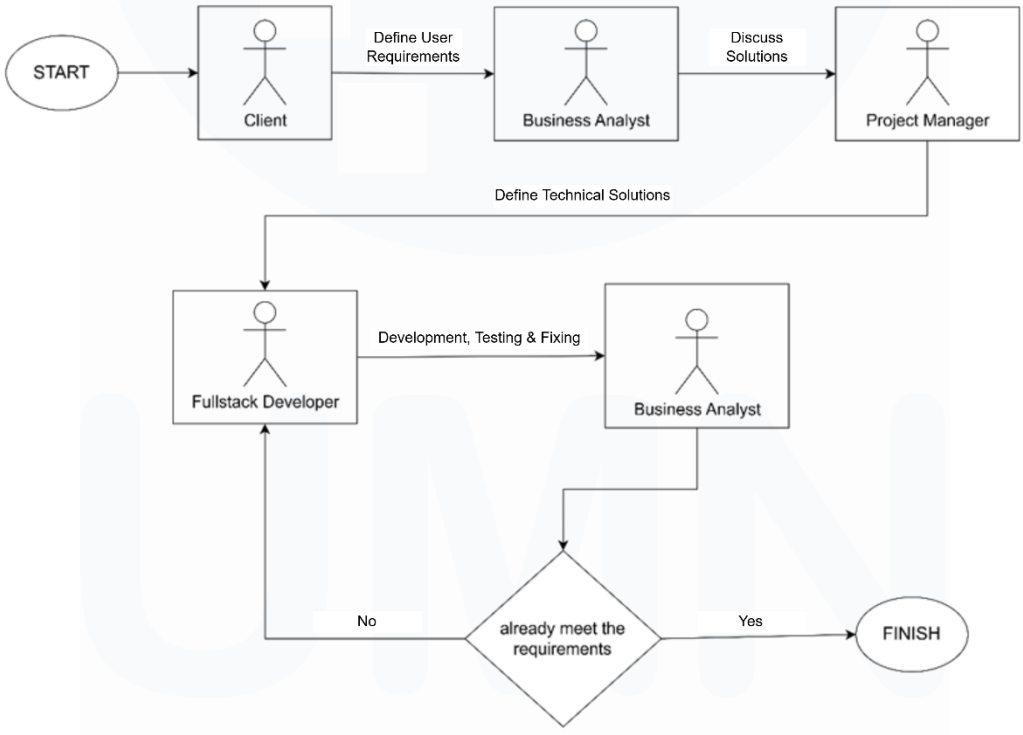
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Posisi Fullstack Developer Intern dalam proyek pengembangan Distributor Management System (DMS) di PT. Magna Solusi Indonesia melibatkan tanggung jawab dalam analisis kebutuhan bisnis, pengembangan sistem, hingga pemeliharaan teknis. Posisi ini berada di bawah supervisi langsung Senior Fullstack Developer (sebagai mentor teknis dan supervisor) dan Project Manager untuk memastikan kesesuaian hasil kerja dengan tujuan proyek.



Gambar 3.1 Alur Proses Koordinasi

Analisis kebutuhan bisnis klien dilakukan untuk menentukan spesifikasi teknis fitur, seperti membuat fitur monitoring data master. Proses ini melibatkan kolaborasi dengan Business Analyst dan Project Manager guna memastikan solusi

teknologi selaras dengan kebutuhan operasional klien. Hasil analisis kemudian diterjemahkan ke dalam desain fungsional dan dokumentasi teknis.

Proses koordinasi diawali dengan tahap START, di mana identifikasi kebutuhan (*requirements*) dari klien atau stakeholder dilakukan untuk menentukan ruang lingkup dan tujuan proyek. Pada fase ini, tim melakukan diskusi awal dengan Business Analyst dan Project Manager guna memastikan pemahaman yang sama terkait spesifikasi teknis dan prioritas tugas. Pembagian tugas kemudian direncanakan melalui *Sprint Planning* dengan menggunakan platform seperti Microsoft Teams untuk komunikasi dan manajemen tugas.

Setelah pengembangan sistem dimulai, tahap Already Meet the Requirements menjadi fase kritis untuk memvalidasi kesesuaian hasil kerja dengan kebutuhan yang telah ditetapkan. Contohnya, fitur *monitoring data lokasi* diuji melalui kolaborasi dengan tim *Business Analyst* untuk memastikan fungsionalitas sesuai ekspektasi. Proses ini melibatkan *testing* menyeluruh, identifikasi bug, serta perbaikan teknis. Koordinasi harian melalui *daily stand-up meeting* membantu tim tetap sinkron dalam melaporkan progres dan mengatasi hambatan.

Pada tahap FINISH, sistem yang telah memenuhi semua persyaratan diimplementasikan ke lingkungan produksi. Tahap ini mencakup *deployment* fitur, dokumentasi teknis, dan serah terima ke tim operasional untuk pemeliharaan berkelanjutan. Hasil kerja dipresentasikan dalam *Sprint Review* kepada stakeholder guna mendapatkan persetujuan akhir. Seluruh proses didukung oleh penggunaan tools seperti GitHub untuk kolaborasi kode, DBeaver untuk manajemen database, serta PostgreSQL sebagai penyimpanan data real-time.

Pengembangan sistem mencakup pembangunan antarmuka pengguna (*frontend*) menggunakan Angular dan logika bisnis (*backend*) dengan Spring Boot. Fitur seperti dashboard monitoring data master dirancang untuk meningkatkan akurasi dan kecepatan akses informasi. Integrasi dengan database PostgreSQL dilakukan untuk mendukung penyimpanan data real-time dan manajemen transaksi

distribusi. Pengujian sistem dilaksanakan untuk memvalidasi fungsionalitas fitur. Tahap ini mencakup identifikasi bug, perbaikan kesalahan kode, serta optimalisasi performa query database. Selain itu, dukungan teknis diberikan pasca-*deployment* untuk memastikan sistem berjalan stabil, seperti memperbaiki error pada fitur data master atau meningkatkan respons API.

koordinasi tim dilakukan melalui rapat mingguan untuk melaporkan progress pekerjaan dan hambatan teknis. Proses ini mencakup pembagian tugas berdasarkan prioritas, diskusi solusi teknis, serta presentasi hasil kerja di akhir periode pengembangan. Platform Microsoft Teams digunakan untuk komunikasi, sementara GitHub menjadi sarana kolaborasi kode dan manajemen versi.

Seluruh alur kerja bertujuan menciptakan sistem distribusi yang efisien, transparan, dan mudah diadaptasi. Dengan pendekatan kolaboratif dan penggunaan teknologi terkini, proyek ini diharapkan dapat memberikan solusi berkelanjutan bagi peningkatan manajemen rantai pasok klien.

3.2 Tugas dan Uraian Kerja Magang

Dalam industri pengembangan perangkat lunak, peran Fullstack Developer memiliki tanggung jawab pada dua sisi utama aplikasi, yaitu frontend dan backend. Di sisi frontend, pekerjaan mencakup pembuatan antarmuka pengguna yang interaktif, responsif, dan user-friendly dengan memanfaatkan framework seperti Angular, React, atau Vue. Di sisi backend, fokus utamanya adalah membangun logika bisnis, mengelola database, serta menyediakan REST API untuk komunikasi data menggunakan teknologi seperti Spring Boot, Node.js, atau .NET. Fullstack Developer juga berperan dalam mengintegrasikan kedua sisi aplikasi agar dapat saling berkomunikasi dengan lancar. Selain itu, pengujian fungsional, perbaikan bug, optimasi performa, serta pemeliharaan sistem menjadi bagian dari tanggung jawab yang penting untuk menjaga stabilitas dan keberlangsungan aplikasi.

Pada proyek pengembangan Distributor Management System (DMS) di PT. Magna Solusi Indonesia, tugas difokuskan pada pembuatan fitur berbasis mockup

yang telah dirancang dan disepakati oleh tim Business Analyst dan Project Manager. Mockup tersebut diimplementasikan ke dalam kode pada sisi frontend menggunakan Angular dan sisi backend menggunakan Spring Boot. Setelah masing-masing sisi selesai dikembangkan, dilakukan integrasi agar fitur berjalan sesuai dengan skenario bisnis yang diharapkan. Selain membangun fitur baru, beberapa pekerjaan lain mencakup perbaikan struktur tabel di database PostgreSQL, penyesuaian skema data, serta penyelesaian bug pada fitur yang telah ada. Seluruh fitur yang dikembangkan diuji secara menyeluruh untuk memastikan stabilitas, fungsionalitas, dan kesesuaian dengan kebutuhan pengguna.

Dalam mendukung pekerjaan sebagai fullstack developer, sejumlah tools digunakan untuk menunjang proses pengembangan perangkat lunak secara efektif dan efisien. Visual Studio Code (VS Code) digunakan sebagai editor utama untuk membuka dan mengelola proyek frontend yang dikembangkan dengan framework Angular. Sementara itu, NetBeans dimanfaatkan sebagai Integrated Development Environment (IDE) untuk pengembangan backend yang menggunakan java springboot. Git berperan penting sebagai sistem version control guna mengelola revisi kode sumber secara terstruktur dan kolaboratif. Selain itu, DBeaver digunakan untuk mengakses dan mengelola database PostgreSQL, yang menjadi basis data utama dalam proyek tersebut. Excel dimanfaatkan sebagai alat bantu untuk pengelolaan sprint project, termasuk pencatatan daftar pengujian, fitur yang perlu diperbaiki, serta fitur yang telah siap untuk diimplementasikan. Untuk mendukung aspek desain dan komunikasi antar tim, Figma digunakan sebagai platform untuk membuat serta meninjau mockup yang telah dikerjakan oleh tim UI/UX dan Business Analyst (BA). Penggunaan tools tersebut secara sinergis memastikan kelancaran proses pengembangan aplikasi serta mendukung koordinasi antar tim dalam proyek pengembangan Distributor Management System

3.2.1 Uraian Kegiatan Magang

Sebagai bagian dari program magang yang berlangsung di salah satu proyek pengembangan Distributor Management System (DMS), pelaksanaan kegiatan dibagi ke dalam tiga tahapan utama: tahap awal & analisis desain, tahap pengembangan aplikasi, serta tahap final dan dokumentasi. Pembagian ini bertujuan untuk memberikan alur kerja yang sistematis dan mendalam terhadap proses pengembangan perangkat lunak berbasis industri. Masing-masing tahapan memiliki kegiatan serta output yang berbeda, mulai dari pemahaman lingkungan kerja dan teknologi, implementasi fitur, hingga pengujian dan dokumentasi teknis. Tabel 2.1 merangkum seluruh kegiatan yang dilakukan selama magang beserta hasil yang dicapai pada setiap tahap.

