequestDto = SalesmanRequestDto.builder()
            .salesmanCode(salesmanCode)
            .salesmanName(salesmanName)
            .ktpNo(ktpNo)
            .salesmanAddress(salesmanAddress)
            .phoneNo(phoneNo)
            .branchName(branchName)
            .salesmanType(salesmanType)
            .provinceName(provinceName)
            .warehouseName(warehouseName)
            .salesmanTitle(salesmanTitle)
            .salesmanSuperiorName(salesmanSuperiorName)
            .status(status)
            .fromDate(fromDate)
            .toDate(toDate)
            .build();

    SalesmanSpecification<Salesman> salesmanSpecification = new SalesmanSpecification<Salesman>();
    Specification<Salesman> specification = salesmanSpecification.build(salesmanRequestDto);

    Page<Salesman> salesmanPage = salesmanRepository.findAll(specification, pageable);
    int maxPageNumber = salesmanPage.getTotalPages();

    if (pageable.getPageNumber() > maxPageNumber && maxPageNumber > 0) {
        pageable = PageRequest.of( pageNumber: maxPageNumber - 1, pageable.getPageSize(), Sort.by( ...properties: "updatedAt").asc
        salesmanPage = salesmanRepository.findByDeletedFalse(pageable);
    }
    return salesmanPage.map(salesman -> salesmanMapper.map(salesman, SalesmanDto.class));
}
```

Gambar 3.6. Tangkapan layar potongan kode *SalesmanService* sebelum perbaikan

```
1  @GetMapping(value = "/salesmans", headers = "X-Api-Version=1")
2  @ResponseStatus(value = HttpStatus.OK)
3  @IsMasterSalesmanRead
4  @LogExecutionTime
5  public Page<SalesmanDto> findAllSalesman(Pageable pageable,
                    @RequestParam(value = "salesmanCode", required = false) String
       salesmanCode,
6  @RequestParam(value = "salesmanName", required = false) String salesmanName,
7  @RequestParam(value = "ktpNo", required = false) String ktpNo,
8  @RequestParam(value = "salesmanAddress", required = false) String salesmanAddress
       ,
9  @RequestParam(value = "phoneNo", required = false) String phoneNo,
10 @RequestParam(value = "branchName", required = false) String branchName,
11 @RequestParam(value = "salesmanType", required = false) Integer salesmanType,
12 @RequestParam(value = "warehouseName", required = false) String warehouseName,
13 @RequestParam(value = "provinceName", required = false) String provinceName,
14 @RequestParam(value = "status", required = false) Integer status,
15 @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.
       DATE_TIME) LocalDateTime fromDate,
16 @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.
```

```
        DATE_TIME) LocalDateTime toDate,
@RequestParam(required = false, value = "salesmanTitle") Integer salesmanTitle)
        throws DataNotFoundException {
    SalesmanRequestDto salesmanRequestDto = SalesmanRequestDto.builder()
            .salesmanCode(salesmanCode)
            .salesmanName(salesmanName)
            .ktpNo(ktpNo)
            .salesmanAddress(salesmanAddress)
            .phoneNo(phoneNo)
            .branchName(branchName)
            .salesmanType(salesmanType)
            .provinceName(provinceName)
            .warehouseName(warehouseName)
            .salesmanTitle(salesmanTitle)
            .status(status)
            .fromDate(fromDate)
            .toDate(toDate)
            .build();

    return salesmanService.findAllSalesman(pageable, salesmanRequestDto);
}
```

Kode 3.1: Endpoint untuk Find All Salesman

```
@Transactional(value = "masterTransactionManager", readOnly = true)
public Page<SalesmanDto> findAllSalesman(Pageable pageable, SalesmanRequestDto
    salesmanRequestDto) {

    if (pageable.getSort().isEmpty()) {
        pageable = PageRequest.of(pageable.getPageNumber(), pageable.getPageSize
    (), Sort.by("createdAt").descending());
    }

    SalesmanSpecification<Salesman> salesmanSpecification = new
    SalesmanSpecification<Salesman>();
    Specification<Salesman> specification = salesmanSpecification.build(
    salesmanRequestDto);

    Page<Salesman> salesmanPage = salesmanRepository.findAll(specification,
    pageable);
    int maxPageNumber = salesmanPage.getTotalPages();

    if (pageable.getPageNumber() > maxPageNumber && maxPageNumber > 0) {
        pageable = PageRequest.of(maxPageNumber - 1, pageable.getPageSize(), Sort
    .by("updatedAt").ascending());
        salesmanPage = salesmanRepository.findByDeletedFalse(pageable);
    }
    return salesmanPage.map(salesman -> salesmanMapper.map(salesman, SalesmanDto.
    class));
}
```

Kode 3.2: Metode Find All Salesman dalam Service

Kolom *salesman title* yang tidak bisa di-*update* hanya memerlukan

perbaikan 1 baris kode pada *SalesmanService*, yaitu kode untuk mendefinisikan *salesmanTitle* dalam *entity* yang akan disimpan datanya ke *database*. Gambar 3.7 menunjukkan baris kode yang ditambahkan.

```
salesman.setSalesmanTitle(salesmanUpdateDto.getSalesmanTitle());
```

Kode 3.3: Kode yang dibutuhkan untuk set salesman title

```
@Transactional(value = "masterTransactionManager")
public SalesmanDto updateSalesman(Long id, SalesmanUpdateDto salesmanUpdateDto) throws DataNotFoundEx
    Optional<Salesman> salesmanOptional = Optional.empty();
    Optional<String> myScope = starterUserScopeService.getMyScopeValue( scopeName: "MASTER_SALESMAN_OWN
    if (myScope.isPresent()) {
        if (myScope.get().equals(Long.toString(UserAuthInfo.getUserId()))) {
            salesmanOptional = salesmanRepository.findWithLockingPessimisticByIdAndCreatedByAndDelete
        } else {
            salesmanOptional = salesmanRepository.findWithLockingByIdAndDeletedFalse(id);
        }
    }
    if (salesmanOptional.isEmpty()) {
        throw new DataNotFoundException(masterMessageSource.getMessage(MASTER_SALESMAN_FINDID_NOTFOUN
    }
    Salesman salesman = salesmanOptional.get();
    if (!salesman.getVersion().equals(salesmanUpdateDto.getVersion())) {
        throw new DataLockException("Database version: " + salesman.getVersion() + ", Current version
    }
    salesman.setSalesmanName(MasterUtil.normalizeString(salesmanUpdateDto.getSalesmanName()));
    salesman.setKtpNo(salesmanUpdateDto.getKtpNo());
    salesman.setSalesmanAddress(salesmanUpdateDto.getSalesmanAddress());
    salesman.setPhoneNo(salesmanUpdateDto.getPhoneNo());
    salesman.setStatus(salesmanUpdateDto.getStatus());
    salesman.setBranch(branchRepository.getReferenceById(salesmanUpdateDto.getBranchId()));
    salesman.setSalesmanType(salesmanUpdateDto.getSalesmanType());
    salesman.setWarehouse(warehouseRepository.getReferenceById(salesmanUpdateDto.getWarehouseId()));
    salesman.setSalesmanTitle(salesmanUpdateDto.getSalesmanTitle());
    salesman.setProvince(provinceRepository.getReferenceById(salesmanUpdateDto.getProvinceId()));

    salesman = salesmanRepository.save(salesman);
    SalesmanDto salesmanDto = salesmanMapper.map(salesman, SalesmanDto.class);
    return salesmanDto;
}
```

Gambar 3.7. Tangkapan layar potongan kode bagian *updateSalesman*dengan penambahan *setSalesmanTitle*

B Submodul Supplier

B.1 Analisis Masalah

Dalam submodul *Supplier*, belum ada fitur untuk membuat atau "*generate*" *Chart of Account*. Fitur ini dibutuhkan untuk menyambungkan *Supplier* maupun

submodul lainnya dengan CoA baru spesifik untuk *Supplier* tersebut. Objek *Supplier* akan menyimpan id dari CoA terkait sebagai *foreign key*.

Data CoA yang perlu dibuat adalah kolom-kolom berikut pada *Supplier*:

- *accPayableCoaId*
- *accUnbilledPayableCoaId*
- *unbilledReceivableCoaId*
- *accGiroPayableCoaId*
- *accGiroReceivableCoaId*
- *accTransferPayableCoaId*
- *accTransferReceivableCoaId*

Pembuatan CoA memiliki perbedaan untuk setiap jenisnya pada bagian kode CoA, penamaan CoA, tipe akun, dan jenis pembayaran (debit/kredit). Sebagai contoh, *accPayableCoa* dibuat dengan kode prefix "100100", penamaan dasar Hutang Dagang, tipe akun "Accounts Payable", dan tipe pembayaran kredit.

Alur pembuatan CoA juga harus dirubah. Alur yang sudah ada menggunakan *dropdown* pada halaman pembuatan *Supplier* untuk memilih CoA pada masing-masing *field* dari data CoA yang sudah ada, sehingga tidak ada pembuatan CoA baru spesifik untuk *Supplier*. Alur yang baru memberikan *user* pilihan untuk mengosongkan masing-masing *field* CoA ketika pertama membuat *Supplier*, dan men-*generate* CoA setelah *Supplier* dibuat terlebih dahulu. Selain itu, *field* CoA yang akan di-*generate* hanya *field* yang belum menyimpan ID CoA atau *field* CoA yang masih kosong.

B.2 Solusi

Fitur *generate* CoA memerlukan perubahan pada *backend* untuk menangani *request* dari *frontend* dan untuk menangani logika pembuatan CoA, dengan perubahan paling banyak pada bagian *SupplierService*.

Field yang dibutuhkan untuk menyimpan ID CoA sudah ada dalam migrasi dan sudah didefinisikan dalam *entity* maupun DTO *Supplier*. Perubahan yang dilakukan terhadap DTO hanya berupa penghapusan validator @NotNull dan @MasterChartOfAccountIsExists pada *SupplierCreateDTO* serta

SupplierUpdateDTO yang berfungsi memastikan agar CoA ID ada dalam *request body* dan sudah ada dalam database ketika menjalankan *create* atau *update*. Penghapusan ini dilakukan sehingga kolom-kolom CoA dapat dikosongkan ketika melakukan *create* atau *update*. Gambar 3.8 menunjukkan perubahan yang dilakukan terhadap DTO.

```
-     @NotNull(message = "
  {master.common.accpayablecoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long accPayableCoaId;                    28      private Long accPayableCoaId;
-     @NotNull(message = "
  {master.common.accunbilledcoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long accUnbilledPayableCoaId;            29      private Long accUnbilledPayableCoaId;
-     @NotNull(message = "
  {master.common.unbillereceivablereturncoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long unbilledReceivableReturnCoaId;      30      private Long unbilledReceivableReturnCoaId;
-     @NotNull(message = "
  {master.common.accgiropayablecoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long accGiroPayableCoaId;                31      private Long accGiroPayableCoaId;
-     @NotNull(message = "
  {master.common.accgiroreceivablecoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long accGiroReceivableCoaId;             32      private Long accGiroReceivableCoaId;
-     @NotNull(message = "
  {master.common.acctransferpayablecoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long accTransferPayableCoaId;            33      private Long accTransferPayableCoaId;
-     @NotNull(message = "
  {master.common.acctransferreceivablecoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long accTransferReceivableCoaId;         34      private Long accTransferReceivableCoaId;
-     @NotNull(message = "
  {master.common.purchasescoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long purchasesCoaId;                     35      private Long purchasesCoaId;
-     @NotNull(message = "
  {master.common.purchasesreturncoa.notnull}")
-     @MasterChartOfAccountIsExists
      private Long purchasesReturnCoaId;               36      private Long purchasesReturnCoaId;
```

Gambar 3.8. Tangkapan layar *diff* (perubahan) pada *SupplierCreateDto* dan *SupplierUpdateDto*

Selain itu, dibuat juga DTO baru dengan nama *SupplierGenerateCoaDTO* khusus untuk mengirim *version* ketika menjalankan fitur *generate* CoA, mengingat fitur *generate* CoA ini sifatnya berupa *update request* data *Supplier*.

```
1 @Data
2 @Builder
3 @Jacksonized
4 public class SupplierGenerateCoaDto {
5     private Long version;
```

```
6 }
```

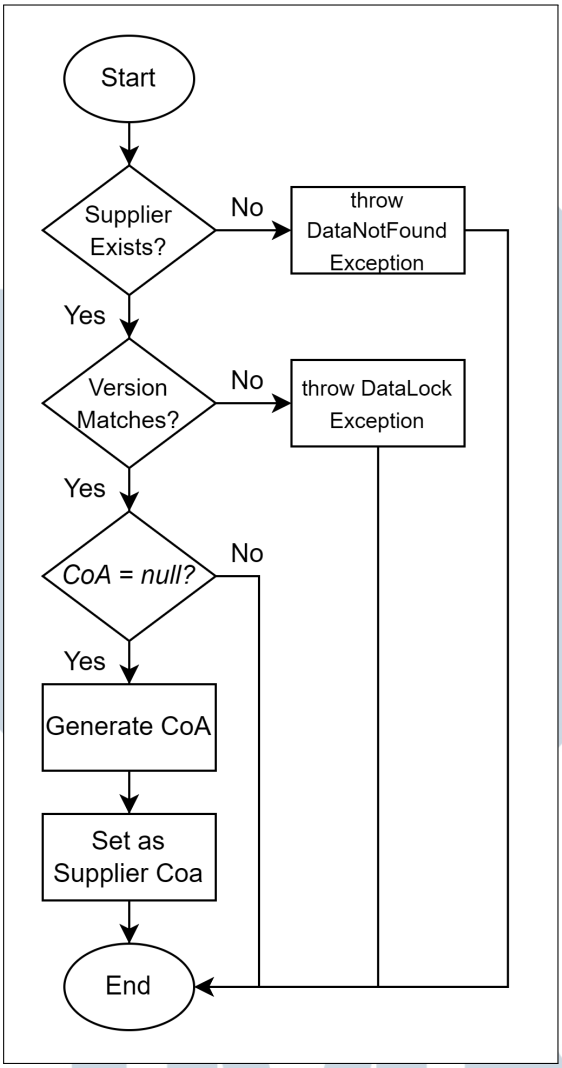
Kode 3.4: Class *SupplierGenerateCoaDto*

Endpoint baru ditambahkan pada *SupplierController* untuk menangani *request* pembuatan CoA dengan metode PATCH. Endpoint baru ini akan memanggil metode *generateSupplierCoa* dalam *SupplierService*. Endpoint ini akan menerima ID *Supplier* yang akan dibuat CoA-nya dalam bentuk *path variable* dan *SupplierGenerateCoaDTO* dalam bentuk *request body* untuk mengirimkan *version* dari *Supplier*.

```
1 @PatchMapping(value = "/supplier/{id}/coa", headers = "X-Api-Version=1")
2 @IsMasterSupplierUpdate
3 @ResponseStatus(value = HttpStatus.OK)
4 @LogExecutionTime
5 public SupplierDto generateSupplierCoa(@PathVariable Long id, @RequestBody
      @Validated SupplierGenerateCoaDto supplierGenerateCoaDto) throws
      DataNotFoundException, EntityNotFoundException, DataLockException {
6     return supplierService.generateSupplierCoa(id, supplierGenerateCoaDto);
7 }
```

Kode 3.5: Metode *generateSupplierCoa* dalam *SupplierController*

Metode *generateSupplierCoa* dalam *SupplierService* berfungsi untuk menangani semua logika pembuatan CoA untuk *Supplier*. Alur metode *generateSupplierCoa* dapat dijelaskan dalam gambar 3.9.



Gambar 3.9. Flowchart Pembuatan CoA pada Supplier (Sumber: Dokumen internal)

Pertama, *Supplier* akan diambil dulu berdasarkan ID yang didapatkan dari *SupplierController*. Apabila ID yang dikirimkan tidak valid dan tidak ada *Supplier* dengan ID tersebut, maka aplikasi akan mengirimkan *DataNotFoundException* yang akan ditampilkan kepada *user* dalam bentuk *response* 404 *NOT_FOUND*. Jika *Supplier* valid sudah didapatkan, maka *version* dari *request* dan dari *Supplier* akan dicek terlebih dahulu, apabila tidak sesuai maka aplikasi akan mengirimkan *DataLockException* dengan *response* berbentuk 409 *CONFLICT*. Setelah itu, CoA akan dibuat untuk masing-masing *field*, dengan pengecekan terlebih dahulu untuk memastikan *field* CoA yang sudah ada isinya tidak di-*generate* lagi. Setelah CoA dibuat dan disimpan ke database, ID CoA yang dibuat di-*assign* ke *Supplier* sebelum *Supplier* diperbarui datanya ke database.