Tahap Awal Magang & Analisis Desain Aplikasi		
Tahapan	Kegiatan	Output
Onboarding	Pengenalan lingkungan kantor dan belajar tech stack yang digunakan di kantor	Mengenal lingkungan kantor dan mendapatkan basic knowledge angular dan springboot
Analisis Awal	Meeting pertama tentang alur kerja aplikasi dan kebutuhan klien	Requirements Document
Tahap Pengembangan Aplikasi		
Tahapan	Kegiatan	Output
Penugasan Pertama	Mengerjakan Fitur CRUD Monitoring Incoterm	Halaman Monitoring yang sudah di uji
Melengkapi Fitur Monitoring Data Master	Mengerjakan CRUD untuk beberapa data master	
Membuat Template Report	Membuat Jasper report yang terhubung dengan backend aplikasi	PDF Report berdasarkan data yang
Migrasi Database	Memperbaiki database	Database yang sudah optimal
Membuat Authentication	Mengerjakan form registrasi user (company & distributor), halaman login, dan halaman monitoring user	Halaman monitoring, form registrasi dan login yang sudah di uji

Tahap Final & Dokumentasi		
Tahapan	Kegiatan	Hasil
Unit Test	Melakukan testing fitur CRUD dan tampilan pada aplikasi	Dokumen Unit Test
Fixing Bugs	Melakukan perbaikan kepada fitur yang terdapat bug	Fitur yang sesuai requirements
Dokumentasi	Melakukan dokumentasi pada bagian – bagian kode	Kode yang sudah didokumentasikan

Tabel 3.1 Uraian Kegiatan Magang

Pada tahap awal magang dan analisis desain aplikasi, peserta terlebih dahulu menjalani proses *onboarding* untuk mengenal lingkungan kantor serta memahami teknologi inti yang digunakan dalam proyek, yaitu Angular untuk frontend dan Spring Boot untuk backend. Proses ini memberikan bekal awal dalam memahami arsitektur sistem yang akan dikembangkan. Selanjutnya dilakukan *analisis awal* melalui pertemuan dengan tim untuk membahas alur kerja aplikasi serta kebutuhan klien. Output dari tahap ini berupa *Requirements Document* yang menjadi dasar pengembangan fitur.

Memasuki tahap pengembangan aplikasi, peserta mulai diberikan penugasan teknis. Tugas pertama adalah mengembangkan fitur *CRUD Monitoring Incoterm*, diikuti dengan pelengkapan fitur CRUD untuk berbagai data master lainnya. Selain itu, peserta juga membuat *template laporan* menggunakan JasperReport yang terhubung ke backend aplikasi untuk menghasilkan *PDF report* berbasis data transaksi. Tugas penting lainnya adalah melakukan *migrasi dan optimalisasi database*, serta membangun sistem *autentikasi* yang mencakup form registrasi, login, dan halaman monitoring pengguna. Semua fitur diuji untuk memastikan fungsionalitasnya berjalan sesuai rencana.

Pada tahap final dan dokumentasi, fokus kegiatan beralih pada *unit testing* terhadap fitur-fitur yang telah dikembangkan untuk memverifikasi kesesuaian dengan kebutuhan sistem. Setelah itu dilakukan *bug fixing* terhadap fitur yang

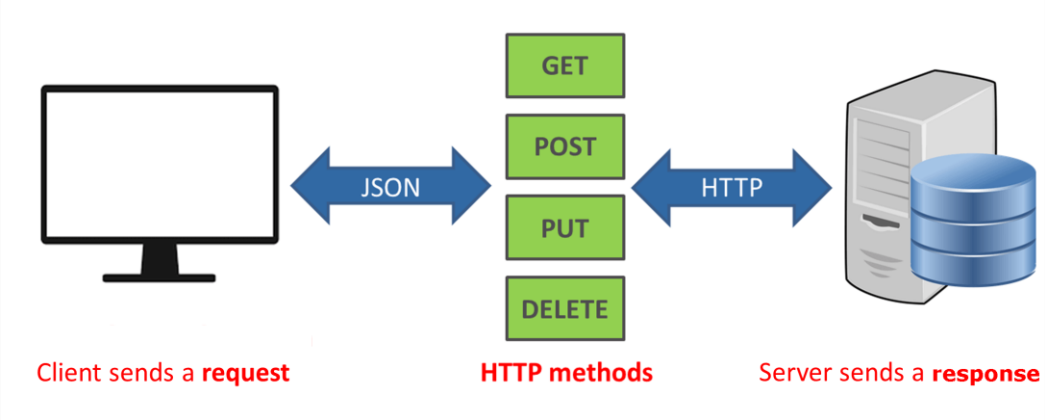
belum sesuai. Terakhir, seluruh kode yang telah dikembangkan dan diuji didokumentasikan secara sistematis agar dapat digunakan sebagai referensi dalam pemeliharaan dan pengembangan sistem di masa mendatang.

3.2.1.1 Tahap Awal & Analisis Desain Aplikasi

Pada hari pertama pelaksanaan magang, kegiatan dimulai dengan proses *onboarding* yang mencakup pengenalan sejarah perusahaan, struktur organisasi, serta perkenalan dengan staf dan karyawan yang bekerja di lingkungan kantor. Selain itu, dilakukan juga tur keliling kantor untuk memahami tata letak dan budaya kerja di lingkungan perusahaan. Setelah proses pengenalan, dilakukan pengaturan lingkungan kerja (*workspace*), yang mencakup konfigurasi koneksi jaringan internet untuk mengakses Virtual Private Server (VPS) perusahaan, pengaturan akun GitHub, serta instalasi dan konfigurasi perangkat lunak pengembangan seperti Visual Studio Code dan NetBeans.

Selama satu minggu pertama, kegiatan difokuskan pada pendalaman teknologi (*tech stack*) yang digunakan dalam pengembangan proyek *Distributor Management System* (DMS). Teknologi yang dipelajari meliputi *Angular* dengan *TypeScript* pada sisi *frontend*, serta *Spring Boot* berbasis bahasa pemrograman *Java* pada sisi *backend*.

Pada sisi *frontend*, dipelajari konsep dasar pengembangan aplikasi menggunakan *component-based architecture*, termasuk pembuatan *component*, *service*, serta implementasi *routing* untuk navigasi antar halaman. Selain itu, dipelajari pula teknik pengelolaan *state* lokal, penggunaan *Reactive Forms* untuk validasi data, serta pemanggilan *REST API* menggunakan *HttpClient*.



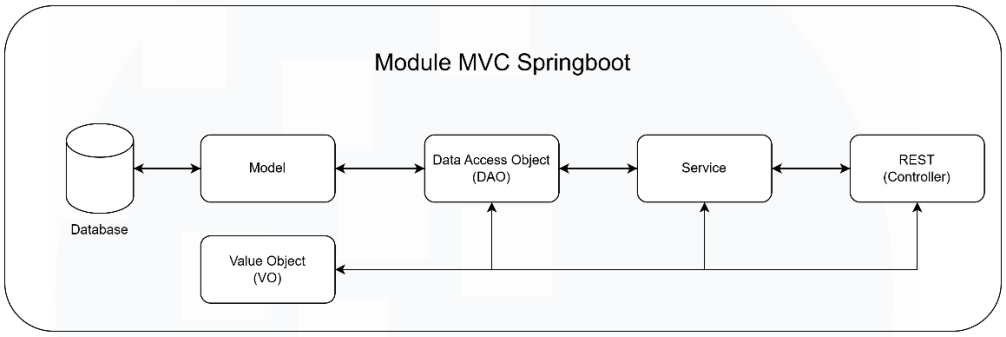
Gambar 3.2 RESP API Process [10]

Gambar 3.2 menggambarkan proses komunikasi data antara *client* dan *server* menggunakan metode HTTP, yang terdiri atas GET, POST, PUT, dan DELETE. Permintaan dari *client* dikirim dalam format JSON dan diproses oleh *server* melalui protokol HTTP. Setelah diproses, *server* mengembalikan *response* kepada *client*. Proses ini merupakan dasar dari arsitektur RESTful API yang umum digunakan dalam pengembangan aplikasi berbasis web. Setiap metode HTTP memiliki fungsi tersendiri dalam operasi manipulasi data, yaitu pengambilan (GET), penyimpanan (POST), pembaruan (PUT), dan penghapusan (DELETE). Pemahaman terhadap alur ini sangat penting dalam pengembangan sistem yang efisien dan terstruktur.

Pada sisi *backend*, materi yang dipelajari mencakup arsitektur *Model-View-Controller* (MVC) berbasis *Spring Boot*, yang melibatkan pembuatan *controller*, *service layer*, *repository (DAO)*, serta pemodelan entitas dengan *Java Persistence API* (JPA). Selain itu, dipelajari pula konsep *dependency injection*, manajemen koneksi basis data menggunakan *Spring Data JPA*, serta proses konfigurasi koneksi ke database PostgreSQL.

Gambar 3.3 menunjukkan arsitektur modular MVC (Model-View-Controller) pada aplikasi Spring Boot, yang terdiri dari lima komponen utama: Model, Value Object (VO), Data Access Object (DAO), Service, dan REST Controller.

Diagram ini menggambarkan alur data dan tanggung jawab masing-masing komponen dalam pengembangan sistem backend berbasis Java Spring.



Gambar 3.3 Module MVC Springboot

1. Model mewakili struktur data yang berkaitan langsung dengan tabel di dalam database. Kelas model ini biasanya diberi anotasi JPA seperti @Entity
2. DAO (Data Access Object) berfungsi sebagai lapisan yang mengelola operasi CRUD (Create, Read, Update, Delete) terhadap database melalui objek model. DAO melakukan interaksi langsung dengan database dan dapat berupa interface JpaRepository atau class yang diimplementasikan secara manual.
3. Service menjadi jembatan antara controller dan DAO. Lapisan ini berisi logika bisnis aplikasi, seperti validasi, pemrosesan data, atau aturan-aturan yang berlaku sebelum data dikirim ke atau dari database.
4. REST Controller Bertugas menangani permintaan (*request*) dari client dan mengirimkan respon (*response*) kembali. Controller memanggil service untuk memproses permintaan dan hasilnya dikembalikan ke client melalui format seperti JSON.
5. Value Object (VO) Digunakan sebagai representasi data yang lebih fleksibel dan tidak terikat langsung dengan entitas database. VO biasanya digunakan untuk membawa data antar lapisan (service ↔ controller) atau sebagai bentuk respons yang lebih terstruktur dan sesuai kebutuhan tampilan frontend.

Alur data pada gambar dimulai dari client yang mengirim permintaan ke REST Controller, diteruskan ke Service, kemudian DAO mengambil data dari Model yang terhubung ke Database. Jika diperlukan, Value Object (VO) digunakan untuk menyusun atau mengubah struktur data sebelum dikembalikan ke pengguna. Struktur modular seperti ini memisahkan tanggung jawab antar komponen, sehingga memudahkan proses pengembangan, pengujian, dan pemeliharaan sistem.

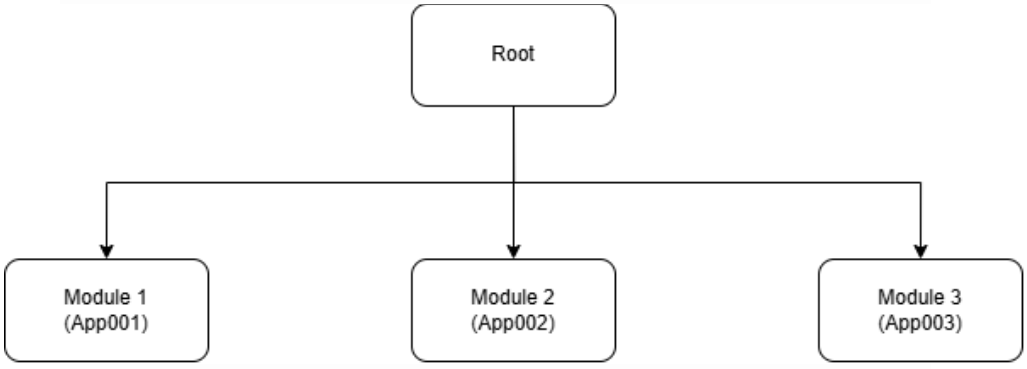
Melalui proses ini, diperoleh pemahaman mengenai pengembangan aplikasi berskala besar dengan pendekatan *microservices*, yang mendukung skalabilitas, modularitas, dan kemudahan dalam pemeliharaan sistem. Pendekatan ini juga memfasilitasi integrasi yang lebih fleksibel antara sisi *frontend* dan *backend* melalui komunikasi berbasis *RESTful API*.

Setelah itu, dilakukan pertemuan bersama dengan tim *Business Analyst* (BA) dan *Project Manager* (PM) dari pihak pengembang aplikasi untuk proyek *Distributor Management System* (DMS). Pertemuan ini bertujuan untuk memaparkan ruang lingkup proyek, menjelaskan alur kerja aplikasi, proses bisnis utama yang akan didigitalisasi, serta pembagian modul dan tugas dalam tim implementasi. Melalui proses ini, alur pengerjaan menjadi lebih terstruktur dan dapat dipantau dengan sistematis selama seluruh fase proyek.

Dari sesi pemaparan awal dijelaskan bahwa kebutuhan pengembangan sistem DMS ini dilatarbelakangi oleh sejumlah tantangan yang dihadapi oleh pihak pengguna, antara lain proses distribusi yang masih dilakukan secara manual, kurangnya visibilitas terhadap transaksi secara real time, serta kesulitan dalam mengelola data harga, diskon, promosi, dan penagihan yang tersebar di berbagai wilayah. Sistem yang digunakan sebelumnya belum mampu mendukung otomatisasi penuh serta belum terintegrasi dengan baik antar unit bisnis distribusi.

Sebagai solusi atas permasalahan tersebut, sistem DMS dirancang untuk mencakup berbagai modul utama seperti manajemen pesanan (*Order Management*), pengelolaan stok dan inventori, pengaturan promosi, pengelolaan wilayah distribusi, manajemen produk dan harga, penagihan pembayaran, serta pengelolaan tugas kunjungan penjualan. Sistem ini juga dilengkapi dengan fitur pelaporan dan dashboard yang memberikan visibilitas menyeluruh terhadap data transaksi dan kinerja di berbagai tingkatan distribusi. Desain sistem dibuat agar mampu mendukung struktur hierarki antar distributor, sub-distributor, dan pelanggan.

Dalam tahap perancangan awal, sistem diusulkan menggunakan pendekatan modular yang mendukung pengembangan secara fleksibel dan terpisah berdasarkan fungsionalitas masing-masing modul. Meskipun arsitektur *microservices* tidak disebutkan secara eksplisit dalam proposal, namun struktur modular yang digunakan memperlihatkan karakteristik sistem yang siap dipecah menjadi layanan-layanan terpisah, sehingga lebih mudah untuk dikelola, dikembangkan, dan diintegrasikan, termasuk dengan sistem ERP eksternal melalui format *CSV* atau *connector* khusus.



Gambar 3.4 Aplikasi Microservices Pada DMS

Gambar 3.4 menunjukkan struktur arsitektur aplikasi berbasis microservices pada sistem *Distributor Management System (DMS)*. Setiap modul dikembangkan sebagai layanan terpisah namun tetap terintegrasi di bawah satu proyek

utama (*root*). Pendekatan ini memudahkan pengelolaan, pengembangan, serta pemeliharaan sistem karena setiap modul memiliki tanggung jawab dan fungsionalitas yang spesifik.

1. Module 1 (App001) berfungsi sebagai pusat autentikasi dan manajemen pengguna (*user management*). Modul ini menangani proses login, otorisasi, pengelolaan peran (*role*), dan pengaturan akses aplikasi. Selain itu, App001 juga memuat konfigurasi umum atau *general settings* yang dibutuhkan oleh modul-modul lainnya.
2. Module 2 (App002) bertanggung jawab untuk mengelola data master dalam sistem. Modul ini mencakup entitas data utama seperti master produk, harga, distributor, dan konfigurasi wilayah distribusi. Modul ini mendukung kestabilan dan konsistensi data yang digunakan oleh modul lain.
3. Module 3 (App003) berfokus pada proses bisnis inti distribusi, yaitu pengelolaan Purchase Order (PO), Sales Order (SO), dan Delivery Order (DO). Modul ini menangani seluruh siklus transaksi mulai dari pemesanan hingga pengiriman barang, termasuk pencatatan status dan histori transaksi.

Struktur modular ini memungkinkan setiap aplikasi dikembangkan secara independen dan di-*deploy* secara terpisah, namun tetap dapat saling berkomunikasi melalui API. Hal ini sejalan dengan prinsip *microservices architecture*, yang memberikan skalabilitas dan fleksibilitas tinggi terhadap sistem.

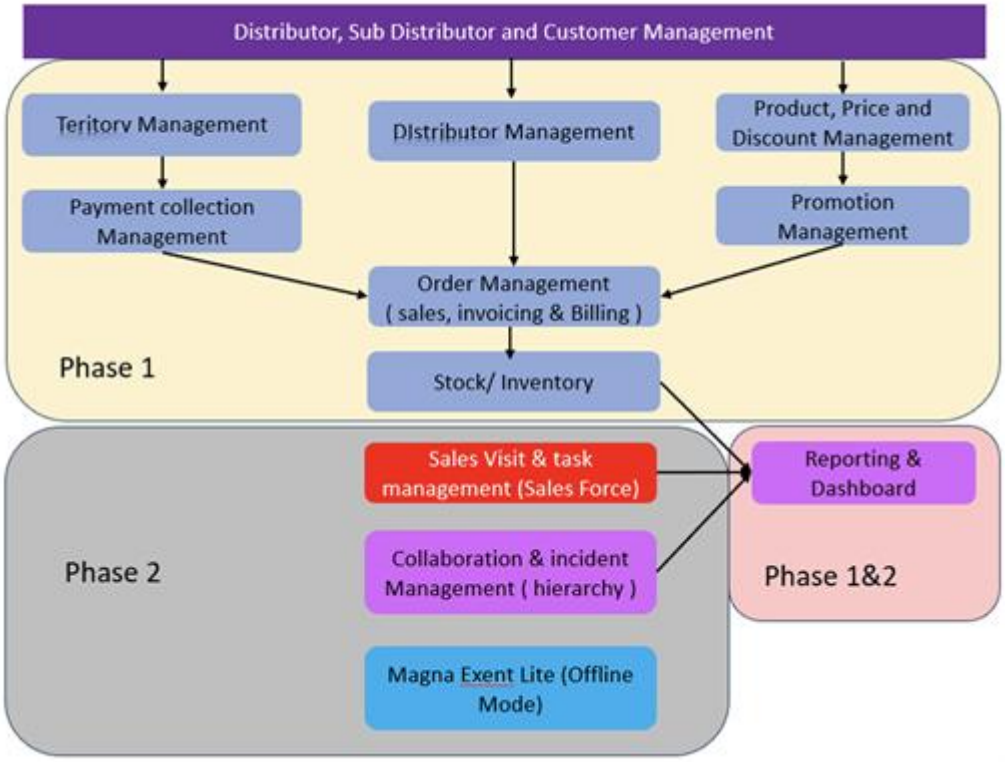
Secara garis besar, proses bisnis dalam aplikasi ini dimulai dari tahapan *onboarding* pelanggan dan distributor, pemrosesan pesanan melalui *sales order* dan *purchase order*, pengiriman barang, penagihan pembayaran, hingga pelaporan performa distribusi dan wilayah. Seluruh proses tersebut didukung oleh mekanisme persetujuan (approval) yang dinamis dan dapat dikonfigurasi sesuai kebutuhan organisasi pengguna.

Dalam tahapan analisis dan perancangan sistem, dilakukan penelaahan terhadap dokumen kebutuhan fungsional, pemetaan proses bisnis pengguna ke dalam alur sistem, serta identifikasi struktur data yang relevan pada masing-masing modul. Selain itu, dirancang pula alur komunikasi antar modul yang melibatkan integrasi antara *frontend*, *backend*, serta format pertukaran data antar sistem eksternal. Proses ini mencakup penyesuaian desain antarmuka pengguna dan model data agar sesuai dengan kebutuhan dan peran masing-masing pemangku kepentingan.

Tanggung jawab yang dijalankan dalam tahap ini meliputi dokumentasi kebutuhan sistem, identifikasi entitas data utama, penyusunan struktur modul, serta pembuatan diagram alur sistem berdasarkan hasil diskusi dengan tim teknis dan analis bisnis. Beberapa proses bisnis yang dianalisis antara lain *Order to Cash*, *Procure to Pay*, dan *Claim Management*, yang kemudian dijadikan acuan dalam pengembangan fitur pada fase berikutnya.

Gambar 3.5 memperlihatkan struktur integrasi aplikasi pada sistem *Distributor Management System (DMS)* yang dirancang secara modular dan diimplementasikan dalam dua fase utama. Setiap modul mewakili proses bisnis yang saling terhubung, mulai dari pengelolaan pelanggan hingga proses penjualan dan pelaporan. Pendekatan ini bertujuan untuk memastikan sistem dapat dikembangkan secara bertahap dan tetap memenuhi kebutuhan bisnis pengguna secara menyeluruh.

Pada Phase 1, sistem difokuskan pada implementasi modul-modul inti yang berkaitan langsung dengan proses distribusi. Modul utama dalam fase ini adalah *Distributor, Sub Distributor, and Customer Management* yang menjadi dasar untuk pengelolaan seluruh entitas distribusi. Modul ini berperan sebagai pusat data utama dan menjadi penghubung bagi semua proses lainnya.



Gambar 3.5 Integrasi Aplikasi

Beberapa modul pendukung seperti Territory Management dan Distributor Management digunakan untuk mengatur cakupan wilayah distribusi, serta struktur dan peran antar distributor. Modul ini terintegrasi dengan *Payment Collection Management* untuk mencatat dan mengelola proses penagihan berdasarkan wilayah dan pelanggan yang ditangani.

Selanjutnya, Order Management menjadi inti dari alur transaksi yang mengelola proses *sales order*, *invoicing*, hingga *billing*. Modul ini terhubung dengan Product, Price, and Discount Management serta Promotion Management guna memastikan bahwa pesanan yang dilakukan pelanggan sudah sesuai dengan harga dan promosi yang berlaku. Semua data transaksi kemudian akan tercatat dan diperbarui dalam Stock/Inventory Management untuk mencerminkan kondisi ketersediaan barang secara real-time.

Pada Phase 2, sistem diperluas untuk mencakup aktivitas lapangan dan komunikasi. Modul Sales Visit & Task Management mendukung proses kunjungan dan tugas tenaga penjualan melalui aplikasi mobile. Modul ini memungkinkan pencatatan aktivitas harian dan pelaporan informasi pasar secara langsung dari lapangan. Selanjutnya, Collaboration & Incident Management menyediakan platform komunikasi terpusat untuk menangani keluhan, permintaan perubahan, atau insiden yang dilaporkan oleh distributor, sub-distributor, atau pelanggan. Modul ini memperkuat sistem dalam aspek pelayanan dan dukungan teknis. Sebagai pelengkap, tersedia pula Magna Exent Lite, yaitu versi ringan dari DMS yang dapat beroperasi secara *offline*. Modul ini ditujukan untuk distributor di wilayah dengan keterbatasan koneksi internet. Transaksi akan disimpan secara lokal dan disinkronisasi secara otomatis saat koneksi tersedia.