```
@Transactional(value="masterTransactionManager")
public SupplierDto generateSupplierCoa(Long id, SupplierGenerateCoaDto
    supplierGenerateCoaDto) throws DataNotFoundException, EntityNotFoundException
    , DataLockException{
    Optional<Supplier> supplierOptional = Optional.empty();
    Optional<String> myScope =  starterUserScopeService.getMyScopeValue("
    MASTER_SUPPLIER_OWN");
    if(myScope.isPresent()){
        if (myScope.get().equals(Long.toString(UserAuthInfo.getUserId()))) {
            supplierOptional = supplierRepository.
    findWithLockingByIdAndCreatedByAndDeletedFalse(id, UserAuthInfo.getUserId());
        } else {
            supplierOptional = supplierRepository.
    findWithLockingByIdAndDeletedFalse(id);
        }
    }
    if (supplierOptional.isEmpty()){
        throw new DataNotFoundException(masterMessageSource.getMessage(
    MASTER_SUPPLIER_FINDID_NOTFOUND, new Object[]{id}, LocaleContextHolder.
    getLocale()));
    }
    Supplier supplier = supplierOptional.get();

    if (!supplier.getVersion().equals(supplierGenerateCoaDto.getVersion())){
        throw new DataLockException("Database version: " + supplier.getVersion()
    + ", Current version: " + supplierGenerateCoaDto.getVersion());
    }

    String supplierCode = supplier.getSupplierCode();
    String supplierCodeSuffix = supplierCode.length() < 4 ? supplierCode :
    supplierCode.substring(supplierCode.length() - 4);

    // Logika untuk generate Acc Payable COA jika masih null
    if (supplier.getAccPayableCoa() == null) {
        ChartOfAccountCreateDto accPayableCoaCreateDto = ChartOfAccountCreateDto.
    builder()
                .coaCode("100100." + supplierCodeSuffix)
                .coaName("Hutang Dagang " + supplier.getSupplierName())
                .description("Generated COA for 100100. " + supplierCodeSuffix +
    " Hutang Dagang")
                .completeCode("100100." + supplierCodeSuffix)
                .accountTypeId(AccountTypeId.ACCOUNTS_PAYABLE.getValue())
                .debitCreditAccountType(2)
                .setStartingBalance(false)
                .status((short) 1)
                .build();
        ChartOfAccountDto accPayableCoaDto = createChartOfAccount(
    accPayableCoaCreateDto);
        supplier.setAccPayableCoa(chartOfAccountRepository.getReferenceById(
    accPayableCoaDto.getId()));
    }

    // Logika untuk generate Acc Unbilled Payable COA, Unbilled Receivable Return
     COA,
```

```
41      // Acc Giro Payable COA, Acc Giro Receivable COA, Acc Transfer Payable COA,
        dan Acc Transfer Receivable COA.
42      // Pola implementasinya mirip dengan Acc Payable COA di atas.
43
44      supplier = supplierRepository.save(supplier);
45
46      return supplierMapper.map(supplier, SupplierDto.class);
47  }
```

Kode 3.6: Metode *generateSupplierCoa*

Unit test dan *contract test* pada BE juga disesuaikan terhadap perubahan-perubahan yang dilakukan, disertai dengan tambahan *test case* untuk fitur *generate CoA*.

Fitur *generate* CoA membutuhkan perubahan pada *frontend*. Perubahan pertama adalah pembuatan *hook* baru untuk mengakses *endpoint* yang telah dibuat di BE. *Hook* baru ini akan mengirimkan data berbentuk *type* baru bernama *UpdateSupplierCoaData* berisi *version*, sesuai dengan *SupplierGenerateCoaDTO*, serta ID dari *Supplier* pada *path variablenya*.