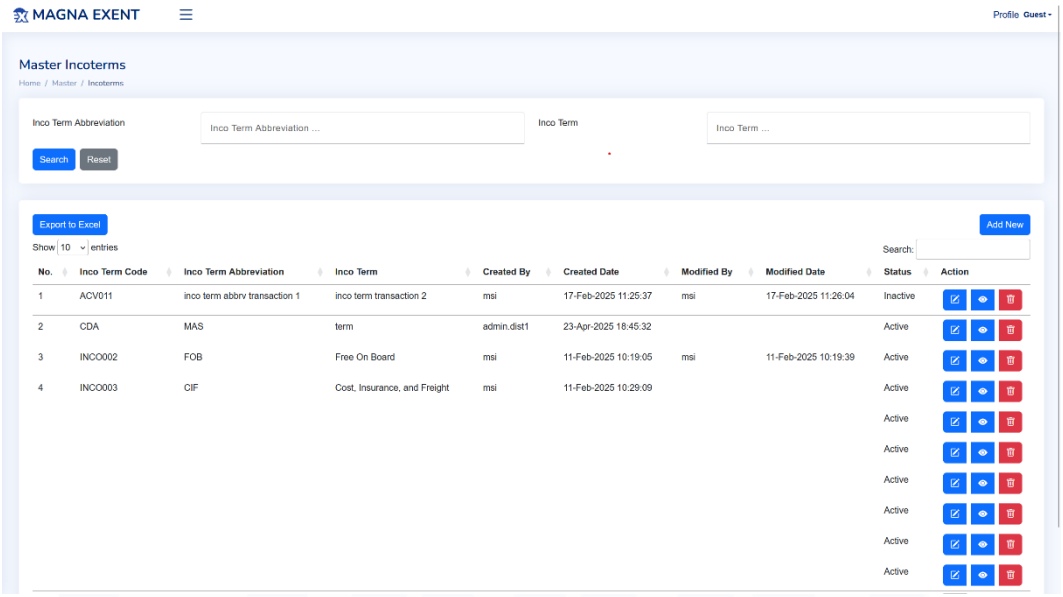
Terakhir, seluruh data dari Phase 1 dan Phase 2 akan dihimpun dalam Reporting & Dashboard, yang menjadi sumber informasi strategis bagi manajemen. Modul ini mendukung visualisasi dan pelaporan data transaksi, kinerja distributor, aktivitas sales, dan data inventori dalam format tabular maupun grafik.

Setelah seluruh kebutuhan dan desain sistem telah dikonfirmasi, proses selanjutnya memasuki tahap pengembangan aplikasi. Setiap modul dikembangkan berdasarkan hasil *fit-gap analysis* yang dihasilkan selama sesi *Conference Room Pilot* (CRP). Proses pengembangan dilakukan secara bertahap dan diikuti oleh pengujian unit dan integrasi, untuk memastikan bahwa sistem telah sesuai dengan rancangan yang telah disepakati sebelum dilanjutkan ke tahap implementasi langsung.

3.2.1.2 Tahap Pengembangan Aplikasi

Pada tahap ini, proses pengembangan aplikasi telah dimulai. Tugas pertama yang diberikan adalah mengembangkan halaman monitoring untuk *Incoterm*. *Incoterm* (International Commercial Terms) merupakan serangkaian istilah perdagangan internasional yang digunakan untuk menjelaskan tanggung jawab

antara penjual dan pembeli dalam proses pengiriman barang, termasuk aspek pengemasan, asuransi, biaya pengangkutan, dan risiko yang ditanggung masing-masing pihak. Halaman monitoring ini dirancang untuk menampilkan data *Incoterm* yang digunakan dalam transaksi distribusi, serta dilengkapi dengan fitur CRUDS (Create, Read, Update, Delete, dan Search) untuk memudahkan pengguna dalam mengelola informasi yang berkaitan dengan term perdagangan tersebut. Pengembangan halaman ini menjadi langkah awal dalam memahami alur kerja sistem distribusi dan standar yang diterapkan dalam pengelolaan data transaksi.



Gambar 3.6 Halaman Monitoring Incoterm

Pada Gambar 3.6, ditampilkan hasil implementasi halaman *Incoterm* yang telah dilengkapi dengan berbagai fitur fungsional. Beberapa fitur utama yang tersedia antara lain pencarian data (search) berdasarkan kolom tertentu, ekspor data ke format Excel atau CSV untuk keperluan pelaporan, serta fitur pengurutan (sorting) pada setiap kolom tabel. Selain itu, pengguna juga dapat melakukan aksi edit, view, dan hapus (delete) terhadap entri yang ada, serta menambahkan

data baru melalui tombol Add New. Seluruh fitur ini dirancang untuk meningkatkan kemudahan dalam pengelolaan data *Incoterm* secara efisien, akurat, dan user-friendly.

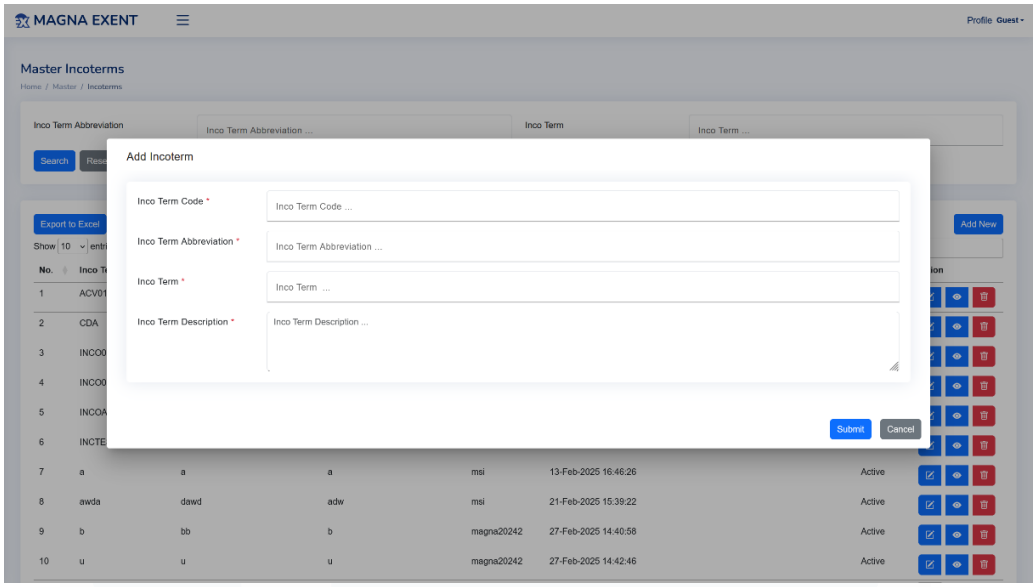
Halaman ini menjadi bagian dari sistem manajemen data master dan dibekali dengan fitur CRUDS (Create, Read, Update, Delete, dan Search) yang memudahkan pengguna dalam melakukan input, pembaruan, penghapusan, serta pencarian data berdasarkan parameter tertentu.

Di bagian atas halaman, terdapat form pencarian dengan dua kolom input: Inco Term Abbreviation dan Inco Term, yang memungkinkan pengguna melakukan penyaringan data secara spesifik. Di sisi kanan, tersedia tombol Add New untuk menambahkan entri *Incoterm* baru ke dalam sistem.

Tabel utama pada halaman ini menampilkan daftar data yang mencakup beberapa atribut penting, yaitu:

a. Inco Term Code (kode term),
b. Inco Term Abbreviation (singkatan term),
c. Inco Term (nama atau deskripsi),
d. Created By, Created Date, Modified By, dan Modified Date,
e. serta kolom Status untuk menunjukkan apakah data aktif atau tidak aktif.

Di kolom paling kanan terdapat tombol aksi (edit dan delete) yang memungkinkan pengguna memperbarui atau menghapus data secara langsung dari tabel. Halaman ini juga dilengkapi dengan tombol Export to Excel yang memungkinkan pengguna mengunduh seluruh data dalam format spreadsheet untuk keperluan pelaporan atau analisis lebih lanjut. Secara keseluruhan, halaman ini dirancang untuk mendukung pengelolaan informasi *Incoterm* secara efisien dan konsisten, yang merupakan bagian penting dalam pengaturan transaksi distribusi lintas wilayah. Dengan tampilan yang intuitif dan fungsionalitas yang lengkap, halaman ini menjadi salah satu komponen awal dalam membangun sistem distribusi yang terdigitalisasi dan terstandarisasi.



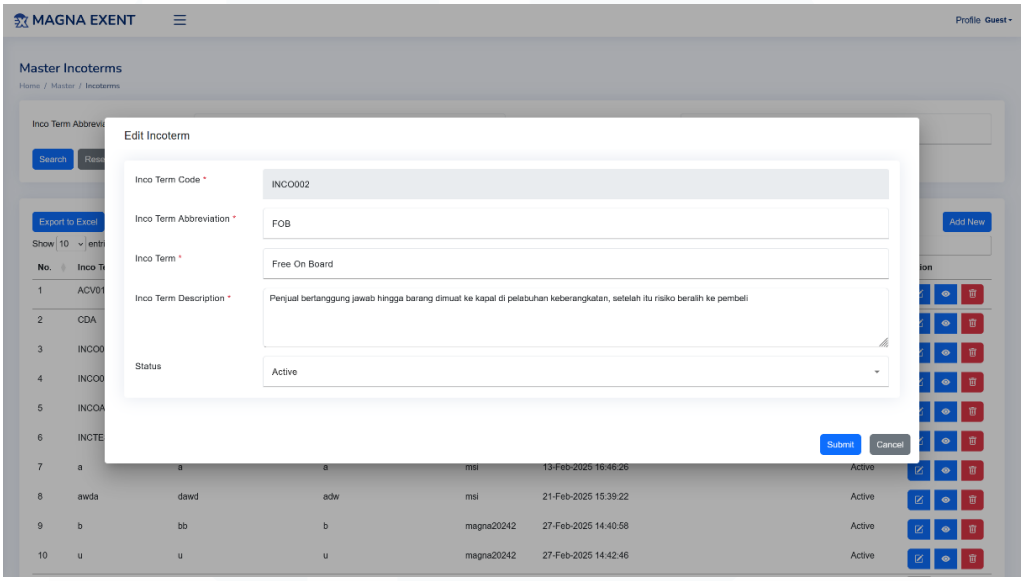
Gambar 3.7 Form Pada Halaman Incoterm

Gambar 3.7 menampilkan tampilan antarmuka dari fitur Add Incoterm yang merupakan bagian dari halaman *Master Incoterms*. Fitur ini digunakan untuk menambahkan entri data *Incoterm* baru ke dalam sistem dan dilengkapi dengan form input yang dirancang secara sederhana namun fungsional. Terdapat empat kolom input yang wajib diisi oleh pengguna, yaitu:

a. Inco Term Code: kode unik yang merepresentasikan setiap term,
b. Inco Term Abbreviation: singkatan dari term perdagangan yang digunakan,
c. Inco Term: nama term secara lengkap, dan
d. Inco Term Description: penjelasan rinci mengenai makna dan penggunaan term tersebut dalam konteks distribusi.