```
1   type UpdateSupplierCoaOptions = {
2     id: number;
3     data: UpdateSupplierCoaData;
4   };
5
6   export const updateSupplierCoa = ({
7     id,
8     data,
9   }: UpdateSupplierCoaOptions): Promise<Supplier> => {
10    return getApiClient().patch(
11      `/master-service/supplier/${id}/coa`,
12      data
13    );
14  };
15
16  type UseUpdateSupplierCoaOptions = {
17    onSuccess?: (supplier: Supplier) => void;
18    onError?: (error: ResponseError) => void;
19  };
20
21  export const useUpdateSupplierCoa = ({
22    onSuccess,
23    onError,
24  }: UseUpdateSupplierCoaOptions = {}) => {
25    const {
26      mutate: submit,
27      isLoading,
28      isError,
29      isSuccess,
30    } = useMutation({
31      mutationFn: updateSupplierCoa,
```

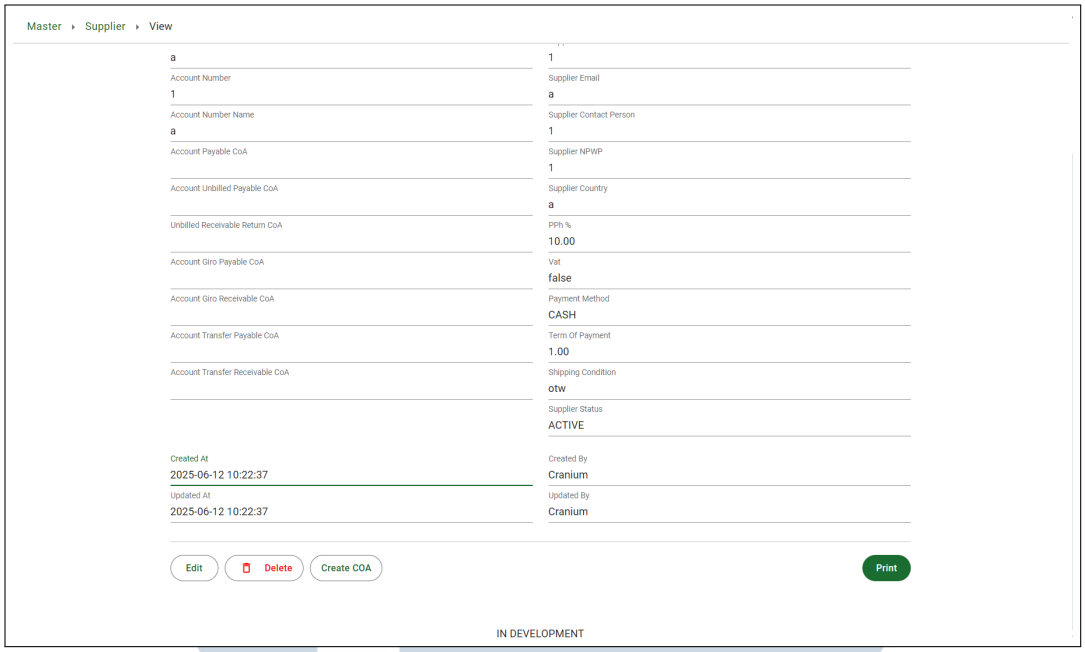
```
32      onSuccess: (supplier) => {
33        queryClient.invalidateQueries(['master.suppliers']);
34        queryClient.invalidateQueries([
35          'master.chartOfAccounts',
36        ]);
37        onSuccess?.(supplier);
38      },
39      onError: (error) => {
40        const err = customError(error);
41        onError?.(err);
42      },
43    });
44
45    return { submit, isLoading, isError, isSuccess };
46  };
```

Kode 3.7: API Hook useUpdateSupplierCoa

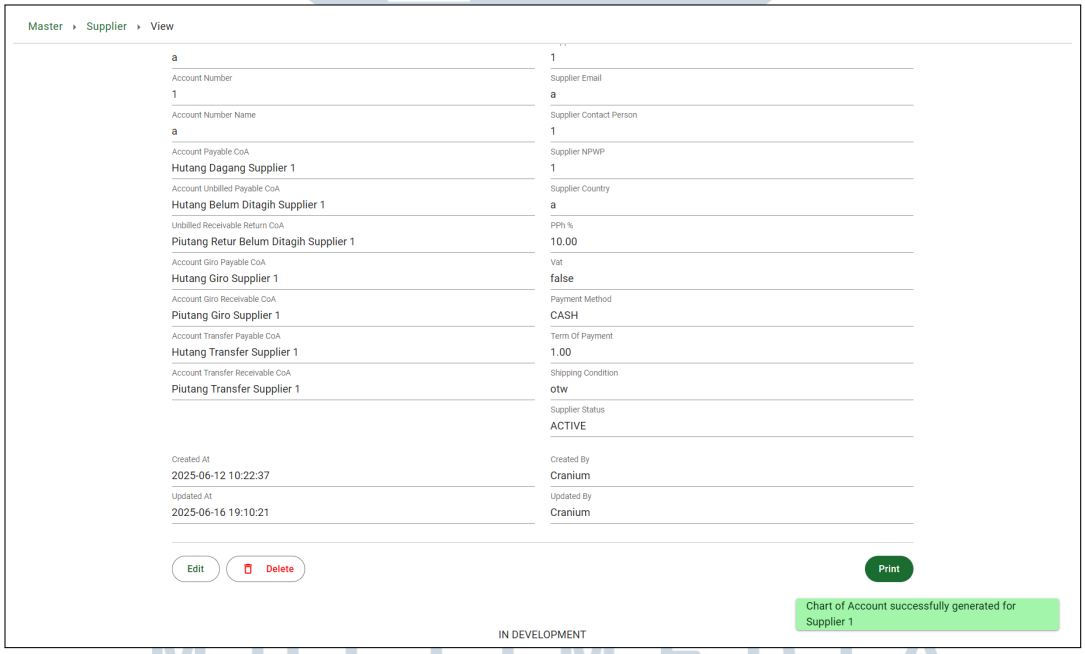
```
1  export type UpdateSupplierCoaData = Pick<
2      Supplier,
3      | 'version'
4  >;
```

Kode 3.8: UpdateSupplierCoaData type

Modifikasi juga dilakukan pada halaman *detail Supplier*, dengan penambahan tombol baru untuk *generate CoA*, tombol ini memunculkan *confirmation modal* sebelum melakukan *hook* untuk *generate CoA*. Tombol tidak akan muncul lagi apabila semua CoA sudah terbuat. Tampilan proses tersebut dapat dilihat pada gambar 3.10 dan gambar 3.11.



Gambar 3.10. Halaman detail *Supplier* sebelum CoA dibuat. (Sumber: Dokumen internal)



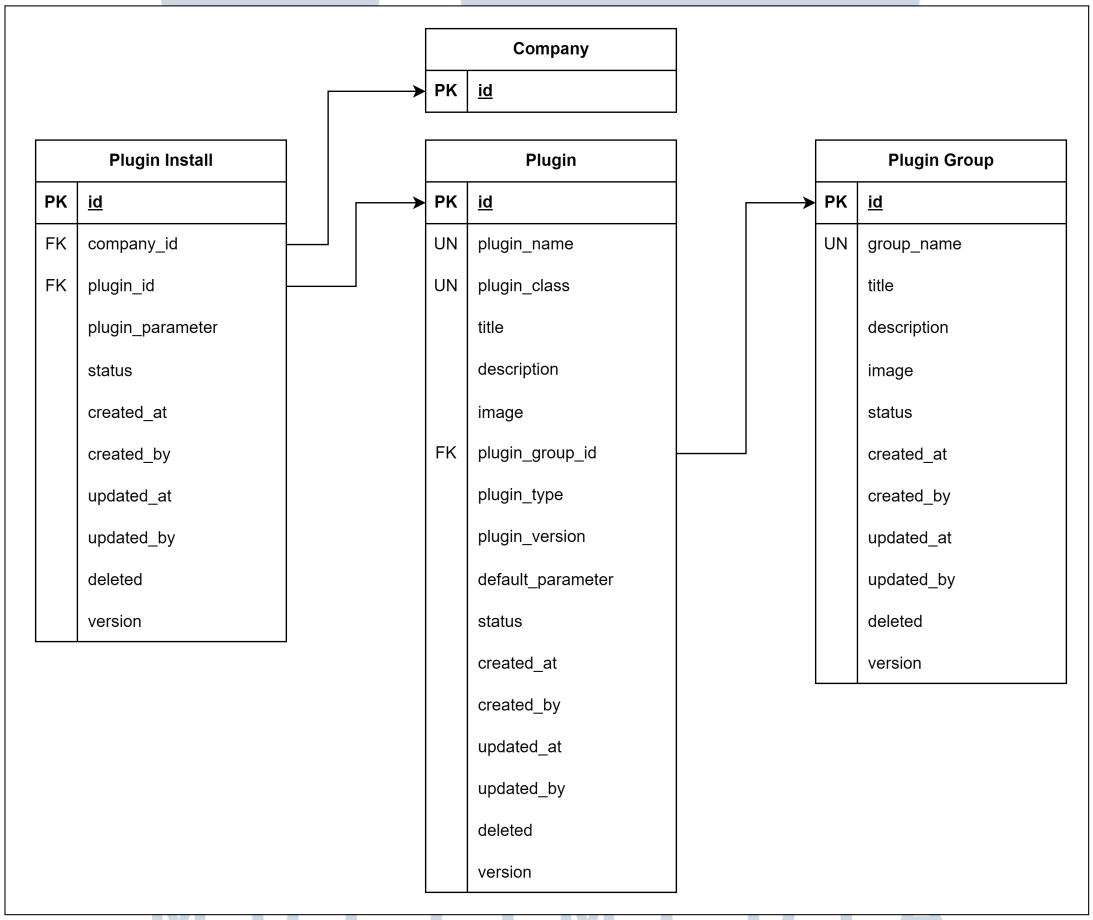
Gambar 3.11. Halaman detail *Supplier* setelah CoA dibuat. (Sumber: Dokumen internal)

Selain itu, modifikasi juga dilakukan pada form *create* dan *update*, dengan penghilangan validasi dan tag *required* pada kolom-kolom CoA sehingga dapat dikosongkan ketika *user* melakukan *create* atau *update Supplier*. Penyesuaian juga dilakukan pada *unit test frontend*.

C Submodul Plugin Group

C.1 Analisis Masalah

Plugin Group merupakan salah satu submodul dalam modul *Master* yang berfungsi untuk mengatur fungsionalitas *Plugin* dalam ERP Cranium. *Plugin Group* bekerja dengan *Plugin* dan *Plugin Install* untuk membentuk sistem pengaturan *plugin* dalam aplikasi ERP Cranium. *Plugin* dirancang sebagai sarana untuk memodifikasi *business logic* pada ERP Cranium tanpa memerlukan perubahan pada kode aplikasi. Skema database untuk sistem *plugin* dapat dilihat pada gambar 3.12. Tabel 3.2 menunjukkan struktur tabel *plugin_group* di *database*.



Gambar 3.12. ERD Database Plugin, PluginGroup, dan PluginInstall (Sumber: Dokumen internal)

Column Name	Column Type	Length	Null	Default	PK	Uniq
id	BIGINT				V	
group_name	VARCHAR	255				UQ1
title	VARCHAR	255				
description	VARCHAR	255				
image	VARCHAR	255				
status	INT		V			
created_at	TIMESTAMP					
created_by	BIGINT					
updated_at	TIMESTAMP					
updated_by	BIGINT					
deleted	BOOLEAN			FALSE		
version	BIGINT			1		

Tabel 3.2. Struktur Tabel Plugin Group (Sumber: Dokumen internal)

Plugin Group dibutuhkan untuk mengelompokkan *plugin* yang nantinya akan digunakan dalam ERP Cranium. Pengembangan yang dilakukan untuk *Plugin Group* adalah pembuatan CRUD pada *backend*, lengkap hingga pembuatan *endpoint* dan *unit test*. Pengembangan CRUD *Plugin Group* dilakukan secara independen terlebih dahulu, sehingga belum mendefinisikan relasi *Plugin Group* dengan tabel lainnya.

C.2 Solusi

Langkah pertama yang dilakukan dalam pembuatan CRUD untuk submodul *Plugin Group* adalah mendefinisikan tabel *plugin_group* dan *plugin_group_aud* menggunakan migrasi. Tabel dengan akhiran "*_aud*" merupakan tabel audit, yang merekam seluruh perubahan data yang terjadi pada database akibat operasi-operasi yang dilakukan oleh aplikasi. *Constraint* seperti *unique* dan *primary key* juga didefinisikan dalam migrasi.

Kemudian, *entity* untuk *Plugin Group* dibuat sebagai bentuk representasi objek dalam aplikasi. *Data Transfer Object* juga dibuat untuk *Plugin Group*, yaitu:

- *PluginGroupDTO* sebagai DTO utama, digunakan sebagai DTO yang dikirimkan dalam *response*.

- *PluginGroupCreateDTO* sebagai DTO yang digunakan dalam proses *create Plugin Group*.
- *PluginGroupUpdateDTO* sebagai DTO yang digunakan dalam proses *update Plugin Group*.
- *PluginGroupNameDTO* sebagai DTO yang digunakan dalam proses pengambilan daftar nama *Plugin Group*.
- *PluginGroupRequestDTO* sebagai DTO yang digunakan dalam *request* untuk pencarian *Plugin Group*, misalnya pada fitur *search box*.

Mapper dibuat untuk menghubungkan antara *entity* dan DTO. Selain itu, *enumeration* juga dibuat untuk *status* dari *Plugin Group* (ACTIVE atau INACTIVE). Selain itu terdapat *custom PluginGroupName* dan *custom mapper PluginGroupNameMapper*. *PluginGroupName* merupakan turunan dari *entity PluginGroup* yang hanya mendefinisikan data yang dibutuhkan dalam proses pengambilan daftar nama *PluginGroup*, hal ini memastikan hanya data nama dan *id* yang diambil dari database ketika *request* untuk daftar nama *Plugin Group* dijalankan. *PluginGroupMapper* digunakan untuk *mapping* antara *PluginGroupName* dengan *PluginGroupNameDTO*.

PluginGroupController, PluginGroupService, dan *PluginGroupRepository* dibuat untuk mendefinisikan *endpoint*, *business logic*, dan interaksi dengan database, secara berurutan.

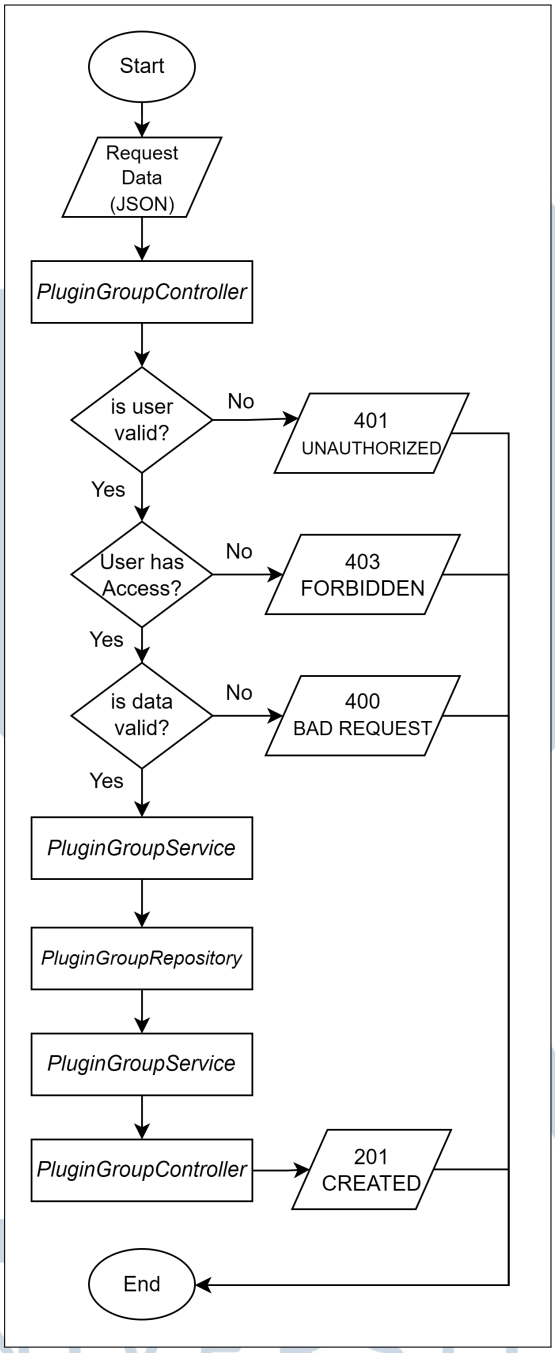
Validator dan *Security Annotation* juga dibuat untuk menjamin akses dan manipulasi data sesuai dengan kriteria dan aturan yang berlaku.

C.2.1 Proses *Create Plugin Group*

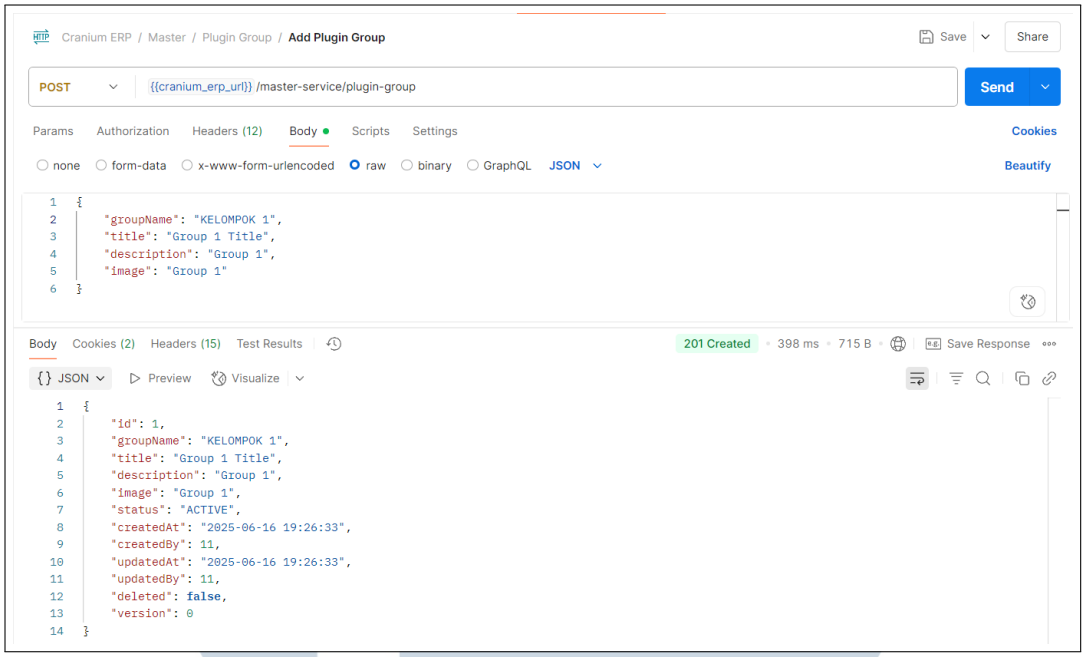
Proses *create Plugin Group* menggunakan *endpoint* "/master-service/plugin-group" dengan metode POST. Data dapat dikirimkan oleh *client* dalam format JSON pada *request body*. Setelah data dikirimkan, *user* akan dicek terlebih dahulu apakah memiliki *authority* untuk melakukan *create request*. Jika tidak ditemukan *user* yang terautentikasi (tidak ada *bearer token* yang dikirim bersama dengan data *request*), maka aplikasi akan mengirimkan *response* 401 *UNAUTHORIZED*. Jika *user* tidak memiliki *authority* untuk melakukan *request*, maka aplikasi akan mengirimkan *response* berupa 403 *FORBIDDEN*.

Setelah pengecekan *user*, data yang dikirimkan akan dicek juga untuk memastikan kelengkapan dan kesesuaian. Kriteria kelengkapan dan batasan data didefinisikan dalam *PluginGroupCreateDTO*. Apabila ada data yang tidak lengkap, tidak sesuai kriteria atau melewati batasan, maka aplikasi akan mengirimkan *response* berupa 400 *BAD_REQUEST*. Data yang lengkap dan sesuai dengan kriteria yang telah ditentukan akan dikirimkan ke *PluginGroupService* dalam bentuk DTO untuk diproses sesuai dengan *business logic*, kemudian ditransformasi menjadi bentuk *entity* sebelum dimasukkan ke database melalui *PluginGroupRepository*. Ketika data sudah berhasil di-input ke database, maka aplikasi akan mengirimkan *response* berupa 201 *CREATED* disertai data *PluginGroup* yang berhasil dibuat (dalam bentuk *PluginGroupDTO* yang dikirim dari *PluginGroupService* ke *PluginGroupController*) untuk menunjukkan data berhasil dibuat di database.

Dalam proses *create Plugin Group*, terdapat *security annotation* @IsMasterPluginGroupCreate dan *validator* @MasterPluginGroupNameIsUnique yang dicantumkan sebagai anotasi pada *PluginGroupController* serta *validator* @MasterPluginGroupNameIsCapitalized yang dicantumkan pada *PluginGroupCreateDTO*. *IsMasterPluginGroupCreate* merupakan anotasi yang merujuk kepada pengecekan *role* yang dibutuhkan oleh *user* agar dapat melakukan *create Plugin Group*. Sementara *MasterPluginGroupNameIsUnique* merujuk ke *validator* yang memastikan nama *Plugin Group* yang dikirimkan pada *request body* tidak sama dengan nama *Plugin Group* yang sudah ada di dalam database, sehingga memenuhi *unique constraint* yang sudah ditentukan pada kolom *group_name*. *MasterPluginGroupNameIsCapitalized* merupakan *validator* yang memastikan data yang dikirimkan dari *client* mengandung variabel *groupName* yang seluruhnya menggunakan huruf kapital. Gambar 3.13 menunjukkan *flowchart* untuk *create Plugin Group*. Gambar 3.14 menunjukkan hasil pengujian *endpoint* untuk *create* menggunakan Postman.



Gambar 3.13. Flowchart *Create Plugin Group* (Sumber: Dokumen internal)



Gambar 3.14. Pengujian *create Plugin Group* menggunakan Postman

C.2.2 Proses *Update Plugin Group*

Proses *update Plugin Group* menggunakan *endpoint* "/master-service/plugin-group/{id}" dengan metode PATCH. Data perubahan dikirimkan oleh *client* dalam format JSON pada *request body*, sementara *id* dari *Plugin Group* yang akan diubah dikirim sebagai *path variable*. Kemudian, *user* akan dicek terlebih dahulu apakah memiliki *authority* yang dibutuhkan untuk melakukan *update request*. Jika tidak ditemukan *user* yang terautentikasi (tidak ada *bearer token* yang dikirim bersama dengan data *request*), maka aplikasi akan mengirimkan *response* 401 *UNAUTHORIZED*. Jika *user* tidak memiliki *authority* untuk melakukan *request*, maka aplikasi akan mengirimkan *response* berupa 403 *FORBIDDEN*.

Setelah pengecekan *user*, data akan dikirimkan dalam bentuk *PluginGroupUpdateDTO* ke *service* untuk penanganan *business logic*. Apabila ada kolom data yang tidak lengkap atau tidak sesuai dengan ketentuan pada DTO, maka aplikasi akan mengirimkan *response* 400 *BAD_REQUEST*. Pada *PluginGroupService*, dilakukan pengecekan *scope* terlebih dahulu. Pengecekan *scope* dilakukan sehingga *user* hanya dapat mengakses data yang ada di dalam batasan aksesnya.

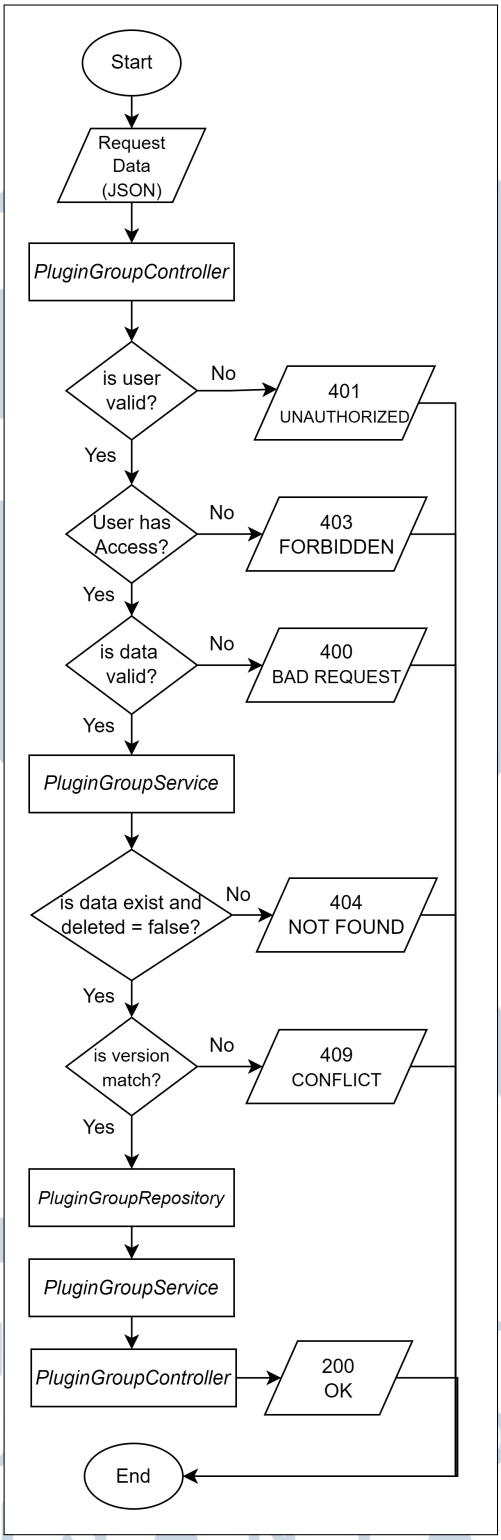
Apabila pengecekan *scope* berhasil, maka aplikasi akan mengambil data

Plugin Group berdasarkan *id* yang dikirimkan dari *controller* menggunakan metode yang sudah didefinisikan pada *repository*. Karena ERP Cranium menggunakan metode *soft delete* untuk penghapusan data, maka data yang dicari hanya data dengan kolom *deleted* bernilai *false*. Pengambilan data juga menggunakan metode *locking* untuk memastikan perubahan data tidak bertabrakan dengan transaksi lain terhadap data. Apabila data tidak ditemukan berdasarkan kriteria yang diminta, maka aplikasi akan mengirimkan *response* berupa 404 *NOT_FOUND*.

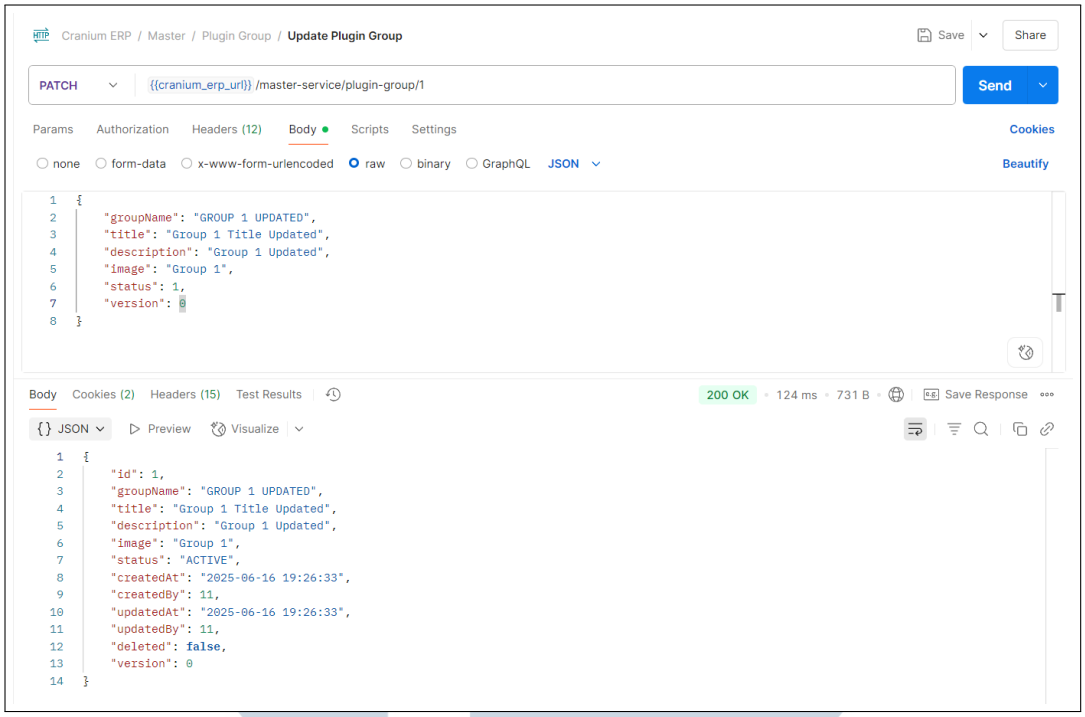
Data yang didapatkan dari database kemudian diubah menjadi bentuk objek *PluginGroup*, kemudian diperiksa kesesuaian *version* antara *version* yang dikirimkan dari *client* (dalam *PluginGroupUpdateDTO*), dan *version* yang didapatkan dari database (dalam objek *PluginGroup*). Apabila *version* tidak sesuai, maka aplikasi akan melontarkan *DataLockException* dan mengirimkan *response* 409 *CONFLICT*. Proses ini merupakan bentuk implementasi *optimistic locking*, yang memastikan manipulasi data dilakukan terhadap versi data yang terbaru.

Jika *version* sesuai, maka aplikasi akan mengubah data dari objek *PluginGroup* dengan data dari *PluginGroupUpdateDTO* kemudian menyimpan objek *PluginGroup* ke database melalui *repository*. Aplikasi akan mengirimkan *response* 200 *OK* disertai dengan data *Plugin Group* yang telah diperbarui pada *response body*.

Proses *update Plugin Group* menggunakan *security annotation* @IsMasterPluginGroupUpdate dan *validator* @MasterPluginGroupNameIsUnique yang dicantumkan sebagai anotasi pada *PluginGroupController* serta *validator* @MasterPluginGroupNameIsCapitalized yang dicantumkan pada *PluginGroupUpdateDTO*. *IsMasterPluginGroupUpdate* merujuk pada pengecekan *role* yang dibutuhkan oleh *user* untuk melakukan perubahan data *PluginGroup*. Penggunaan *MasterPluginGroupNameIsUnique* adalah untuk memastikan tidak ada nama *PluginGroup* yang sama dan *MasterPluginGroupNameIsCapitalized* untuk memastikan nama *PluginGroup* seluruhnya menggunakan huruf kapital. Gambar 3.15 menunjukkan *flowchart* untuk *update Plugin Group*. Gambar 3.16 menunjukkan hasil pengujian *endpoint* untuk *create* menggunakan Postman.



Gambar 3.15. Flowchart *Update Plugin Group* (Sumber: Dokumen internal)



Gambar 3.16. Pengujian *update Plugin Group* menggunakan Postman

C.2.3 Proses *Read Plugin Group*

Proses *read* atau *get Plugin Group* dapat dibagi menjadi tiga jenis, yaitu:

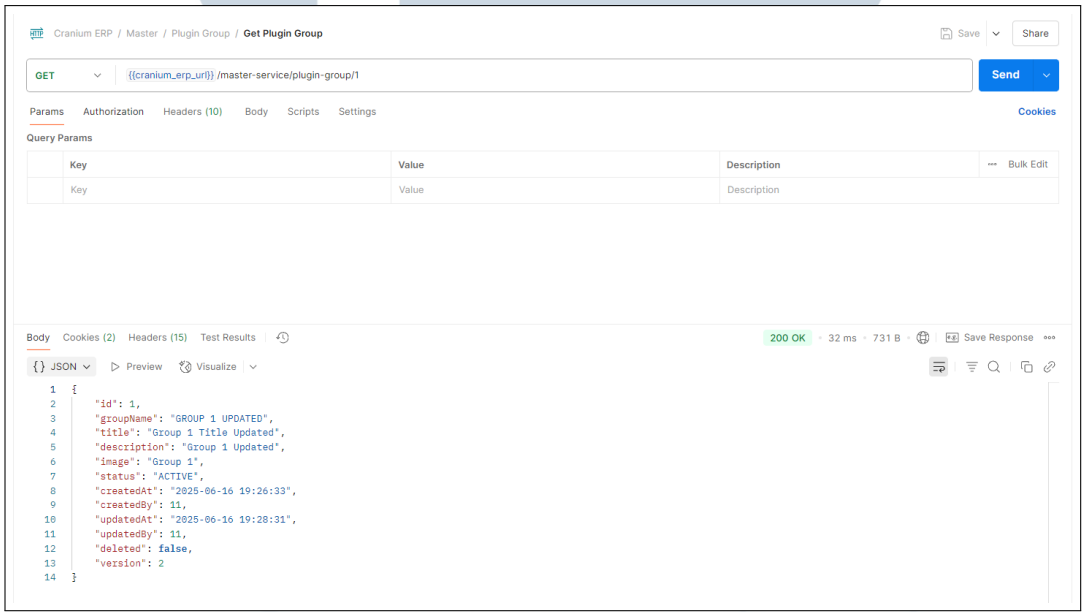
- *Get Plugin Group by ID*
- *Get Plugin Group List (Searching)*
- *Get Plugin Group Names*

Proses *get Plugin Group by ID* menggunakan *endpoint* "/master-service/plugin-group/{id}" dengan metode GET. *Client* mengirimkan *request* ke server dengan menempatkan *id* dari *Plugin Group* yang diinginkan sebagai *path variable*. Kemudian, *user* akan dicek terlebih dahulu apakah memiliki *authority* yang dibutuhkan untuk melakukan *read* data *PluginGroup*. Jika tidak ditemukan *user* yang terautentikasi (tidak ada *bearer token* yang dikirim bersama dengan data *request*), maka aplikasi akan mengirimkan *response* 401 *UNAUTHORIZED*. Jika *user* tidak memiliki *authority* untuk melakukan *request*, maka aplikasi akan mengirimkan *response* berupa 403 *FORBIDDEN*.

Setelah *user* dicek, data *id* akan dikirim ke *service*. Pada *service*, dilakukan pengecekan *scope* terlebih dahulu untuk memastikan data *PluginGroup* yang akan

dicari merupakan data yang berada di dalam batasan akses *user*. Apabila *scope* sesuai, maka aplikasi akan mengambil data *Plugin Group* menggunakan *id* yang dikirim dari *controller* dengan menggunakan metode yang sudah didefinisikan dalam *repository*. ERP Cranium menggunakan metode *soft delete* untuk penghapusan data, maka data yang dicari hanya data dengan kolom *deleted* bernilai *false*. Apabila data tidak ditemukan berdasarkan kriteria yang diminta (*id* sesuai dan *deleted* bernilai *false*) , maka aplikasi akan melontarkan *DataNotFoundException* dan mengirimkan *response* berupa 404 *NOT_FOUND*.

Data yang didapatkan dari database melalui *repository* kemudian diubah menjadi bentuk objek *PluginGroup*, dan di-*map* menjadi bentuk *PluginGroupDTO* untuk diterima oleh *controller* dan dikirimkan ke *client* dalam bentuk JSON beserta dengan *response* 200 *OK*. Pengujian proses *get Plugin Group* dapat dilihat pada gambar 3.17.



Gambar 3.17. Pengujian *get Plugin Group* menggunakan Postman

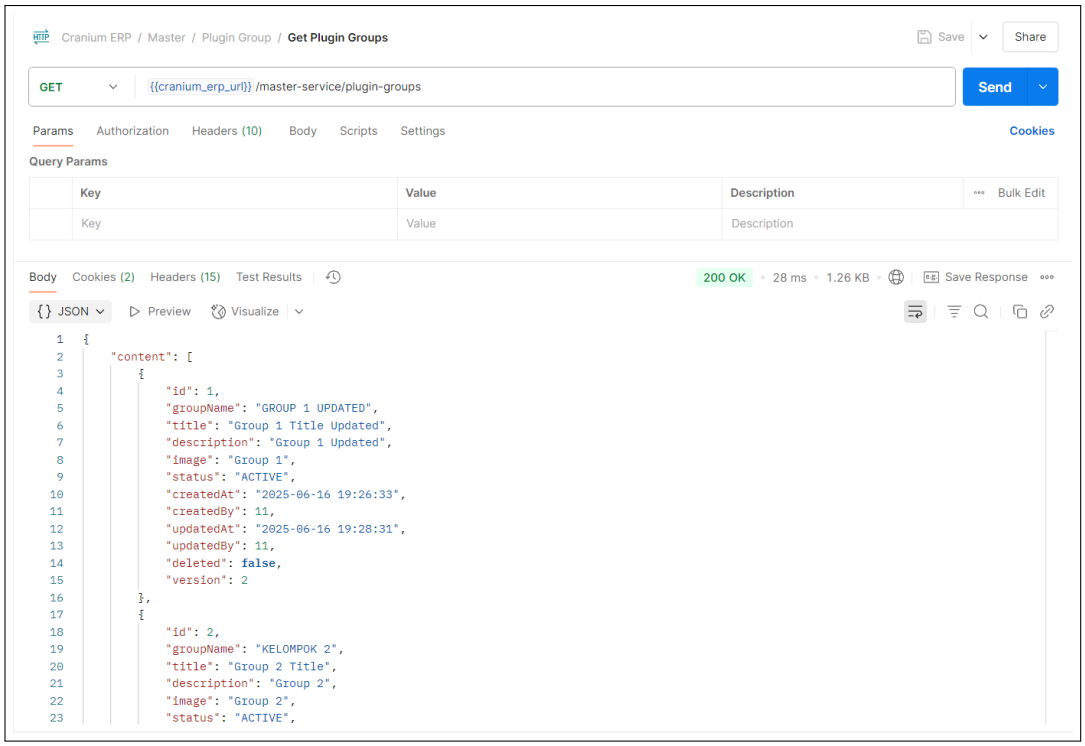
Proses *get Plugin Group List* atau *search Plugin Group* menggunakan *endpoint* "/master-service/plugin-groups" dengan metode GET dan disertai dengan *query parameters* dan *pageable* yang akan digunakan untuk melakukan pencarian *Plugin Group*.

Pageable merupakan *interface* yang disediakan oleh Spring Data JPA untuk melakukan *pagination* dan *sorting* dalam proses pengambilan data. *Pageable* berfungsi sebagai wadah untuk informasi yang akan digunakan dalam *pagination*

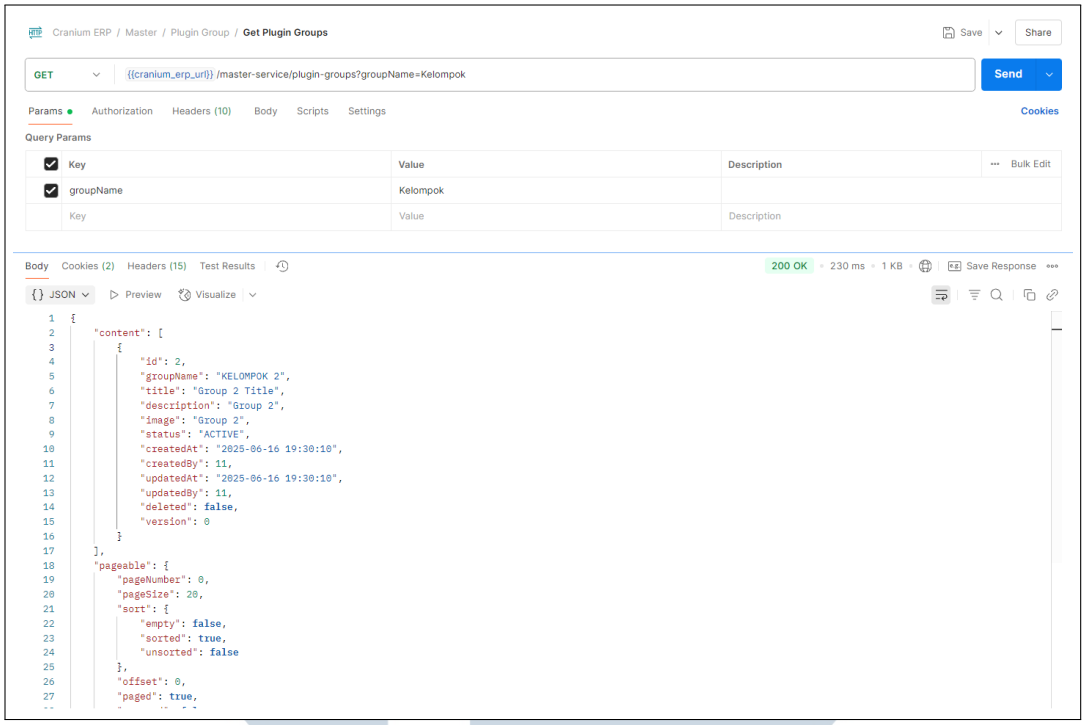
dan *sorting*. *Pageable* dapat menyimpan informasi seperti *pageNumber*, *pageSize*, dan *sort*.

Query parameters akan dikonversi menjadi bentuk *PluginGroupRequestDTO* untuk dikirimkan ke *service*. DTO ini dibuat khusus untuk mendefinisikan data yang bisa digunakan untuk fitur *searching*. Pada *service*, aplikasi akan membuat *specification* untuk mendefinisikan kriteria pencarian. *Specification* adalah fitur dalam Spring Data JPA yang memungkinkan penyusunan kriteria pencarian secara dinamis. *Pageable* dan *specification* akan digunakan oleh *repository* untuk menyusun pencarian objek *Plugin Group* yang sesuai di database. Informasi yang disimpan oleh *pageable* akan diterjemahkan menjadi klausa "LIMIT","OFFSET", dan "ORDER BY" dalam SQL, sementara informasi dari *specification* akan diterjemahkan menjadi klausa "WHERE" dalam SQL untuk pencarian data.

Hasil pencarian data akan berbentuk *Page* berisi objek *PluginGroup* yang sesuai kriteria pencarian. Masing-masing objek dalam *page* akan di-*map* menjadi bentuk *PluginGroupDTO* sebelum dikirimkan dalam bentuk JSON disertai dengan *response* 200 *OK*. Pengujian proses *get Plugin Groups* dapat dilihat pada gambar 3.18 dan 3.19.

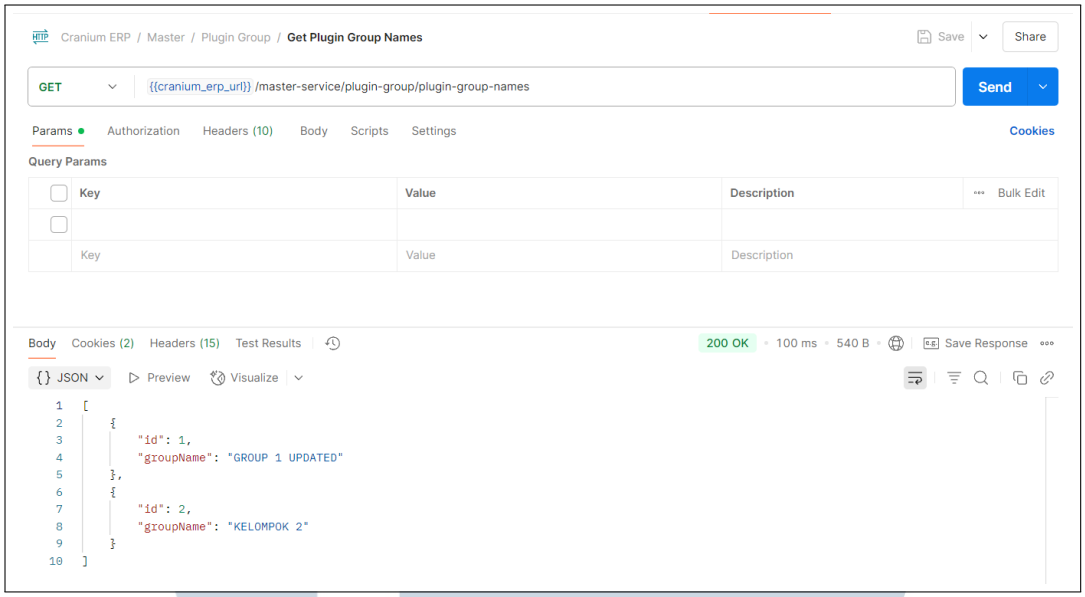


Gambar 3.18. Pengujian *get Plugin Groups* menggunakan Postman



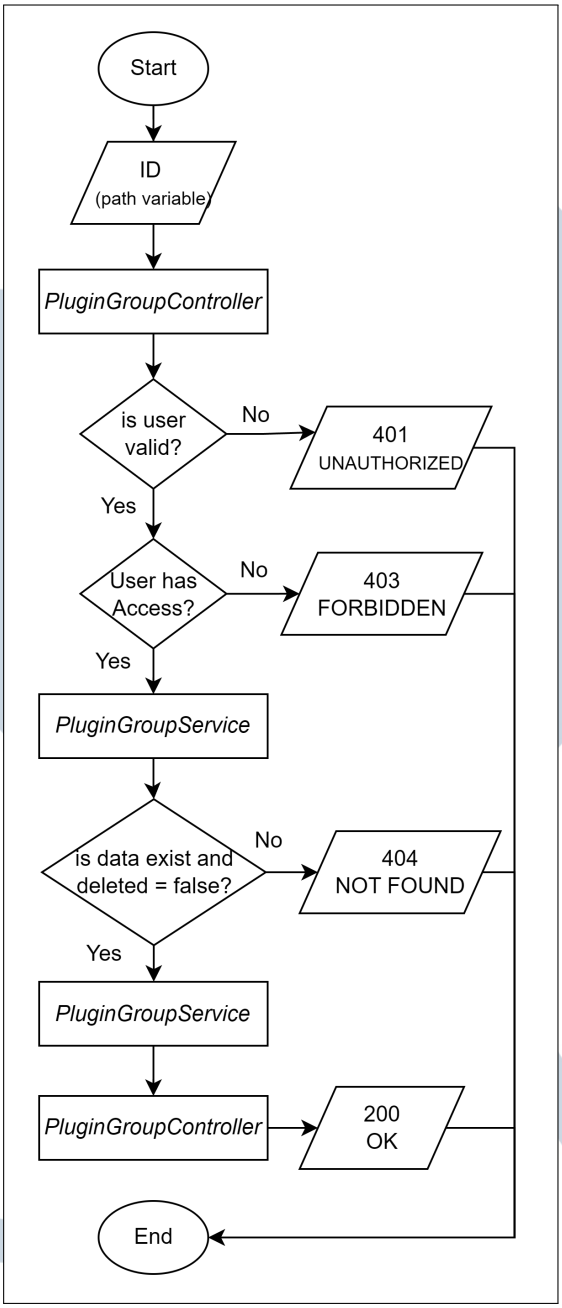
Gambar 3.19. Pengujian *get Plugin Groups* dengan *groupName* sebagai *search parameter* menggunakan Postman

Proses *get Plugin Group Names* menggunakan *endpoint* "/master-service/plugin-group/plugin-group-names" dengan metode GET. *Service* akan langsung memanggil metode *repository* yang bertanggung jawab mengambil semua data nama *Plugin Group*. Metode *repository* yang digunakan untuk mengambil data nama *Plugin Group* menggunakan *native query* untuk memastikan hanya data *group_name* dan *id* dari *Plugin Group*. Hasil pencarian data akan berbentuk *List* berisi *custom entity PluginGroupName*, yang akan di-map oleh *service* menjadi *List* yang berisi *PluginGroupNameDTO*. DTO ini dibuat khusus untuk mendefinisikan data yang akan ditampilkan yaitu id dan nama *Plugin Group*. Pengujian proses *get Plugin Group names* dapat dilihat pada gambar 3.20.



Gambar 3.20. Pengujian *get Plugin Group names* menggunakan Postman

Semua proses dengan *method* GET menggunakan anotasi @IsMasterPluginGroupRead untuk pengecekan *role* yang dibutuhkan oleh *user* untuk melakukan *request* pengambilan data. Gambar 3.21 menunjukkan *flowchart* untuk *get Plugin Group*.



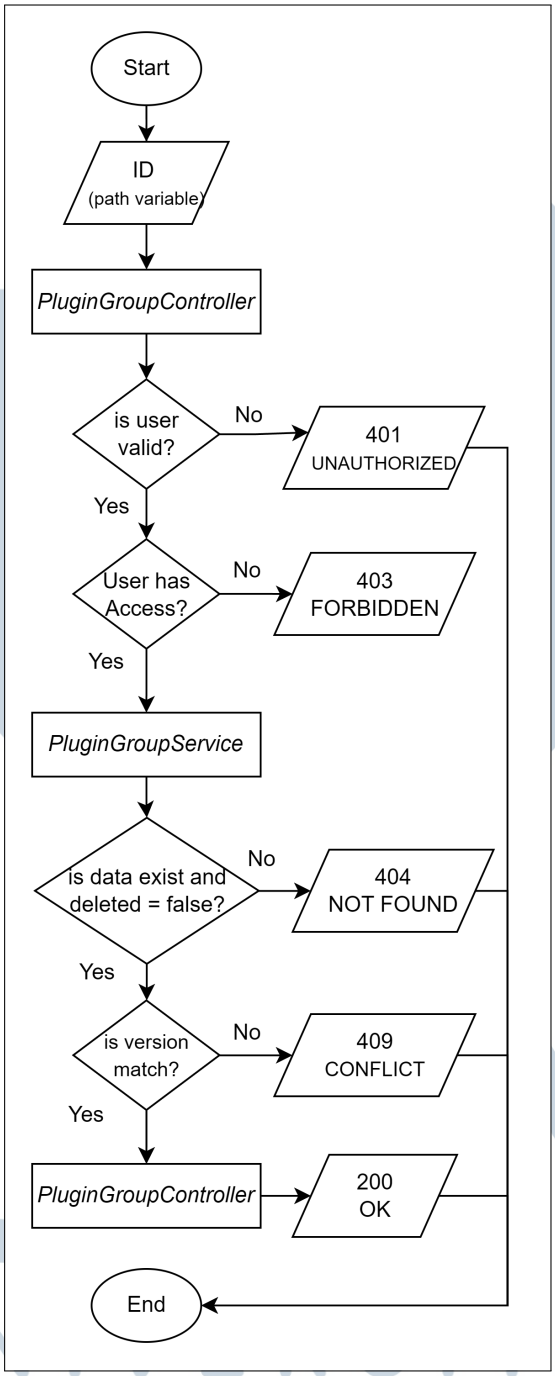
Gambar 3.21. Flowchart *Read Plugin Group* (Sumber: Dokumen internal)

C.2.4 Proses *Delete Plugin Group*

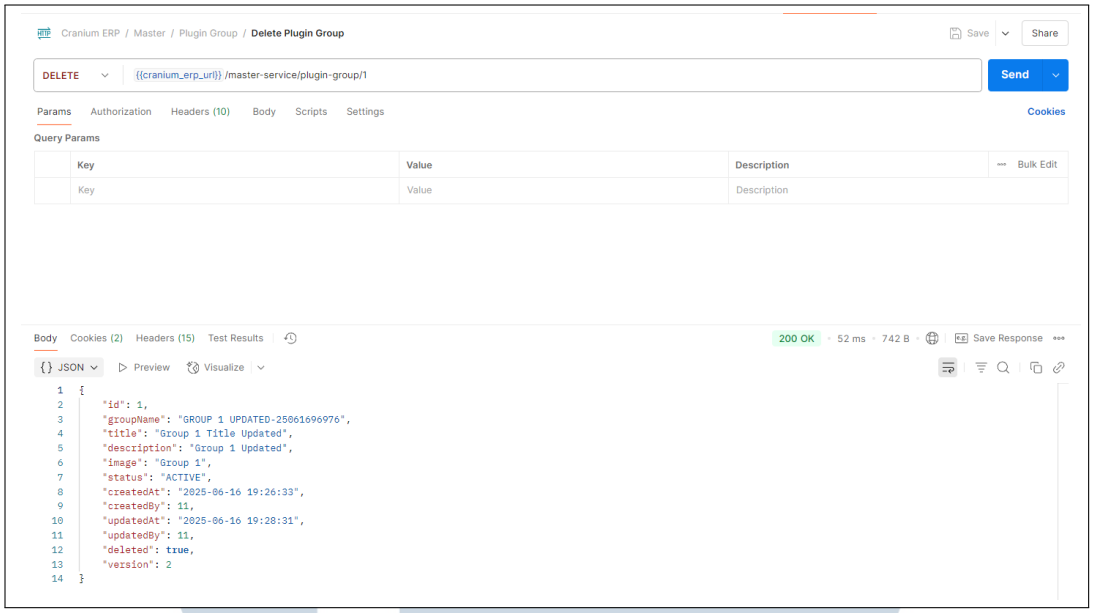
Proses *delete Plugin Group* menggunakan *endpoint* "/master-service/plugin-group/{id}" dengan metode DELETE. *Client* mengirimkan *request* ke server dengan mencantumkan id dari data yang ingin dihapus sebagai *path variable*, dan id tersebut akan dikirimkan ke *service*. Pada *service*, aplikasi akan

melakukan *try-catch* untuk penghapusan *Plugin Group*. Aplikasi akan mencoba untuk menjalankan logika penghapusan data, dengan terlebih dahulu melakukan pengecekan *scope* dan pengecekan eksistensi data yang ingin dihapus pada database. Sama seperti proses pengambilan data, aplikasi akan melontarkan *DataNotFoundException* dan mengirimkan *response* 404 *NOT_FOUND* apabila data tidak ditemukan. Objek *Plugin Group* yang sudah didapatkan akan di-*soft delete*, dengan mengubah properti *deleted* menjadi bernilai *false* dan menambahkan rangkaian acak pada *groupName* sebagai *suffix* (akhiran) untuk memastikan nama *Plugin Group* yang dihapus dapat digunakan oleh *entry* data lain. Setelah properti diubah, data akan disimpan ke database dan aplikasi melakukan *mapping* data objek *PluginGroup* yang dihapus menjadi bentuk *PluginGroupDTO* untuk dikirimkan kembali ke *client* dalam bentuk JSON disertai dengan *response* 200 *OK*.

Try-catch digunakan dalam proses penghapusan karena salah satu tahapan dalam penghapusan adalah mengambil data *Plugin Group* dari database dengan menggunakan metode *pessimistic locking* untuk mencegah konflik dalam pengubahan data. Hal ini berarti apabila ada transaksi lain yang sedang melakukan manipulasi terhadap data yang akan dihapus, maka aplikasi akan melontarkan *PessimisticLockingFailure*, yang akan dikirimkan oleh aplikasi sebagai *DataLockException* dengan *response* 409 *CONFLICT*. Gambar 3.22 menunjukkan *flowchart* untuk *delete Plugin Group*. Gambar 3.23 menunjukkan hasil pengujian *endpoint* untuk *create* menggunakan Postman.



Gambar 3.22. Flowchart *Delete Plugin Group* (Sumber: Dokumen internal)



Gambar 3.23. Pengujian *delete Plugin Group* menggunakan Postman

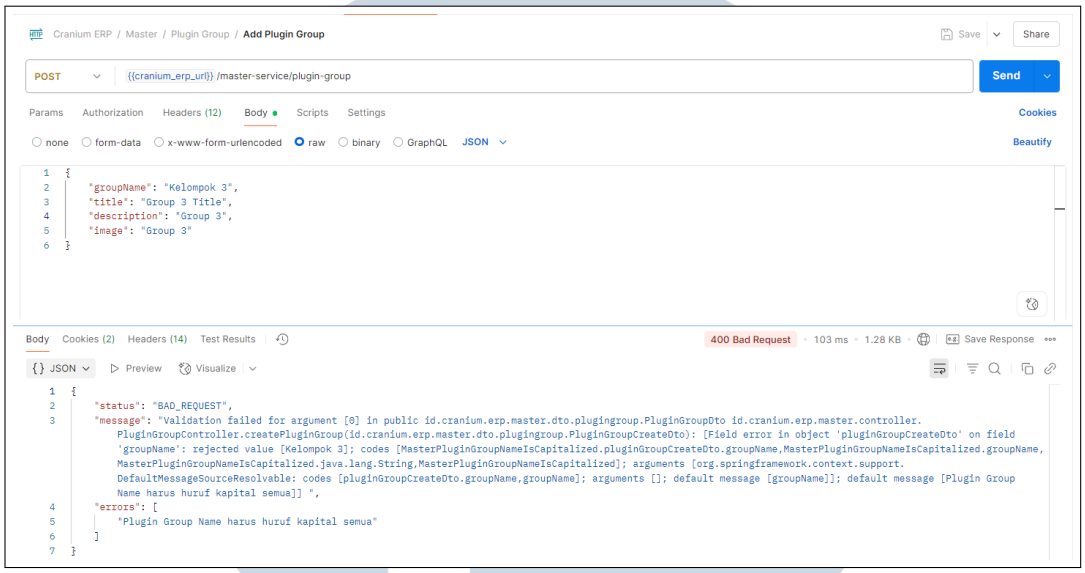
C.2.5 Custom Validator

Validasi data merupakan hal yang penting untuk diimplementasikan dalam pembuatan CRUD. Beberapa jenis validasi sudah disediakan oleh Java Spring Boot, contohnya seperti validasi *NotNull*, *NotEmpty*, *MinMax*, dan sebagainya. Namun, ada beberapa kriteria data yang memerlukan *custom validator* untuk melakukan validasinya. Terdapat dua *custom validator* dalam submodul *Plugin Group* untuk properti *groupName*, yaitu *MasterPluginGroupNameIsCapitalized* dan *MasterPluginGroupNameIsUnique*. Keduanya berupa *interface* dan dicantumkan sebagai *annotation*.

MasterPluginGroupNameIsCapitalized adalah *validator* yang bertujuan untuk memastikan nama *Plugin Group* yang dikirimkan oleh *client* pada proses *create* dan *update* dikapitalisasi (tersusun atas huruf kapital semuanya). Anotasi *validator* ini dicantumkan pada *PluginGroupCreateDTO* dan *PluginGroupUpdateDTO*, sehingga pengecekannya dilakukan ketika data *client* dikonversi menjadi DTO. *Validator* ini bekerja dengan membandingkan antara nama yang dikirimkan oleh *client* dengan nama yang dikirimkan oleh *client* namun dikonversi ke *uppercase*. Gambar 3.24 menunjukkan tampilan pada Postman ketika pengguna mencoba membuat *Plugin Group* dengan nama yang tidak dikapitalisasi. Logika utama dari *validator* ini adalah sebagai berikut:

```
return groupName.equals(groupName.toUpperCase());
```

Kode 3.9: Kode logika utama MasterPluginGroupNameIsCapitalized

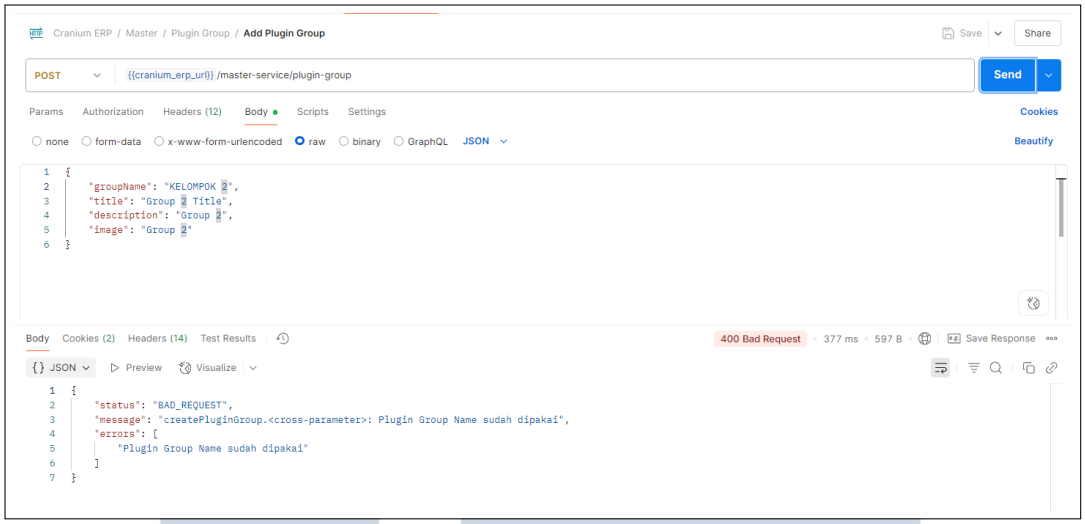


Gambar 3.24. Tampilan pada Postman jika nama tidak dikapitalisasi

MasterPluginGroupNameIsUnique adalah *validator* yang bertujuan untuk memastikan tidak ada nama *Plugin Group* yang sama, sesuai dengan *unique constraint* yang telah diterapkan pada database. Anotasi *validator* ini dicantumkan pada *PluginGroupController*, sehingga pengecekannya dilakukan ketika data *client* diterima oleh *endpoint*. *Validator* ini bekerja dengan melakukan *query* database untuk mencari apakah nama yang dikirimkan oleh *client* sudah ada dalam database atau belum. Gambar 3.25 menunjukkan tampilan pada Postman apabila pengguna mencoba membuat *Plugin Group* dengan nama yang sudah terpakai. Logika utama dari *validator* ini adalah sebagai berikut:

```
1 PluginGroupCreateDto pluginGroupCreateDto =
      pluginGroupCreateDtoOptional.get();
2 pluginGroupService.findByName(pluginGroupCreateDto.getGroupName
      ());
```

Kode 3.10: Kode logika utama MasterPluginGroupNameIsUnique



Gambar 3.25. Tampilan pada Postman jika nama sudah terpakai

C.2.6 *Unit Test* dan *Contract Test*

Unit test dan *contract test* juga dibuat untuk setiap metode dan setiap *endpoint* yang dibuat dalam submodul *Plugin Group*. Hal ini memastikan semua metode dan *endpoint* yang dibuat dapat berjalan sesuai dengan ekspektasi.

3.3.5 Pengembangan Modul Selling

A SellingVatWebClient

Dalam penerapan arsitektur *modular monolithic* ERP Cranium, aplikasi tidak dirancang dengan metode pengambilan data antar modul. Komunikasi antar modul dilakukan dengan menggunakan *webclient*. *Webclient* adalah komponen yang memampukan aplikasi Java untuk mengirim *HTTP Request* ke server web lain dan menerima *HTTP response* [15]. Dalam ERP Cranium, *webclient* digunakan untuk mengirim *request* ke server sendiri (internal *webclient*), secara spesifik ke *endpoint* pada modul lain.

SellingVatWebClient dibuat untuk mengirim *request* ke submodul *Value Added Tax* di modul *Master* dari submodul dan proses yang ada dalam modul *Selling*. *SellingVatWebClient* berisi beberapa fungsi pengambilan data dari modul *Master Value Added Tax*. Dua fungsi pengambilan data VAT yang dibuat adalah *getVatById* dan *getVatListByVatIds*. Kedua metode tersebut berfungsi mengambil data VAT atau PPn (Pajak Pertambahan) dari modul *Master* agar perhitungan PPn

pada modul *Selling* dapat menggunakan nilai persentase PPn yang sudah dibuat dalam modul *Master*.

Metode *getVatById* digunakan untuk mengambil data objek VAT berdasarkan id. Metode ini mengirimkan *HTTP request* ke *endpoint* "/master-service/value-added-tax/{id}" disertai dengan id sebagai *path variable*. Metode ini menerima *HTTP response* dalam format JSON dan diubah menjadi bentuk *ValueAddedTaxDTO*.