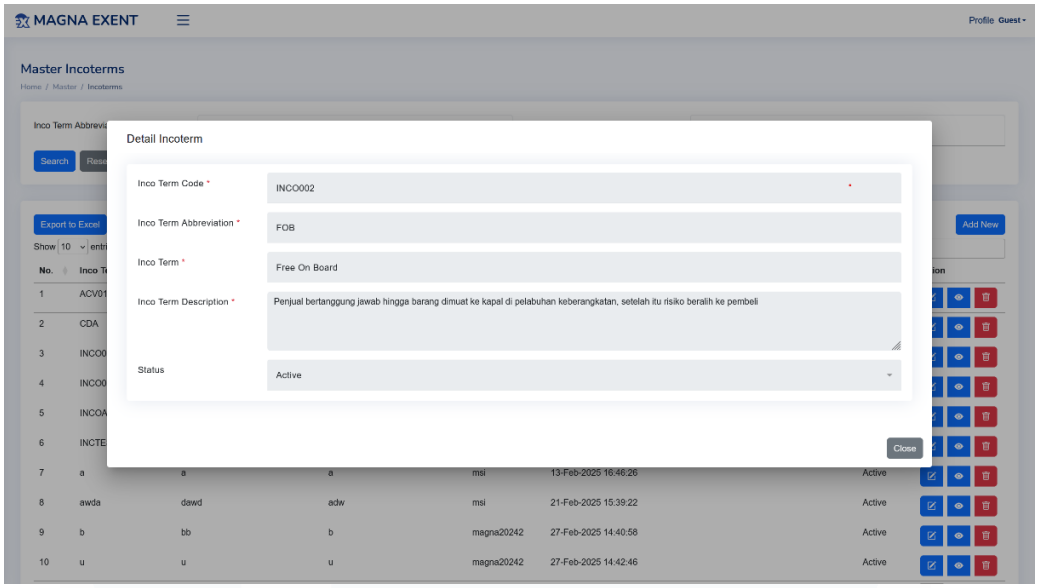
Setiap kolom input pada form Add Incoterm diberi tanda bintang (*) yang menunjukkan bahwa isian tersebut bersifat wajib (*required*) agar data dapat tersimpan ke dalam sistem. Di bagian bawah form, terdapat dua tombol aksi utama, yaitu Submit untuk menyimpan data dan Cancel untuk membatalkan proses input. Fitur ini merupakan bagian penting dari proses Create dalam skema CRUDS yang diterapkan pada halaman tersebut. Dengan adanya form ini, pengguna dapat dengan mudah menambahkan referensi *Incoterm* baru yang

diperlukan untuk mendukung operasional distribusi perusahaan. Desain antarmuka ini dibuat secara intuitif, sehingga dapat digunakan dengan mudah oleh pengguna tanpa memerlukan pelatihan teknis yang mendalam.



Gambar 3.8 Form Edit Halaman Incoterm

Form input yang digunakan pada fitur Add, View, dan Edit memiliki struktur tampilan yang serupa. Perbedaannya terletak pada fitur Edit dan View, di mana terdapat tambahan satu kolom input yaitu Status, yang memungkinkan pengguna untuk mengatur apakah data *Incoterm* tersebut berada dalam kondisi Active atau Inactive. Secara default, ketika data baru ditambahkan melalui fitur Add, status akan otomatis disetel ke Active. Kolom status ini juga berfungsi sebagai bagian dari mekanisme soft delete, di mana fitur Delete tidak benar-benar menghapus data dari sistem, tetapi hanya mengubah status entri dari Active menjadi Inactive. Dengan pendekatan ini, sistem tetap mempertahankan jejak data historis, sekaligus memberikan fleksibilitas bagi pengguna untuk mengelola data tanpa risiko kehilangan permanen. Pendekatan soft delete ini juga mendukung prinsip audit trail dan keamanan data dalam pengelolaan sistem distribusi yang kompleks.



Gambar 3.9 Form View Halaman Incoterm

Halaman monitoring seperti *Master Incoterm* dikembangkan menggunakan pendekatan Reactive Form di sisi frontend, yang merupakan salah satu teknik pengelolaan form berbasis model pada kerangka kerja Angular. Pendekatan ini memungkinkan kontrol penuh terhadap struktur data, validasi input, serta perubahan status form secara reaktif dan dinamis. Secara teoritis, reactive form merupakan implementasi dari *Model-Driven Form* yang memisahkan logika tampilan dengan logika bisnis form, sehingga meningkatkan keterbacaan, modularitas, serta kemudahan pengujian (*testability*) dari kode program [11].

Potongan kode pada Gambar 3.9 merupakan contoh implementasi Reactive Form pada halaman monitoring *Incoterm*, yang digunakan untuk menangani input data dari form *Add/Edit Incoterm* di sisi frontend (Angular). Pendekatan Reactive Form dalam Angular memungkinkan kontrol form yang lebih eksplisit dan terstruktur melalui deklarasi programatik, dibandingkan dengan Template-driven Form yang berbasis pada binding di HTML.

```
formAddEdit: FormGroup = new FormGroup({
    inputId: new FormControl(null),
    inputCode: new FormControl(null, [
      Validators.required, Validators.pattern(/^[\w]+$/),
    ]),
    inputDescription: new FormControl(null, [
      Validators.required, Validators.pattern(/^[\s\S]*$/),
    ]),
    inputAbbreviation: new FormControl(null, [
      Validators.required, Validators.pattern(/^[\s\S]*$/),
    ]),
    inputName: new FormControl(null, [
      Validators.required, Validators.pattern(/^[\s\S]*$/),
    ]),
    inputStatus: new FormControl(null),
  });
```

Gambar 3.10 Contoh Implementasi Reactive Form Pada Monitoring Incoterm

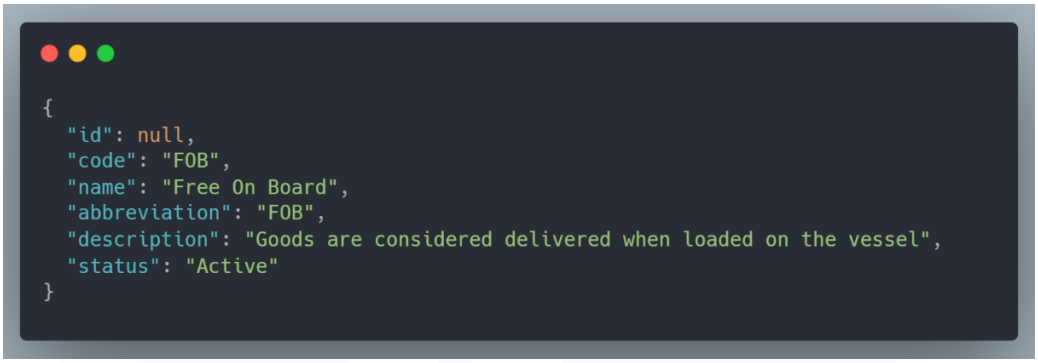
Dalam potongan kode tersebut, dideklarasikan sebuah objek form bernama formAddEdit yang bertipe FormGroup, terdiri dari beberapa FormControl yang merepresentasikan masing-masing input pada form *Incoterm*. Komponen inputId digunakan untuk menyimpan nilai ID, yang umumnya diperlukan saat melakukan proses pembaruan data (*edit*). Field inputCode berfungsi untuk menerima input berupa kode *Incoterm* dan dilengkapi dengan dua validator, yaitu Validators.required yang memastikan bahwa field wajib diisi, serta Validators.pattern(/^[\w]+$/) yang membatasi input agar hanya mengandung karakter alfanumerik tanpa spasi. Sementara itu, field inputDescription, inputAbbreviation, dan inputName masing-masing juga dilengkapi dengan validasi required dan pattern guna menjamin bahwa data yang diinputkan tidak kosong dan sesuai dengan pola karakter yang valid. Terakhir, inputStatus digunakan untuk menangani status data apakah berada dalam kondisi aktif atau tidak aktif, yang juga menjadi bagian penting dalam implementasi fitur *soft delete* pada sistem. Seluruh komponen ini bekerja secara reaktif untuk memvalidasi form sebelum data

dikirimkan ke backend, sesuai dengan prinsip penggunaan *Reactive Form* dalam Angular.

Kode ini merupakan bagian dari inisialisasi form di komponen Angular, di mana setiap kontrol dapat diakses, divalidasi, dan dipantau statusnya secara reaktif. Hal ini memungkinkan tim pengembang untuk memproses form secara dinamis, termasuk menampilkan pesan kesalahan, mengaktifkan tombol submit secara kondisional, dan mengirim data yang telah tervalidasi ke backend melalui HTTP request. Implementasi ini mencerminkan penerapan prinsip *separation of concerns* dan *form validation best practices*, serta mendukung *maintainability* dan *testability* dalam pengembangan halaman-halaman monitoring data master seperti *Incoterm*.

Dalam konteks pengolahan data, input yang dikumpulkan melalui form di sisi klien dikirimkan ke backend dalam bentuk objek JSON melalui protokol HTTP. Data ini kemudian ditangkap oleh backend menggunakan Value Object (VO) sebuah representasi objek sederhana yang hanya memuat data tanpa logika bisnis. VO berperan sebagai media transfer data antara lapisan presentasi dan lapisan layanan (*service layer*), sehingga menjaga isolasi tanggung jawab antar lapisan sesuai prinsip *Separation of Concerns* dalam arsitektur perangkat lunak berlapis (*layered architecture*) [12].

Gambar 3.11 menampilkan contoh struktur data JSON yang dikirimkan dari frontend ke backend saat pengguna mengisi dan mengirim form *Add/Edit Incoterm* melalui halaman monitoring. JSON ini mewakili data yang dihasilkan oleh Reactive Form dan akan ditangkap oleh backend dalam bentuk objek Value Object (VO).



```json
{
  "id": null,
  "code": "FOB",
  "name": "Free On Board",
  "abbreviation": "FOB",
  "description": "Goods are considered delivered when loaded on the vessel",
  "status": "Active"
}
```

Gambar 3.11 Contoh JSON Dalam Halaman Monitoring Incoterm

Format JSON ini merupakan hasil serialisasi data dari form frontend sebelum dikirim ke backend melalui request HTTP. Backend kemudian akan memetakan struktur ini ke dalam kelas VO dan meneruskannya ke lapisan service untuk diproses lebih lanjut. Pendekatan ini mencerminkan praktik umum dalam pengembangan aplikasi berbasis REST API, di mana frontend dan backend berkomunikasi melalui pertukaran data terstruktur dalam format JSON.

```java
@Getter
@Setter
public class voMasterIncoterms {
    private int id;
    private String code;
    private String description;
    private String abbreviation;
    private String name;
    private BigDecimal status;
    private String createdBy;
    private Timestamp createdDate;
    private String modifiedBy;
    private Timestamp modifiedDate;
}
```

Gambar 3.12 Contoh Implementasi VO Pada Halaman Monitoring Incoterm

Gambar 3.12 menunjukkan sebuah kelas Java voMasterIncoterms yang berfungsi sebagai Value Object (VO) untuk menerima dan merepresentasikan data *Incoterm* yang dikirim dari frontend dalam format JSON. Kelas ini digunakan di sisi backend sebagai wadah data tanpa logika bisnis, sesuai dengan prinsip *Sep-*

aration of Concerns. Struktur atribut di dalamnya secara langsung mencerminkan field pada form input, seperti kode, nama, deskripsi, status, serta metadata seperti waktu pembuatan dan modifikasi.

Penggunaan anotasi @Getter dan @Setter dari pustaka Lombok pada kelas voMasterIncoterms merupakan pendekatan yang banyak digunakan dalam pengembangan perangkat lunak berbasis Java untuk meningkatkan efisiensi penulisan kode. Dalam konteks Value Object (VO), getter dan setter berfungsi sebagai metode akses dan mutasi terhadap atribut yang bersifat privat, sehingga tetap menjaga prinsip enkapsulasi dalam paradigma *object-oriented programming (OOP)* [13].

Secara konvensional, setiap atribut pada sebuah kelas harus disertai dengan metode get dan set untuk memungkinkan data dibaca dan dimodifikasi oleh komponen lain tanpa memberikan akses langsung ke field-nya. Namun, pendekatan manual ini dapat menambah beban penulisan kode secara signifikan, terutama pada kelas VO yang umumnya hanya berisi data tanpa logika bisnis. Dengan bantuan anotasi @Getter dan @Setter, Lombok secara otomatis menghasilkan metode tersebut saat proses kompilasi, tanpa perlu menuliskannya secara eksplisit [13].

Hal ini mendukung prinsip *Separation of Concerns* dan *code maintainability*, karena meminimalkan boilerplate code serta menjadikan kelas lebih bersih, terbaca, dan mudah diubah. Dalam praktik pengembangan berbasis arsitektur berlapis, VO yang dilengkapi dengan getter dan setter ini berperan penting dalam proses transfer data dari lapisan presentasi (controller) ke lapisan layanan (service), sekaligus memastikan bahwa alur data tetap terkontrol secara formal dan aman.

Setelah data diterima melalui VO, proses dilanjutkan pada lapisan Service, yang bertugas menangani logika bisnis seperti validasi tambahan, pengecekan integritas data, atau transformasi sebelum disimpan. Selanjutnya, data diteruskan

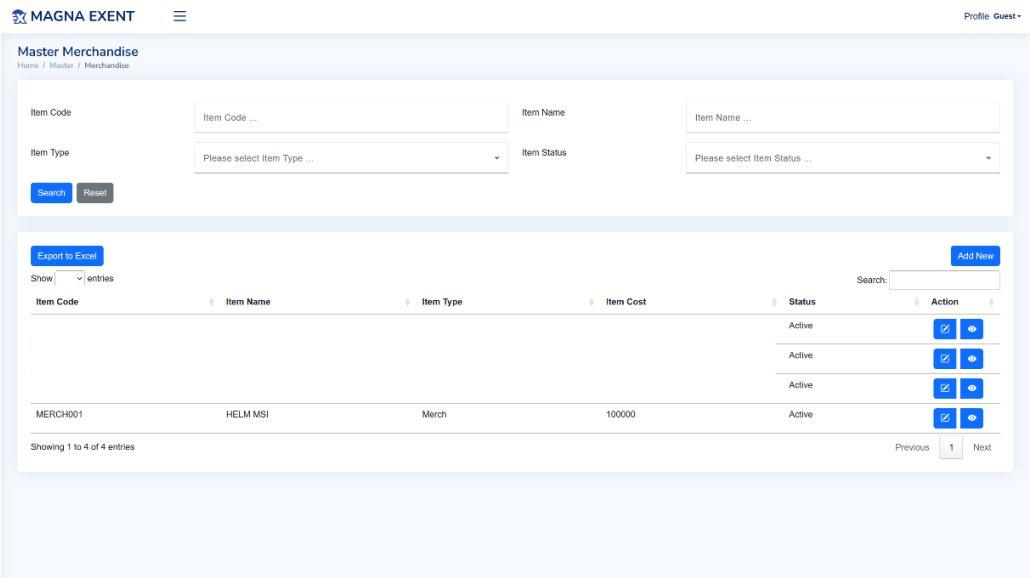
ke lapisan Repository atau Data Access Object (DAO), yang bertanggung jawab dalam interaksi langsung dengan basis data melalui operasi CRUD (Create, Read, Update, Delete). Pola DAO ini secara akademis merupakan implementasi dari prinsip *Data Abstraction*, yang memungkinkan sistem untuk mengakses dan memanipulasi data secara konsisten tanpa harus mengetahui detail teknis penyimpanan data [13].

Walaupun sistem ini masih tergolong sederhana dan berskala terbatas karena belum melibatkan proses yang kompleks seperti sinkronisasi antar modul, validasi lintas entitas, atau interaksi dengan sistem eksternal. Penerapan pola desain seperti reactive form, service-repository pattern, serta penggunaan VO menunjukkan bahwa sistem telah mengikuti praktik pengembangan perangkat lunak modern yang berorientasi pada skalabilitas dan kemudahan pemeliharaan (*maintainability*). Pendekatan ini sejalan dengan prinsip *Clean Architecture* dan *Domain-Driven Design (DDD)* yang banyak digunakan dalam pengembangan sistem enterprise [14].

Pola ini bersifat modular dan diterapkan secara konsisten pada hampir seluruh halaman monitoring data master yang memiliki struktur sederhana tanpa logika bisnis yang kompleks. Salah satu contohnya adalah pada halaman *Master Merchandise*, yang memiliki karakteristik serupa. Perbedaannya hanya terletak pada penyesuaian struktur form, di mana terdapat beberapa field tambahan yang disesuaikan dengan kebutuhan entitas data tersebut. Pendekatan ini memudahkan proses pengembangan karena komponen frontend dan alur backend dapat digunakan kembali dengan sedikit modifikasi, sekaligus menjaga konsistensi antar modul dalam sistem.

Sebagaimana ditampilkan pada Gambar 3.13, halaman *Master Merchandise* memiliki antarmuka yang familiar dan efisien untuk mengelola data barang promosi atau produk merchandise lainnya yang digunakan dalam mendukung kegiatan distribusi. Di bagian atas halaman, terdapat form pencarian yang

memungkinkan pengguna melakukan filter berdasarkan *Item Code*, *Item Name*, *Item Type*, dan *Item Status*. Form ini mendukung proses pencarian data secara cepat dan terarah.



Gambar 3.13 Halaman Monitoring Merchandise

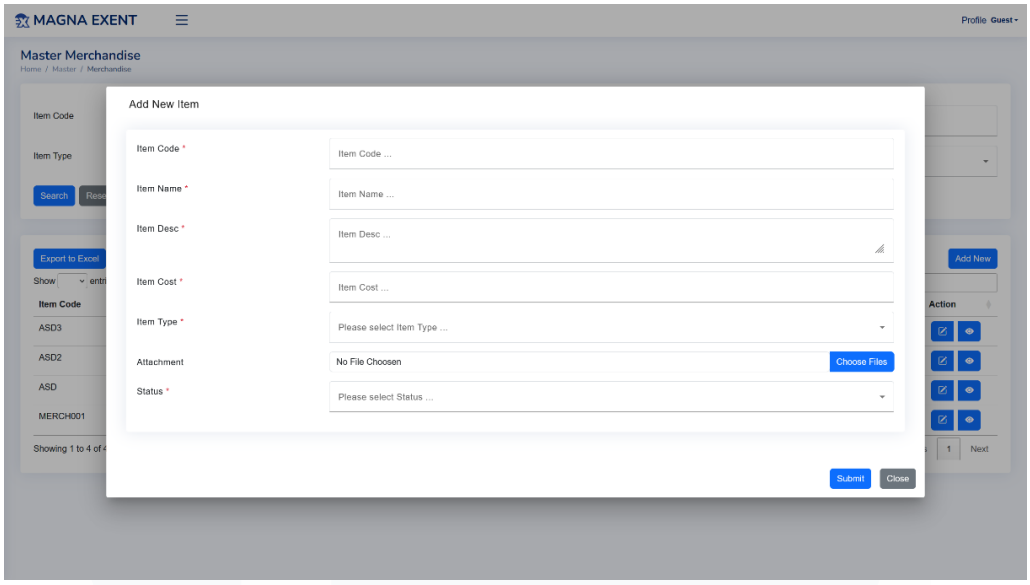
Tabel di bagian bawah menampilkan daftar item yang sudah tercatat, dengan kolom seperti Item Code, Item Name, Item Type, Item Cost, dan Status, serta tombol aksi di sisi kanan yang terdiri dari fitur Edit, View, dan Delete. Fitur ekspor data ke Excel juga tersedia, mendukung kebutuhan dokumentasi dan pelaporan. Struktur dan alur data pada halaman ini juga mengikuti pola yang sama: data input dari frontend akan dikirimkan ke backend menggunakan VO (Value Object), diproses melalui lapisan service, dan disimpan ke dalam database melalui repository (DAO). Dengan pendekatan ini, halaman *Master Merchandise* menjadi bagian dari sistem yang fleksibel, mudah dikembangkan, serta konsisten dengan standar arsitektur yang telah diterapkan sebelumnya.

Halaman Master Merchandise berfungsi sebagai pusat pengelolaan data master untuk item merchandise maupun *rebates* (insentif atau hadiah) yang diberikan dalam rangka mendukung kegiatan distribusi dan promosi perusahaan. Merchandise dalam konteks ini mencakup barang-barang fisik seperti hadiah, bonus

penjualan, atau produk promosi lainnya yang diberikan kepada distributor atau pelanggan dalam skema tertentu. Dengan adanya halaman ini, seluruh daftar item yang termasuk dalam program insentif perusahaan dapat dikelola secara terpusat dan terdokumentasi dengan baik. Fungsi utama dari halaman ini meliputi:

a. Pencatatan dan pengelolaan merchandise/rebates: Pengguna dapat menambahkan entri baru dengan informasi lengkap seperti *Item Code*, *Item Name*, *Item Type*, *Item Cost*, dan *Status*.
b. Pengaturan status aktif/non-aktif: Status digunakan untuk menentukan apakah suatu item masih aktif digunakan dalam skema distribusi atau telah dinonaktifkan (soft delete).
c. Pencarian dan filter data: Fitur filter memudahkan pengguna dalam menelusuri item berdasarkan parameter tertentu, seperti nama item atau tipe barang.
d. Pemutakhiran dan penghapusan data: Fitur edit dan delete (soft delete) mendukung pemeliharaan data secara berkala agar tetap relevan dan akurat.
e. Ekspor data: Tersedia tombol ekspor ke Excel sebagai dukungan untuk kebutuhan pelaporan atau dokumentasi internal.

Halaman ini juga dibangun menggunakan *Reactive Form* yang memungkinkan validasi form dilakukan secara real-time di sisi frontend. Alur pengolahan datanya mengikuti pola arsitektur berlapis, di mana data dikirim dari frontend ke backend melalui objek VO (*Value Object*), kemudian diproses oleh lapisan *Service*, dan disimpan atau dimodifikasi melalui lapisan *Repository (DAO)*. Pendekatan ini memberikan kemudahan dalam pengembangan, pengujian, serta pemeliharaan sistem jangka panjang. Sebagai bagian dari modul data master, halaman ini memegang peranan penting dalam mendukung manajemen program insentif dan pengendalian distribusi merchandise secara lebih efisien dan terdigitalisasi.



Gambar 3.14 Form Add Halaman Merchandise

Sebagai pelengkap dari fungsi halaman *Master Merchandise*, Gambar 3.14 menampilkan tampilan form input untuk fitur Add New Item, yang digunakan untuk menambahkan entri merchandise atau rebate baru ke dalam sistem. Form ini merupakan bagian dari mekanisme *Create* pada skema CRUDS, dan dibangun dengan pendekatan Reactive Form yang memungkinkan validasi input dilakukan secara real-time sebelum data dikirimkan ke backend. Form ini terdiri atas beberapa field yang wajib diisi, ditandai dengan simbol bintang (*), yaitu:

a. Item Code: kode unik untuk mengidentifikasi setiap merchandise,
b. Item Name: nama atau label item,
c. Item Desc: deskripsi detail mengenai item yang dimaksud,
d. Item Cost: nilai atau biaya yang melekat pada item,
e. Item Type: jenis item yang dipilih dari daftar dropdown,
f. Status: status aktif atau tidaknya item dalam sistem,
g. serta Attachment yang bersifat opsional dan digunakan untuk mengunggah dokumen pendukung (misalnya gambar produk atau katalog).