```
1  public ValueAddedTaxDto getVatById(Optional<HttpHeaderDto> httpHeaderDto, Long
       vatId) throws ClientException, ServerException {
2      String url = masterServiceVatById + "/{vatId}";
3      return internalWebClient.get()
4              .uri(url, uriBuilder -> uriBuilder.build(vatId))
5              .accept(MediaType.APPLICATION_JSON)
6              .retrieve()
7              .bodyToMono(ValueAddedTaxDto.class)
8              .contextWrite(context -> context.put(InternalWebFluxConfiguration.
       CONTEXT_HTTPHEADERDTO, httpHeaderDto.get()))
9              .retryWhen(Retry.backoff(3, Duration.ofSeconds(2))
10                     .filter(throwable -> throwable instanceof ServerException)
11                     .onRetryExhaustedThrow((retryBackoffSpec, retrySignal) -> {
12                         throw new ServerException("External Service failed to
       process after max retries", HttpStatus.SERVICE_UNAVAILABLE.value());
13                     }))
14             .publishOn(Schedulers.boundedElastic())
15             .block();
16 }
```

Kode 3.11: Metode getVatById

Metode *getVatListByVatIds* digunakan untuk mengambil data beberapa objek VAT berdasarkan *list* id. Metode ini mengirimkan *HTTP request* ke *endpoint* "/master-service/value-added-tax/value-added-tax-names-ids" disertai dengan *list* id yang dikirimkan sebagai *query parameter*. Metode ini menerima *HTTP response* dalam format JSON dan diubah menjadi bentuk *List ValueAddedTaxNameDTO*.

```
1  public List<ValueAddedTaxNameDto> getVatListByVatIds(Optional<HttpHeaderDto>
       httpHeaderDto, List<Long> vatIds) throws ClientException, ServerException {
2      return internalWebClient.get()
3          .uri(masterServiceVatNameListIds, uriBuilder -> uriBuilder.queryParam("
       vatIds", vatIds).build())
4          .accept(MediaType.APPLICATION_JSON)
5          .retrieve()
6          .bodyToMono(new ParameterizedTypeReference<List<ValueAddedTaxNameDto>>()
       {})
7          .contextWrite(context -> context.put(InternalWebFluxConfiguration.
       CONTEXT_HTTPHEADERDTO, httpHeaderDto.get()))
8          .retryWhen(Retry.backoff(3, Duration.ofSeconds(2))
9              .filter(throwable -> throwable instanceof ServerException)
10             .onRetryExhaustedThrow((retryBackoffSpec, retrySignal) -> {
```

```
11                  throw new ServerException("External Service failed to process
      after max retries", HttpStatus.SERVICE_UNAVAILABLE.value());
12              }))
13          .publishOn(Schedulers.boundedElastic())
14          .block();
15  }
```

Kode 3.12: Metode getVatListByVatIds

B Submodul Quotation

B.1 Analisis Masalah

Submodul *Quotation* merupakan submodul yang merepresentasikan objek *quote* pada proses bisnis. Objek *Quotation* memiliki *child* dengan hubungan *ManyToOne* yaitu *QuotationDetail* (satu *Quotation* bisa memiliki beberapa *QuotationDetail*), yang menyimpan data berupa *list* barang dalam *quote*.

Submodul *Quotation* belum melakukan perhitungan PPn berdasarkan nilai PPn dari *Value Added Tax* pada bagian *QuotationDetail*, melainkan menggunakan nilai yang di-*hardcode*, baik pada perhitungan tampilan nominal PPn pada *frontend* maupun perhitungan nominal PPn pada *backend*.

Pada tampilan *frontend* halaman *create Quotation*, pilihan persentase PPn pada bagian *QuotationDetail* masih berupa pilihan antara PPn 11% dan tanpa PPn, dan perlu diubah pilihannya menjadi pilihan dari nilai persentase PPn yang diambil dari VAT. Perhitungan nominal yang ditampilkan juga disesuaikan dengan nilai PPn yang diambil dari VAT. Demikian pula pilihan persentase PPn pada halaman *update* dan tampilan persentase PPn pada halaman *view detail* perlu disesuaikan dengan nilai PPn yang diambil dari VAT.

Pada bagian *backend*, dibutuhkan penyesuaian struktur database, *entity*, dan DTO untuk menyimpan id VAT yang akan digunakan dalam perhitungan, serta penyesuaian pada logika perhitungan PPn dan nominal keseluruhan di *service*. Selain itu, dibutuhkan perubahan tipe data menjadi *BigDecimal* yang dilakukan terhadap semua properti pada *Quotation* dan *QuotationDetail* yang menggunakan *float* dan *double*. *BigDecimal* merupakan *class* yang disediakan Java sebagai tipe data *number* yang dapat menangani angka dengan presisi tinggi, dengan kemampuan untuk menangani banyak angka di belakang koma.

Penambahan kolom selain id VAT juga perlu dilakukan pada *QuotationDetail* dengan menambahkan kolom untuk *discount4* dan *discount5*.

Penambahan kolom ini juga disertai dengan perubahan pada tampilan dan perhitungan *frontend* serta pada perhitungan diskon pada *backend*. Properti diskon ini merupakan diskon berlapis yang diterapkan pada objek *QuotationDetail*.

B.2 Solusi

Modifikasi *backend* dilakukan terlebih dahulu untuk memastikan aplikasi dapat menyimpan id VAT, diskon4, dan diskon5 pada objek *QuotationDetail* dan agar perhitungan PPn serta nominal keseluruhan menggunakan nilai persentase PPn yang diambil dari VAT, diskon4, dan diskon5.

Perubahan struktur tabel dilakukan dengan membuat *file* migrasi berisi *query* SQL untuk menambahkan dan memodifikasi kolom-kolom yang dibutuhkan. Perubahan tabel yang dilakukan pada tabel *quotation_detail* mencakup penghapusan kolom *ppn_percentage* (sebelumnya digunakan untuk menyimpan nilai PPn) serta penambahan kolom-kolom berikut:

- *vat_id* (BIGINT): Digunakan untuk menyimpan id VAT yang digunakan dalam perhitungan PPn.
- *discount_4* (DECIMAL (5,2)): Digunakan untuk menyimpan persentase diskon 4 jika tipe diskon berupa persen.
- *discount_5* (DECIMAL (5,2)): Digunakan untuk menyimpan persentase diskon 5 jika tipe diskon berupa persen.
- *discount_4_amount* (DECIMAL (15,2)): Digunakan untuk menyimpan jumlah diskon 4 dalam Rupiah jika tipe diskon berupa Rupiah.
- *discount_5_amount* (DECIMAL (15,2)): Digunakan untuk menyimpan jumlah diskon 5 dalam Rupiah jika tipe diskon berupa Rupiah.
- *discount_4_type* (SMALLINT): Digunakan untuk menyimpan tipe diskon 4.
- *discount_5_type* (SMALLINT): Digunakan untuk menyimpan tipe diskon 4.

Selaras dengan modifikasi struktur tabel, *entity* dan DTO *QuotationDetail* juga perlu disesuaikan. Properti *ppnPercentage* dihapus, properti *vatId, discount4, discount5, discount4Amount, discount5Amount, discount4Type,* dan *discount5Type* ditambahkan pada *entity QuotationDetail* dan *QuotationDetailDTO*. Sementara untuk *QuotationDetailCreateDTO* dan *QuotationDetailUpdateDTO* ditambahkan

properti *discount4, discount5, discount4Amount, discount5Amount, discount4Type,* dan *discount5Type*. *QuotationDetailMapper* juga dimodifikasi untuk menyesuaikan dengan perubahan properti pada *QuotationDetailDTO* dan *QuotationDetail*.

Modifikasi perhitungan *backend* dilakukan pada bagian *QuotationService*, secara spesifik pada proses *create* dan *update*. Nilai persentase PPn yang digunakan dalam perhitungan nominal akhir diubah dari nilai yang diambil dari *ppnPercentage* menjadi nilai *ppnValue* yang diambil dari objek VAT. Objek VAT didapatkan menggunakan *SellingVatWebClient* dengan *id* yang didapatkan dari properti *vatId*.

```
1 BigDecimal ppnValue = getPpnValueByVatId(quotationDetail.getVatId
    ());
2 totalPPN = totalPPN.add(getPpnPphAmount(totalAfterDiscount,
    ppnValue));
```

Kode 3.13: Kode contoh penerapan getPpnValueByVatId

```
1 private BigDecimal getPpnPphAmount(BigDecimal totalAfterDiscount,
     BigDecimal percentage)\{
2 return totalAfterDiscount.multiply(percentage.divide(BigDecimal.
    valueOf(100)));
3 \}
```

Kode 3.14: Kode getPpnPphAmount

```
1 private BigDecimal getPpnValueByVatId(Long vatId) throws
    ClientException, ServerException \{
2 return sellingVatWebclient.getVatById(httpHeaderService.
    getHttpHeader(), vatId).getPpnValue();
3
4 \}
```

Kode 3.15: Kode getPpnValueByVatId

Potongan kode 3.13 merupakan contoh penggunaan *ppnValue* sebagai persentase untuk menghitung hasil akhir PPn. Potongan kode 3.14 adalah metode *getPpnPphAmount* yang digunakan untuk menghitung hasil pajak berdasarkan persentase PPn atau PPh. Potongan kode 3.15 merupakan metode dalam QuotationService yang berfungsi mengambil nilai persentase PPn berdasarkan VAT, dengan menggunakan *SellingVatWebClient*.

Perhitungan nilai diskon dan nominal akhir juga disesuaikan dengan penambahan diskon 4 dan diskon 5. Perubahan lain yang dilakukan pada *backend* adalah perubahan properti-properti dengan tipe data *double* dan *float* (baik primitif maupun *Java class*) menjadi *BigDecimal*. Perubahan properti dapat dilihat pada gambar 3.26 dan 3.27.

Properti yang diubah menjadi *BigDecimal* dalam *Quotation* adalah sebagai berikut:

- *discount1*
- *discount2*
- *discount3*
- *discount1Amount*
- *discount2Amount*
- *discount3Amount*
- *subTotalAmount*
- *totalPpnAmount*
- *totalPphAmount*
- *customerDiscountAmount*
- *itemDiscountAmount*
- *grandTotalAmount*

Properti yang diubah menjadi *BigDecimal* dalam *QuotationDetail* adalah sebagai berikut:

- *quantity*
- *price*
- *pphPercentage*
- *discount1*
- *discount2*
- *discount3*
- *discount4*
- *discount5*

- *discount1Amount*
- *discount2Amount*
- *discount3Amount*
- *discount4Amount*
- *discount5Amount*
- *discountAmount*
- *qdSodCreatedQty*
- *qdSodRemainingQty*
- *formQuantity*
- *priceDisplay*

```
-    private Float discount1;                    39  +    private BigDecimal discount1;
-    private Float discount2;                    40  +    private BigDecimal discount2;
-    private Float discount3;                    41  +    private BigDecimal discount3;
-    private Integer discount1Type;              42  +    private Short discount1Type;
-    private Integer discount2Type;              43  +    private Short discount2Type;
-    private Integer discount3Type;              44  +    private Short discount3Type;
-    private Double discount1Amount;             45  +    private BigDecimal discount1Amount;
-    private Double discount2Amount;             46  +    private BigDecimal discount2Amount;
-    private Double discount3Amount;             47  +    private BigDecimal discount3Amount;
```

Gambar 3.26. Tangkapan layar *diff* (perubahan) pada *Quotation*

Gambar 3.27. Tangkapan layar *diff* (perubahan) pada *Quotation Detail*

Modifikasi *frontend* dilakukan untuk memperbarui tampilan sehingga pemilihan dan perhitungan PPn dilakukan berdasarkan nilai yang diambil dari VAT, dan kolom-kolom untuk diskon ditambahkan. Perubahan pertama adalah pembaruan *type* pada VAT dan *Quotation*. Pada bagian VAT, ditambahkan *type* *ValueAddedTaxName* berikut:

```
export type ValueAddedTaxName = {
  id: number;
  vatName: string;
  ppnValue: number;
  isSelling: boolean;
  isPurchasing: boolean;
```

```
};
```

Kode 3.16: Definisi Type ValueAddedTaxName

Pada bagian *Quotation*, *type* yang berfungsi sebagai representasi data-data *QuotationDetail* ditambahkan properti yang dibutuhkan untuk data diskon 4, diskon 5, dan *id* VAT. *API Hook* baru dibuat untuk menangani pengambilan data-data VAT yang dibutuhkan untuk digunakan dalam tampilan dan perhitungan PPn pada *QuotationDetail*. Terdapat dua *API hook* baru yaitu get-value-added-tax-names-ids yang berfungsi mengirimkan *request* ke *endpoint* "/master-service/value-added-tax/value-added-tax-names-ids" untuk mengambil nama-nama VAT berdasarkan list id, dan get-value-added-tax-names yang berfungsi mengirimkan *request* ke *endpoint* "/master-service/value-added-tax/value-added-tax-names" untuk mengambil nama-nama VAT secara keseluruhan.