Tersedia dua tombol aksi utama di bagian bawah, yaitu Submit untuk menyimpan data, dan Close untuk membatalkan atau menutup form. Setelah data dikirim, informasi akan diterima di backend melalui objek Value Object (VO), kemudian diproses oleh lapisan Service, dan disimpan oleh Repository (DAO) ke dalam basis data. Struktur ini mencerminkan pola pengembangan berlapis yang mendukung skalabilitas, validasi berjenjang, serta kemudahan pemeliharaan sistem. Dengan menyediakan fitur tambah data yang intuitif dan lengkap seperti ini, sistem memungkinkan pengelolaan program merchandise dan rebate dilakukan secara efisien, terdokumentasi, serta sesuai dengan standar operasional yang telah ditetapkan perusahaan.

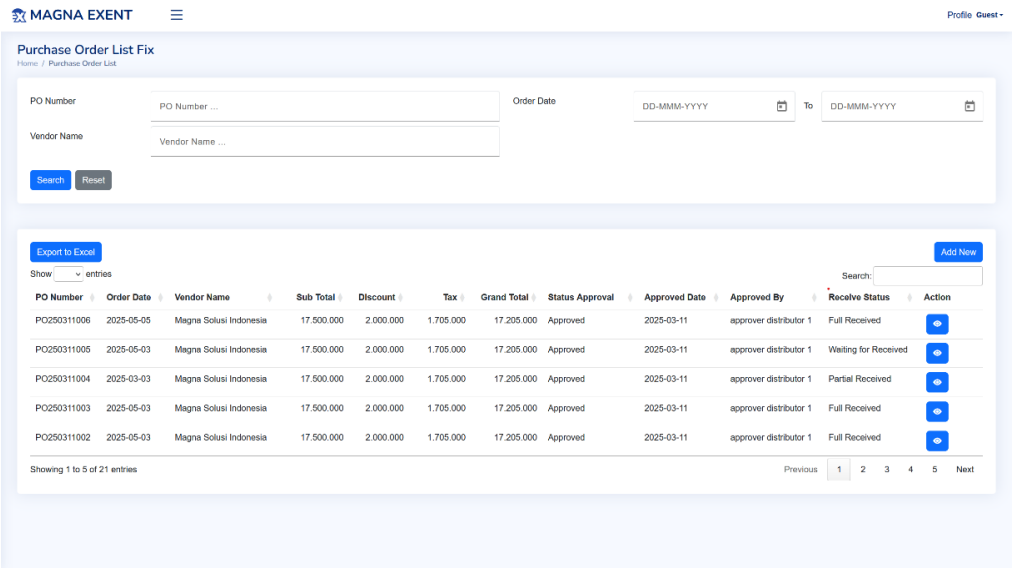
Selain halaman-halaman monitoring data master yang bersifat sederhana, terdapat juga sejumlah halaman dalam sistem DMS yang memiliki proses bisnis cukup kompleks, seperti *Purchase Order (PO)*, *Sales Order (SO)*, dan *Delivery Order (DO)*. Ketiga halaman ini tidak hanya menangani transaksi internal distribusi, tetapi juga dirancang untuk terintegrasi dengan sistem eksternal seperti SAP. Integrasi ini memungkinkan sinkronisasi data secara otomatis dan real-time, sehingga seluruh proses mulai dari pemesanan hingga pengiriman barang dapat dikelola secara efisien dan akurat. Kompleksitas juga muncul karena halaman-halaman ini memiliki ketergantungan langsung terhadap data dari modul lain, seperti data pelanggan, harga, diskon, dan stok barang, yang mengharuskan adanya query dan logika khusus pada sisi backend untuk menjembatani antar modul maupun antar entitas dalam sistem.

Tidak hanya terbatas pada PO, SO, dan DO, sistem DMS ini juga mencakup halaman-halaman lain yang memiliki beban proses yang tinggi dan ketergantungan lintas modul, seperti *Invoice*, *Good Receipt*, *Pick Order*, *Sales Return*, *Payment*, dan *Proof of Delivery*. Halaman *Invoice* digunakan untuk mengelola penagihan yang berkaitan dengan transaksi SO, dengan validasi terhadap status pengiriman dan pembayaran. *Good Receipt* mencatat penerimaan barang dari pemasok, yang berpengaruh langsung terhadap status stok di gudang. *Pick Order*

mendukung proses pengambilan barang dari gudang untuk pengiriman, dan biasanya terkait erat dengan DO. *Sales Return* mengelola proses pengembalian barang dari pelanggan, yang membutuhkan validasi data asal pengiriman dan koreksi stok. *Payment* mencatat dan memverifikasi transaksi pembayaran dari pelanggan, termasuk perhitungan sisa tagihan dan pengelolaan aging piutang. Sementara *Proof of Delivery* berfungsi untuk memastikan bahwa barang telah diterima oleh pihak tujuan dengan validasi tanda tangan elektronik atau dokumen pendukung lainnya.

Seluruh modul tersebut merupakan bagian dari proses bisnis aplikasi yang secara umum dikenal sebagai *Order-to-Cash (O2C)*. O2C adalah alur proses bisnis menyeluruh yang dimulai dari penerimaan pesanan oleh pelanggan hingga transaksi pembayaran selesai dilakukan [15]. Proses ini terdiri atas berbagai tahapan seperti pencatatan pesanan (SO), pengadaan barang (PO), pengiriman (DO), pembuatan invoice, pencatatan pembayaran, hingga proses retur dan pembuktian pengiriman. Setiap tahap memiliki dependensi terhadap data yang akurat dan validasi logis antar sistem, sehingga pengelolaannya memerlukan sistem yang mendukung keterkaitan antar modul dengan integritas data yang tinggi.

Lebih jauh, implementasi proses bisnis dalam aplikasi seperti DMS didasarkan pada prinsip *Business Process Management (BPM)*, yaitu pendekatan sistematis untuk mendesain, menjalankan, memantau, dan mengoptimalkan proses-proses bisnis yang berbasis sistem informasi [16]. BPM menekankan pentingnya keterpaduan antar unit kerja dan fleksibilitas sistem dalam menghadapi dinamika kebutuhan operasional. Oleh karena itu, dalam pengembangan sistem DMS, pendekatan modular berbasis *microservices* menjadi sangat relevan. Arsitektur ini memungkinkan setiap proses bisnis dikembangkan secara terpisah berdasarkan domain, sehingga lebih mudah untuk dikelola, diuji, dan diintegrasikan kembali. Selain itu, microservices juga memudahkan pengembangan berkelanjutan dan mendukung skalabilitas sistem ketika beban transaksi meningkat.



Gambar 3.15 Halaman Monitoring Purchase Order

Gambar 3.15 merupakan halaman monitoring *purchase order* yang menjadi bagian penting dalam alur proses bisnis pengadaan barang di dalam sistem DMS. Secara fungsional, halaman ini dirancang untuk menampilkan daftar pesanan pembelian (*Purchase Order* atau PO) yang telah *dibuat* oleh pengguna, dilengkapi dengan rincian informasi seperti nomor PO, nama vendor, tanggal pesanan, total nilai transaksi, status persetujuan, pihak yang menyetujui, serta status penerimaan barang (*Receive Status*). Seluruh data yang ditampilkan dapat disaring berdasarkan nomor PO, nama vendor, dan rentang tanggal pesanan, serta disediakan fitur pencarian dan ekspor data dalam bentuk Excel untuk mendukung kebutuhan pelaporan atau audit.

Halaman ini menjadi krusial dalam rantai distribusi karena menghubungkan aktivitas pemesanan barang oleh distributor dengan proses pengiriman oleh vendor. Di sisi backend, halaman ini tidak hanya menampilkan data PO secara langsung, tetapi juga harus melakukan agregasi dan join data dari beberapa tabel seperti tabel master vendor, data approval, dan status penerimaan barang yang mungkin berasal dari modul *Good Receipt*. Hal ini membuat query yang

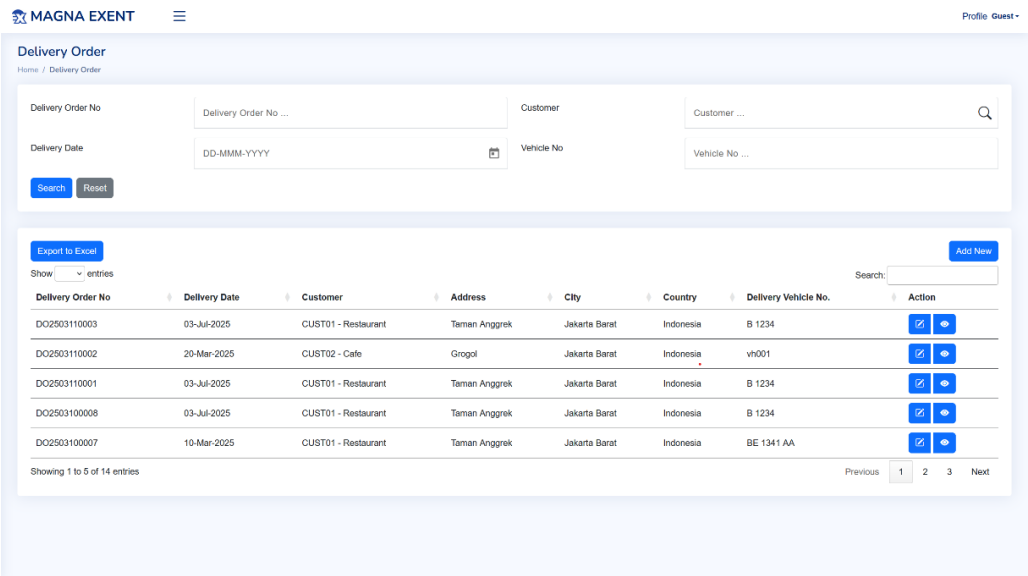
digunakan pada halaman ini cenderung kompleks dan perlu dioptimalkan agar tetap efisien meskipun menangani data dalam jumlah besar.

Dari sisi logika bisnis, keberadaan kolom seperti *Status Approval* dan *Receive Status* mencerminkan penerapan kontrol dua lapis yang umum ditemukan dalam sistem manajemen pengadaan: pertama, verifikasi administratif oleh pihak internal; kedua, konfirmasi penerimaan fisik barang yang dilakukan oleh pihak gudang. Status seperti "Waiting for Received", "Partial Received", dan "Full Received" merupakan bagian dari logika pengendalian stok dan validasi kewajaran invoice, yang akan terintegrasi dengan proses lanjut seperti *Payment* dan *Invoice Matching*. Oleh karena itu, halaman ini tidak berdiri sendiri, tetapi menjadi penghubung langsung ke proses berikutnya dalam siklus *procure-to-pay*, menjadikannya bagian integral dari sistem proses bisnis aplikasi yang terstruktur dan terstandar.

Tidak hanya halaman monitoring *Purchase Order (PO)* yang memiliki tampilan dan struktur seperti ini, tetapi halaman *Sales Order (SO)* dan *Delivery Order (DO)* dalam sistem DMS juga menggunakan pola antarmuka yang serupa. Ketiganya menampilkan daftar transaksi dengan fitur pencarian dinamis, filter berdasarkan parameter tertentu seperti tanggal atau nama pelanggan/vendor, serta kemampuan ekspor data. Konsistensi desain ini bertujuan untuk memudahkan pengguna dalam memahami alur kerja, mempercepat proses operasional, serta menjaga standarisasi antar modul transaksi yang memiliki keterkaitan erat dalam alur distribusi barang.