```
type GetValueAddedTaxNamesIdsOptions = {
    vatIds?: number[];
};

export const getValueAddedTaxNamesIds = async ({
    vatIds
}: GetValueAddedTaxNamesIdsOptions): Promise<Array<ValueAddedTaxName>> => {
    return getApiClient().get(
      '/master-service/value-added-tax/value-added-tax-names-ids',
      {
        params: {
          vatIds: vatIds ? vatIds.toString() : '',
        },
      }
    );
};

export const useValueAddedTaxNamesIds = ({
    vatIds
}:GetValueAddedTaxNamesIdsOptions) => {
    const {
        data,
        error,
        isSuccess,
        isError,
        isFetching,
        isFetched,
    } = useQuery({
        queryKey: ['master.valueAddedTaxes', 'valueAddedTaxNamesIds'],
        queryFn: () => getValueAddedTaxNamesIds({ vatIds }),
        enabled: !!vatIds && vatIds.length > 0,
        initialData: Array<ValueAddedTaxName>,
        initialDataUpdatedAt: Date.now() - 600000,
        staleTime: MASTER_STALE_TIME,
    });
```

```
36
37      return {
38          data,
39          error,
40          isLoading: isFetching && !isFetched,
41          isSuccess,
42          isError,
43      };
44  };
```

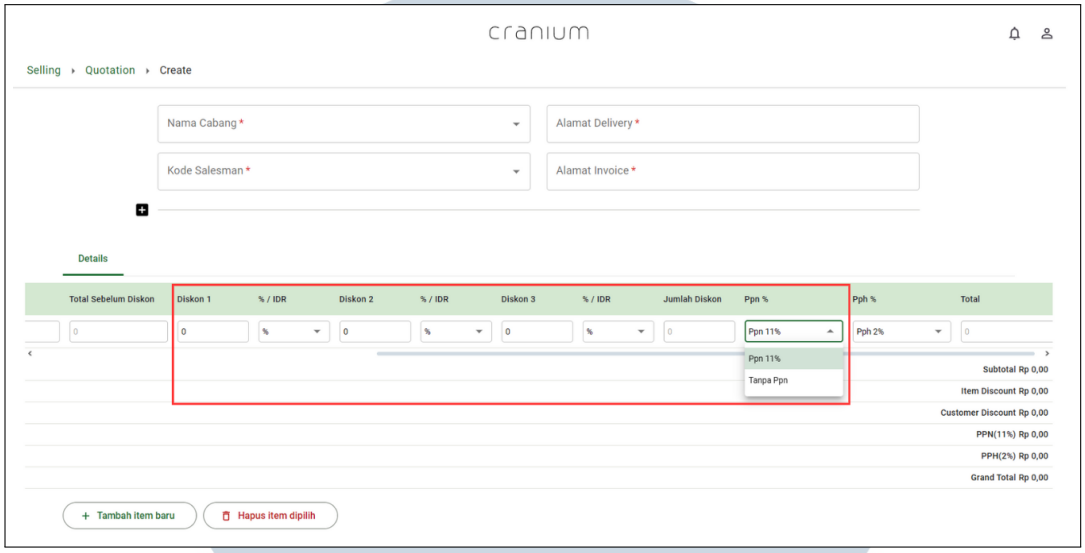
Kode 3.17: API Hook untuk Get Value Added Tax Names by IDs (TypeScript)

```
1   export const getValueAddedTaxNames = async (): Promise<Array<ValueAddedTaxName
        >> => {
2       return getApiClient().get(
3         '/master-service/value-added-tax/value-added-tax-names'
4       );
5   };
6
7   export const useValueAddedTaxNames = () => {
8       const {
9           data,
10          error,
11          isSuccess,
12          isError,
13          isFetching,
14          isFetched,
15      } = useQuery({
16          queryKey: ['master.valueAddedTaxes', 'valueAddedTaxNames'],
17          queryFn: () => getValueAddedTaxNames(),
18          initialData: Array<ValueAddedTaxName>,
19          initialDataUpdatedAt: Date.now() - 600000,
20          staleTime: MASTER_STALE_TIME,
21      });
22
23      return {
24          data,
25          error,
26          isLoading: isFetching && !isFetched,
27          isSuccess,
28          isError,
29      };
30  };
```

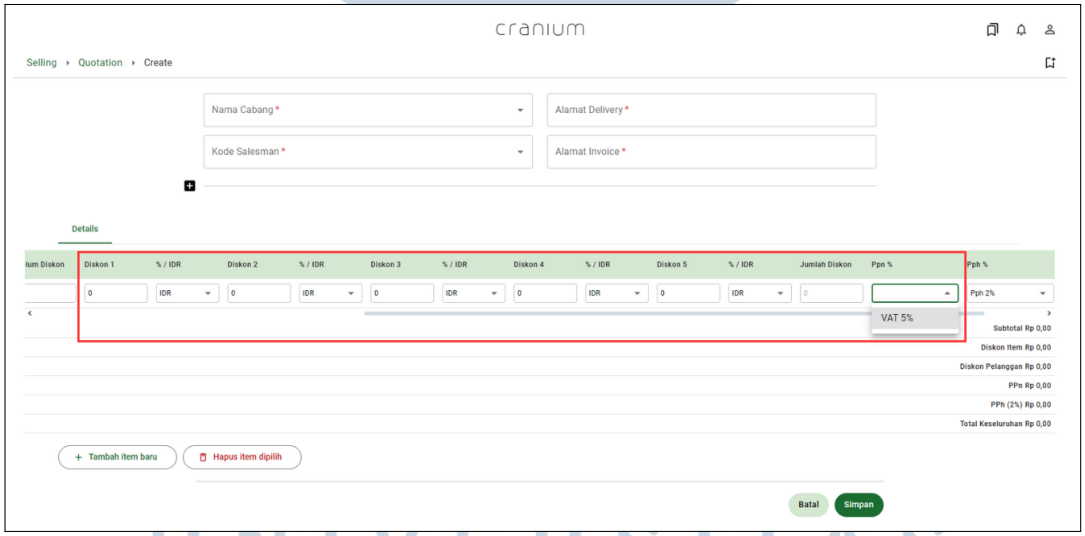
Kode 3.18: API Hook untuk Get Value Added Tax Names (TypeScript)

Pada tampilan *create* dan *update*, perubahan yang dilakukan adalah menambahkan kolom pada tabel *QuotationDetail* untuk diskon 4 dan diskon 5, disertai dengan pilihan tipenya, dan mengubah isi pilihan *dropdown* persentase ppn menjadi nama-nama VAT yang tersedia. Selain itu, tampilan perhitungan juga disesuaikan untuk menggunakan nilai persentase dari VAT yang dipilih oleh *user* serta disesuaikan dengan penambahan perhitungan diskon 4 dan diskon 5. Gambar

3.28 dan 3.29 menunjukkan perubahan yang dilakukan terhadap tampilan *frontend* tabel *QuotationDetail*.

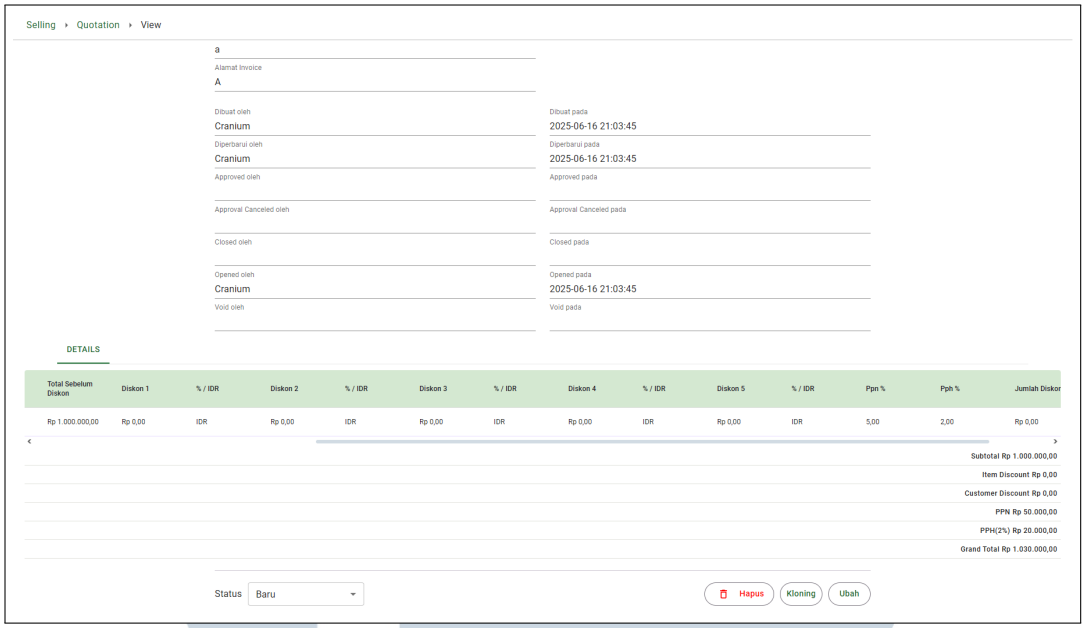


Gambar 3.28. Tampilan form *create Quotation* sebelum perubahan



Gambar 3.29. Tampilan form *create Quotation* setelah perubahan

Pada tampilan *detail view*, perubahan yang dilakukan adalah menambahkan kolom pada tabel *QuotationDetail* untuk menampilkan diskon 4 dan diskon 5 beserta tipenya, dan mengubah kolom persentase PPn sehingga menampilkan persentase VAT yang digunakan oleh *QuotationDetail* terkait. Gambar 3.30 menunjukkan tampilan yang sudah diubah. File-file *testing*/pengujian pada *backend* dan *frontend* juga disesuaikan dengan perubahan-perubahan yang dilakukan.



Gambar 3.30. Tampilan halaman detail *Quotation* setelah perubahan

C Submodul Delivery Order

C.1 Analisis Masalah

Terdapat beberapa *bug* dalam submodul *Delivery Order* yang membutuhkan perbaikan, yaitu:

- *Dropdown Warehouse* pada form *create* seharusnya di-filter berdasarkan *Branch* yang dipilih pada form.
- *Dropdown Status* pada tampilan *view detail* seharusnya menampilkan opsi "Approve" dan "Void", bukan "Approve" dan "New".
- Form *update* tidak dapat di-*submit* walaupun data sudah terisi lengkap.

Selain itu, ada penambahan fitur pada submodul ini, yaitu fitur *fulfillment*. *Delivery Order* dibuat berdasarkan data *Order*, dan memiliki hubungan *OneToMany* (satu *Order* bisa memiliki beberapa *DeliveryOrder*). *DeliveryOrder* merepresentasikan pengiriman untuk *Order* yang dibuat, dan *DeliveryOrder* hanya dapat dibuat berdasarkan suatu *Order* tertentu. Sistem *fulfillment* diterapkan sehingga *Delivery Order* untuk suatu *Order* tidak dapat dibuat lagi apabila sudah memenuhi kriteria *fulfilled*, yaitu kuantitas/*quantity Delivery Order* sudah memenuhi kuantitas yang diminta dalam *Order*.

C.2 Solusi

Perubahan yang perlu dilakukan untuk memperbaiki *bugs* di atas sepenuhnya merupakan perubahan *frontend*. Perbaikan untuk memunculkan data *Warehouse* hanya berdasarkan *Branch* yang dipilih memerlukan perubahan pada komponen form *create-deliveryorder-form* (*header form*) dan *create-deliveryorderdetail-form* (*child form*). *Branch* dipilih pada *header form* dan *branchId* yang dipilih di-*passing* ke *child form* dan digunakan untuk melakukan *filtering* terhadap data *Warehouse* yang didapatkan dari *API*.

```
const warehouseNames = useWarehouseName();
const filteredWarehouses = React.useMemo(
  () =>
    branchId
      ? warehouseNames.data.filter((warehouse) =>
   warehouse.branchId === branchId)
      : [],
  [warehouseNames.data, branchId]
);
```

Kode 3.19: Filtering Warehouse

Kolom *dropdown Warehouse* pada *Delivery Order Detail* juga diperbaharui sehingga menggunakan *Warehouse* yang sudah di-*filter*. Kode 3.20 menunjukkan perubahan kode *dropdown Warehouse*. *Dropdown* status pada tampilan *detail* memerlukan perubahan pada opsi-opsi status yang didefinisikan pada *utility file* status.ts. Kode 3.21 menunjukkan perubahan kode *dropdown* status.

```
<TableCell className={style.detailTableCell160}>
  <Controller
    name={`deliveryOrdersDetailCreateDtoList.${index}.masterWarehouseId`}
    control={control}
    rules={{
      required: t('common.validation', { ns: 'common' }),
    }}
    render={({ field }) => {
      const { onChange, value } = field;
      const selectedWarehouse = warehouseOptions.find(warehouse => warehouse.id
     === value) || null;
      return (
        <Autocomplete
          fullWidth
          sx={autocompleteStyle}
          options={warehouseOptions}
          getOptionLabel={(option) => option.warehouseName}
```

```
            value={selectedWarehouse}
            renderInput={(params) => (
              <TextField
                {...params}
                required
                id={style.detailTableTextFieldId}
                data-testid={`deliveryOrderDetail-warehouse-${index}`}
                error={Boolean(
                  errors.deliveryOrdersDetailCreateDtoList?.[index]?.
    masterWarehouseId
                )}
              />
            )}
            isOptionEqualToValue={(option, value) =>
              option.id === value.id
            }
            onChange={(_, newValue) => {
              onChange(newValue ? newValue.id : null);
            }}
          />
        );
      }}
    />
  </TableCell>
```

Kode 3.20: Delivery Order Warehouse Select (React/TypeScript)

```
<Select
  className={styles.statusLabelSelect}
  fullWidth
  MenuProps={menuSelectStatus}
  onChange={(e) => {
    handleOpenModal({
      modalName: 'status',
    });
    setValueStatus(
      e.target.value as number
    );
  }}
  data-testid="status-select"
  labelId="select-status"
  value={valueStatus}
>
  {status === SellingStatusEnum.NEW && (
    <MenuItem value={SellingStatusEnum.NEW} disabled>
      {t('common.status.new', { ns: 'common' })}
    </MenuItem>
  )}
  {GetUpdateStatusOptions(status)
    .filter((option) => option.value !== SellingStatusEnum.NEW)
    .map((status) => (
      <MenuItem value={status.value} key={status.value}>
        {status.name}
      </MenuItem>
```

```
      ))}
  </Select>
```

Kode 3.21: Komponen Status Select (React/TypeScript)

GetUpdateStatusOptions mendefinisikan pilihan-pilihan status, dan dideklarasikan dalam *file* status.ts. *GetUpdateStatusOptions* digunakan dalam beberapa komponen di submodul yang berbeda, tidak hanya oleh *detail view Delivery Order*, sehingga perubahan ini mempengaruhi opsi-opsi status untuk komponen di submodul lainnya.

```
export const GetUpdateStatusOptions = (
  valueStatus: number
) => {
  const { t } = useTranslation(['common']);
  let selectMenu: SelectMenu[] = [];
  if (valueStatus === 1) {
    selectMenu = [
      {
        name: t('common.status.new', {
          ns: 'common',
        }),
        value: 1,
      },
      {
        name: t('common.status.approved', {
          ns: 'common',
        }),
        value: 20,
      },
      {
        name: t('common.status.void', {
          ns: 'common',
        }),
        value: 30,
      },
    ];
  } else if (valueStatus === 10 || valueStatus === 20) {
    selectMenu = [
      {
        name: t('common.status.approved', {
          ns: 'common',
        }),
        value: 20,
      },
      {
        name: t('common.status.approved_canceled', {
          ns: 'common',
        }),
        value: 10,
      },
      {
        name: t('common.status.approved_closed', {
```

```
43              ns: 'common',
44            }),
45            value: 25,
46          },
47          {
48            name: t('common.status.void', {
49              ns: 'common',
50            }),
51            value: 30,
52          },
53        ];
54      } else if (valueStatus === 30) {
55        selectMenu = [
56          {
57            name: t('common.status.void', {
58              ns: 'common',
59            }),
60            value: 30,
61          },
62        ];
63      }
64      else if (valueStatus === 25) {
65        selectMenu = [
66          {
67            name: t('common.status.approved_closed', {
68              ns: 'common',
69            }),
70            value: 25,
71          },
72        ];
73      }
74
75      return selectMenu;
76    };
```

Kode 3.22: Deklarasi GetUpdateStatusOptions dalam status.ts (TypeScript)

Perbaikan yang dilakukan untuk membuat form *update* bisa di-*submit* adalah penambahan variabel *driverId* dan *vehicleId* pada *defaultValues* di form *update*. Kurangnya kedua variabel tersebut membuat data *default* pada form tidak lengkap sehingga tanpa adanya *user input*, tidak ada data *vehicle* dan *driver* yang dikirimkan oleh form. Penambahan ini memastikan ketika form dibuka, *user* tidak perlu memasukkan ulang data *driver* dan *vehicle* agar form dapat di-*submit*, karena sudah di-*set* terlebih dahulu nilai asal untuk *driverId* dan *vehicleId*.

Penambahan fitur *fulfillment* membutuhkan perubahan pada *backend* dan *frontend*, dengan mayoritas perubahan dilakukan pada *service* di *backend*. Objek *Orders* memiliki kolom data *order_do_fulfilled* (*sellingOrderDeliveryOrderFulfilled* pada *entity*) bertipe *boolean* yang berfungsi menandakan apabila suatu

Order sudah dipenuhi kriteria *fulfilled*-nya. *OrdersDetail* memiliki kolom data *od_dodetail_fulfilled* (*boolean*), *od_dodetail_create* (*number*), dan *od_dodetail_remain* (*number*) dengan representasinya masing-masing pada *entity*. Kolom *fulfilled* berfungsi menandakan apabila suatu *OrdersDetail* sudah dipenuhi oleh *DeliveryOrderDetail*, sementara *create* dan *remain* berfungsi menyimpan kuantitas yang sudah dibuat dalam *DeliveryOrderDetail* dan kuantitas yang tersisa atau belum dibuat. *OrdersDetail* akan bersifat *fulfilled* apabila kuantitasnya sudah terpenuhi seluruhnya oleh *DeliveryOrders* (*remain* = 0). *Orders* akan bersifat *fulfilled* apabila semua *OrdersDetail*-nya sudah berstatus *fulfilled*.

Pertama, dilakukan perubahan nama kolom dalam tabel *delivery_order_detail* menggunakan *file* migrasi baru, dengan kolom *detail_line_id* diubah menjadi *order_detail_id*. Properti *detailLineId* pada *entity DeliveryOrderDetail* diubah menjadi *orderDetail* bertipe *OrdersDetail*, merepresentasikan hubungan *ManyToOne* dengan *OrdersDetail* tersebut. Setiap DTO yang merepresentasikan data *DeliveryOrderDetail* juga ditambahkan properti *ordersDetailId* bertipe Long, dan *DeliveryOrderDetailMapper* juga ditambahkan *mapping* antara *ordersDetail.id* dari *DeliveryOrderDetail* dengan *ordersDetailId* dari *DeliveryOrderDetailDTO*.

Pada bagian *DeliveryOrderService*, logika *fulfillment* dibuat pada proses *create, update, delete,* dan *updateStatus* menjadi *void*. Setiap ada *DeliveryOrder* yang dibuat (*create*), aplikasi akan mengambil *quantity* pada masing-masing *DeliveryOrderDetail* kemudian memperbarui *quantity* dan *fulfillment* pada *OrdersDetail* yang berhubungan. Setelah *OrdersDetail* diperbarui, maka aplikasi akan memeriksa status *fulfillment* setiap *OrdersDetail* dan apabila semua *OrdersDetail* sudah *fulfilled*, maka *Orders* terkait juga akan diperbarui *fulfillment*-nya.

```
1  Boolean orderDeliveryDetailsFulfilled = order.getOrdersDetail()
2          .stream()
3          .allMatch(OrdersDetail::
       getSellingOrdersDetailDeliveryOrdersDetailFulfilled);
4
5  Double orderDeliveryDetailsCreatedQuantity = order.getOrdersDetail()
6          .stream()
7          .mapToDouble(OrdersDetail::
       getSellingOrdersDetailDeliveryOrdersDetailCreatedQuantity)
8          .sum();
9
10 Double orderDeliveryDetailsRemainingQuantity = order.getOrdersDetail()
11         .stream()
12         .mapToDouble(OrdersDetail::
       getSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity)
```

```
13          .sum();
14
15 order.setSellingOrderDeliveryOrderFulfilled(orderDeliveryDetailsFulfilled);
16 order.setSellingOrderDeliveryOrderTotalCreatedQuantity(
       orderDeliveryDetailsCreatedQuantity);
17 order.setSellingOrderDeliveryOrderTotalRemainingQuantity(
       orderDeliveryDetailsRemainingQuantity);
18 ordersRepository.save(order);
```

Kode 3.23: Kode pengecekan *fulfillment* untuk *Order* (*header object*)

```
1  if(ordersDetail.getSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity() > 0
       && !ordersDetail.getSellingOrdersDetailDeliveryOrdersDetailFulfilled()) {
2      if (deliveryOrdersDetail.getQuantity() > ordersDetail.
       getSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity()) {
3          throw new IllegalArgumentException(sellingMessageSource.getMessage(
       SELLING_DELIVERY_ORDER_DETAIL_QUANTITY_EXCEED, new Object[]{
       deliveryOrdersDetail.getQuantity(), ordersDetail.
       getSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity()},
       LocaleContextHolder.getLocale()));
4      }
5  } else {
6      if (deliveryOrdersDetail.getQuantity() > ordersDetail.getQuantity().
       doubleValue()) {
7          throw new IllegalArgumentException(sellingMessageSource.getMessage(
       SELLING_DELIVERY_ORDER_DETAIL_QUANTITY_EXCEED, new Object[]{
       deliveryOrdersDetail.getQuantity(), ordersDetail.getQuantity()},
       LocaleContextHolder.getLocale()));
8      }
9  }
10
11 ordersDetail.setSellingOrdersDetailDeliveryOrdersDetailCreatedQuantity(
12         ordersDetail.getSellingOrdersDetailDeliveryOrdersDetailCreatedQuantity()
       + deliveryOrdersDetail.getQuantity()
13 );
14 ordersDetail.setSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity(
15         ordersDetail.getQuantity().doubleValue() - ordersDetail.
       getSellingOrdersDetailDeliveryOrdersDetailCreatedQuantity()
16 );
17
18 if (ordersDetail.getSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity() <=
       0) {
19     ordersDetail.setSellingOrdersDetailDeliveryOrdersDetailRemainingQuantity(0.0)
       ;
20     ordersDetail.setSellingOrdersDetailDeliveryOrdersDetailFulfilled(true);
21 } else {
22     ordersDetail.setSellingOrdersDetailDeliveryOrdersDetailFulfilled(false);
23 }
24
25 ordersDetailRepository.save(ordersDetail);
```

Kode 3.24: Kode pengecekan *fulfillment* untuk *Order Detail* (*child object*)

Pada proses *update*, alur yang dilalui sama dengan *create*, yaitu dengan

pembaruan dan pengecekan *OrdersDetail* terlebih dahulu baru pembaruan dan pengecekan *Orders*. Status *fulfilled* dapat berubah dari *true* menjadi *false* maupun sebaliknya tergantung data yang sudah diperbarui. Pada proses *delete* dan *updateStatus* menjadi *void*, alur yang dilalui juga sama, namun dengan perbedaan pembaruan kuantitas pasti akan berupa pengurangan kuantitas dan status pasti akan berubah menjadi *false* untuk objek yang diperbarui.

D Submodul Order Return

D.1 Analisis Masalah

Sama seperti submodul *Quotation*, *Order Return* membutuhkan perubahan berupa implementasi nilai persentase yang diambil dari VAT dalam perhitungan PPn dan penambahan kolom diskon 4 dan diskon 5 pada *Order Return Detail*. Selain itu, semua data yang bertipe *float* atau *double* diubah menjadi *BigDecimal*.

D.2 Solusi

Perubahan yang dilakukan pada *backend* hampir sama dengan perubahan yang dilakukan pada *Quotation*. Secara garis besar, perubahan-perubahan pada *backend* yang berkaitan dengan VAT, diskon 4, dan diskon 5 adalah sebagai berikut:

- Penambahan kolom-kolom database untuk diskon 4, diskon 5 dan id VAT pada *OrderReturnDetail* menggunakan migrasi. Penghapusan kolom *ppn_percentage* dari *OrderReturnDetail*.
- Penyesuaian *entity*, DTO, dan *mapper* *OrderReturnDetail* dengan penambahan dan penghapusan kolom yang dilakukan.
- Penambahan *method* pada *service* untuk mengambil persentase PPn berdasarkan VAT menggunakan *webclient*.
- Penyesuaian perhitungan PPn dan nominal total pada *service* sehingga menggunakan persentase PPn yang diambil berdasarkan VAT.
- Data-data yang bertipe *float* dan *double* diubah menjadi *BigDecimal* pada *entity* dan DTO, dan perhitungan pada *service* disesuaikan dengan menggunakan metode-metode perhitungan *BigDecimal*.

Unit Test dan *contract test* disesuaikan dengan perubahan-perubahan terkait penambahan diskon 4, diskon 5, dan VAT, serta perubahan tipe data menjadi BigDecimal. Perubahan yang dilakukan pada *frontend* juga hampir sama dengan *Quotation*, yaitu sebagai berikut:

- Penambahan properti-properti terkait diskon 4, diskon 5, dan VAT pada *type OrderReturnDetail*.
- Modifikasi form *create* dan *update* yaitu penambahan kolom diskon 4 dan diskon 5 pada tabel detail, serta perubahan isi pilihan *dropdown* persentase PPn menjadi pilihan yang didapat dari VAT.
- Modifikasi tampilan detail *OrderReturn* sehingga menampilkan informasi diskon 4 dan diskon 5, serta menunjukkan nama VAT yang digunakan.

Perubahan lainnya pada *frontend* adalah penambahan *enumeration* untuk tipe diskon, sehingga tipe diskon tidak menggunakan nilai yang di-*hardcode* pada komponen *select*, melainkan nilai yang diambil dari *enum*. Potongan kode 3.25 merupakan deklarasi *enum* yang digunakan untuk tipe diskon. Potongan kode 3.26 menunjukkan contoh implementasi *enum* dalam pilihan tipe diskon.

```
export enum SellingOrderReturnDiscountTypeEnum {
    IDR = 1,
    PERCENTAGE = 100
}
```

Kode 3.25: Kode SellingOrderReturnDiscountTypeEnum

```
<MenuItem
    value={SellingOrderReturnDiscountTypeEnum.PERCENTAGE}
    className={style.detailText}
  >
    {t('selling.invoice.percent', {
      ns: 'selling',
    })}
  </MenuItem>
  <MenuItem
    value={SellingOrderReturnDiscountTypeEnum.IDR}
    className={style.detailText}
  >
    {t('selling.invoice.idr', {
      ns: 'selling',
    })}
```

```
16 </MenuItem>
```

Kode 3.26: Kode Implementasi SellingOrderReturnDiscountTypeEnum

E Submodul Delivery Order Return

E.1 Analisis Masalah

Perubahan yang perlu dilakukan dalam submodul *Delivery Order Return* berupa penambahan fitur *fulfillment*, sama seperti submodul *Delivery Order*. Objek *Delivery Order Return* dibuat berdasarkan suatu *Order Return*. Setiap *Delivery Order Return* akan berhubungan dengan satu *Order Return*, dengan hubungan satu *Order Return* bisa memiliki beberapa *Delivery Order Return* (*Many to One*). Objek detail untuk masing-masing yaitu *Order Return Detail* dan *Delivery Order Return Detail* juga memiliki relasi yang sama. Sistem *fulfillment* berfungsi menandakan apabila pengiriman untuk suatu *Order Return* sudah dibuat secara lengkap. Suatu objek *Order Return* akan dianggap *fulfilled* apabila telah dibuat *Delivery Order Return* yang memenuhi kuantitas yang diminta oleh *Order Return*.

E.2 Solusi

Langkah-langkah perbaikan untuk *Delivery Order Return* hampir sama dengan *Delivery Order,* yaitu sebagai berikut:

- Mendefinisikan properti pada *entity Delivery Order Return* dan DTO untuk menandakan relasi dengan *Order Return*.
- Menyesuaikan *mapper* dengan perubahan yang dilakukan pada *entity* dan DTO.
- Menambahkan kode untuk logika *fulfillment* pada *service*, dengan memperbarui kolom *fulfilled* pada *Order Return* dan detailnya ketika sudah ada *Delivery Order Return* yang dibuat
- Menambahkan *filtering* pada *frontend* sehingga *Order Return* yang ditampilkan ketika akan membuat *Delivery Order Return* baru hanya *Order Return* yang belum memenuhi kriteria *fulfilled*.
- Menyesuaikan *type* pada *frontend* dengan perubahan yang dilakukan terhadap DTO.

3.4 Kendala dan Solusi yang Ditemukan

Kendala yang ditemukan selama kerja praktik magang adalah sebagai berikut:

- Terdapat beberapa hal yang harus dipelajari lagi pada arsitektur *backend* maupun *frontend* yang digunakan untuk pengembangan ERP Cranium.
- Terdapat beberapa modul dan submodul yang metode pengembangan sebelumnya tidak selaras, sehingga implementasi fitur-fitur sejenis untuk masing-masing modul bisa berbeda.
- Terdapat beberapa tugas/pekerjaan yang mengharuskan *pull request* sebelumnya di-*merge* terlebih dahulu, sehingga ada tugas yang harus menunggu PR di-*merge*, hal ini membuat pengembangan berjalan sedikit lebih lama.

Solusi atas kendala-kendala yang dihadapi selama kerja praktik magang adalah sebagai berikut:

- Beradaptasi dengan lingkungan perusahaan dan arsitektur yang digunakan, serta belajar mandiri untuk hal-hal yang memerlukan pembelajaran kembali.
- Menyelaraskan proses-proses yang sudah ada dengan *best practice* yang ditemukan, atau melanjutkan pengerjaan submodul dengan metode yang disesuaikan.
- Rutin berdiskusi dengan anggota tim lainnya dan supervisor untuk memastikan tidak ada pekerjaan yang tertinggal jauh atau PR yang terlalu lama di-*merge*.