Halaman *Delivery Order (DO)* seperti yang ditampilkan pada Gambar 3.16 memiliki fungsi utama sebagai pusat kontrol dan pemantauan aktivitas pengiriman barang dari distributor kepada pelanggan. Tujuan dari halaman ini adalah untuk menyediakan informasi terkini (*real-time*) terkait proses distribusi fisik, mulai dari nomor DO, tanggal pengiriman, identitas pelanggan, alamat tujuan, hingga detail kendaraan pengantar barang. Fitur pencarian berdasarkan nomor

DO, tanggal, nama pelanggan, dan nomor kendaraan juga memudahkan tim logistik dan operasional dalam melakukan pelacakan dan verifikasi secara cepat dan akurat.

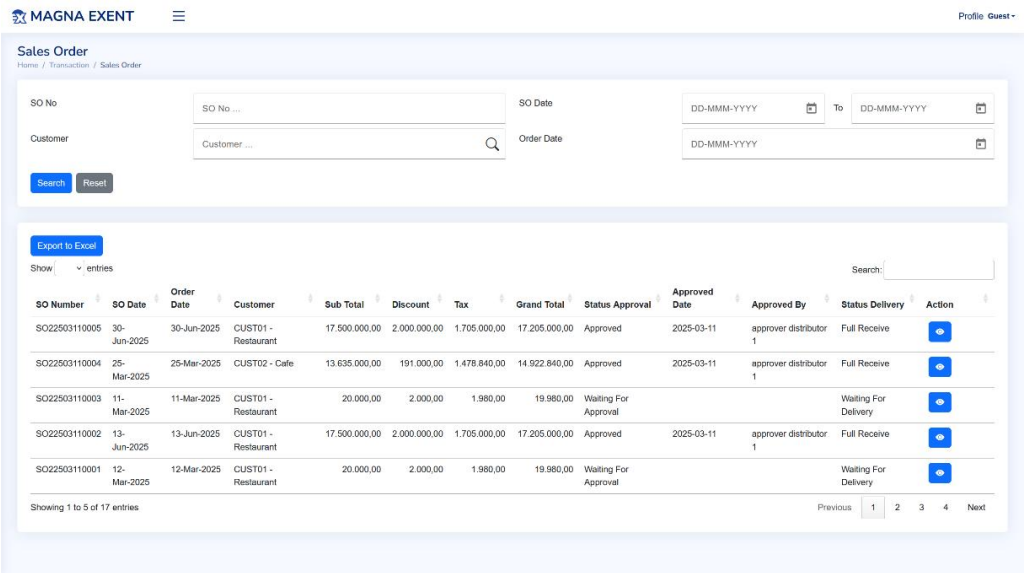


Gambar 3.16 Halaman Monitoring Delivery Order

Secara teknis, halaman ini berintegrasi langsung dengan modul *Sales Order (SO)* sebagai sumber data pesanan, serta modul transportasi yang mengelola penjadwalan armada. Informasi seperti *Delivery Vehicle Number* menjadi elemen penting dalam pelacakan dan validasi pengiriman. Selain itu, data dari halaman ini juga akan dikaitkan dengan modul *Proof of Delivery (POD)* guna memastikan bahwa barang benar-benar telah diterima oleh pelanggan sesuai jadwal dan alamat tujuan.

Dari sisi proses bisnis, halaman ini berperan sebagai penghubung antara tahap administratif (*order entry*) dan eksekusi logistik (*order fulfillment*), sehingga menjadi elemen penting dalam menjamin ketepatan waktu, efisiensi distribusi, dan kepuasan pelanggan. Ketika terjadi keterlambatan, kehilangan, atau ketidaksesuaian dalam pengiriman, halaman DO memungkinkan tim untuk menelusuri histori transaksi dan segera melakukan evaluasi serta perbaikan yang dibu-

tuhkan. Dengan demikian, halaman ini tidak hanya menampilkan data pengiriman, tetapi juga menjalankan fungsi strategis dalam mendukung pengendalian operasional dan pengambilan keputusan berbasis data.



Gambar 3.17 Halaman Monitoring Sales Order

Halaman *Sales Order (SO)* berfungsi sebagai titik awal utama dalam proses bisnis distribusi pada sistem DMS. Kegunaan utamanya adalah untuk mencatat, memantau, dan mengelola seluruh transaksi pemesanan produk dari pelanggan sebelum dilakukan pengiriman maupun penagihan. Dalam alur *Order-to-Cash (O2C)*, halaman ini merepresentasikan fase order entry yang krusial, karena seluruh proses logistik dan finansial selanjutnya, seperti pembuatan Delivery Order (DO), invoice, dan pembayaran akan bergantung pada informasi yang terekam dalam modul SO ini.

Melalui halaman ini, tim operasional dan penjualan dapat melakukan verifikasi terhadap data pesanan seperti jumlah item, harga, diskon, dan total tagihan, serta melakukan validasi terhadap status persetujuan dan status pengiriman. Hal ini memungkinkan perusahaan untuk mengatur jadwal pengiriman secara efisien, memastikan ketersediaan barang, dan menghindari kesalahan pengiriman. Selain itu, keberadaan kolom *Status Approval* dan *Status Delivery* juga

memudahkan pengawasan terhadap progres setiap pesanan, apakah sudah siap untuk diproses lebih lanjut atau masih dalam antrian persetujuan. Dengan demikian, halaman *Sales Order* tidak hanya berperan sebagai alat pencatatan transaksi, tetapi juga sebagai pengendali utama dalam pengambilan keputusan distribusi dan logistik. Keakuratan dan integritas data dalam halaman ini sangat memengaruhi efektivitas operasional serta kepuasan pelanggan secara keseluruhan.

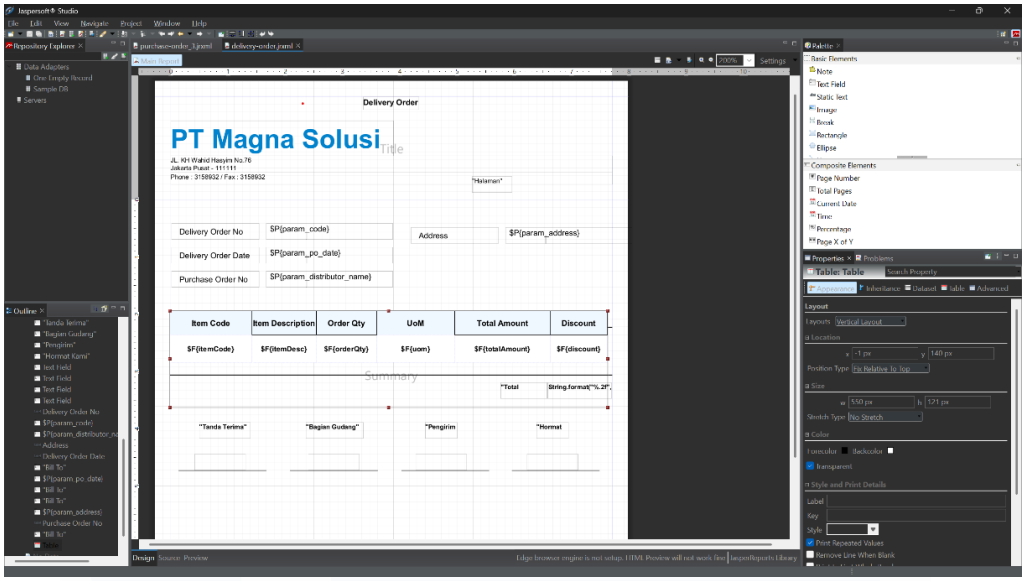
Dalam sistem Distributor Management System (DMS), sejumlah halaman utama seperti *Sales Order (SO)*, *Delivery Order (DO)*, dan *Purchase Order (PO)*, serta modul lain seperti *Invoice*, *Payment*, dan *Proof of Delivery* dilengkapi dengan fitur untuk menghasilkan laporan (*report*) dalam format PDF. Fitur ini tidak hanya berfungsi sebagai alat bantu visual, tetapi juga berperan penting dalam mendokumentasikan dan mengarsipkan aktivitas bisnis secara formal. Laporan-laporan ini menyajikan data transaksi yang terstruktur, mencakup informasi pihak terkait (pelanggan/vendor), daftar barang, nilai transaksi, status persetujuan, dan status pengiriman. Dokumen ini kemudian dapat digunakan sebagai bukti sah transaksi bisnis, sebagai bahan evaluasi, maupun untuk kepentingan audit dan pelaporan manajerial.

Dari sudut pandang bisnis dan operasional, laporan dalam bentuk PDF memainkan peran penting dalam proses tata kelola dan pengendalian internal. Laporan *Sales Order*, misalnya, dapat dijadikan dokumen pendukung proses penagihan dan perencanaan distribusi, sedangkan laporan *Delivery Order* berfungsi sebagai dokumen pendamping pengiriman barang yang diserahkan kepada sopir dan pelanggan. Selain itu, laporan *Purchase Order* dapat digunakan untuk validasi pembelian, rekonsiliasi keuangan, serta penyusunan laporan pengadaan. Seluruh laporan ini menjadi bagian dari *Business Documentation System*, yaitu sistem dokumentasi yang mendukung keterlacakan (*traceability*), transparansi, dan akuntabilitas dalam sistem informasi manajemen perusahaan.

Secara teknis, sistem ini menggunakan JasperReports, salah satu pustaka open-source yang umum digunakan dalam pengembangan perangkat lunak berbasis Java untuk pembuatan laporan yang fleksibel dan terformat dengan baik. JasperReports mendukung pengolahan data secara dinamis berdasarkan parameter, serta memungkinkan desain laporan dengan berbagai komponen visual seperti tabel, teks, gambar, dan grafik. Di sisi backend, laporan dihasilkan melalui integrasi antara *template* .jrxml dan data transaksi yang diproses melalui *service layer*. Data dari database akan dipetakan ke template tersebut, lalu dikompilasi dan dikonversi menjadi file PDF yang dapat diunduh pengguna atau diarsipkan dalam sistem [17].

Keberadaan fitur ini merepresentasikan penerapan nyata dari prinsip *Information Systems Documentation*, yaitu proses penyusunan dokumen terstruktur yang mendukung pengambilan keputusan, pengendalian proses bisnis, serta pemenuhan kebutuhan regulasi dan audit [18]. Selain itu, fitur ini juga mencerminkan implementasi *Separation of Concerns (SoC)* dalam arsitektur perangkat lunak, di mana pengolahan data bisnis dipisahkan dengan tampilan dan format pelaporan, memungkinkan sistem lebih modular dan mudah dikembangkan. Laporan-laporan ini bukan sekadar salinan data, melainkan bentuk artefak digital dari aktivitas operasional yang mendukung *evidence-based decision making* di lingkungan organisasi yang kompleks dan data-driven [19].

Sebagai kelanjutan dari proses implementasi fitur cetak laporan menggunakan JasperReports, Gambar 3.18 menampilkan salah satu contoh konkret desain template laporan *Delivery Order (DO)* yang dibuat melalui Jaspersoft Studio. Template ini berfungsi sebagai cetakan digital yang akan digunakan untuk menghasilkan laporan pengiriman dalam bentuk file PDF, sesuai dengan data transaksi yang terekam di sistem. Desain laporan ini dibangun dengan struktur yang terorganisir dan profesional, mencakup elemen-elemen informasi utama seperti identitas penerima, tanggal pengiriman, nomor purchase order terkait, serta tabel berisi rincian barang yang dikirim.



Gambar 3.18 Implementasi Dokumen Report Untuk Sales Order

Secara teknis, elemen-elemen dalam template ini dibagi menjadi dua jenis utama, yaitu *parameter* dan *field*. Parameter seperti ${param_code}, ${param_po_date}, dan ${param_distributor_name} merupakan nilai dinamis yang diisi pada saat runtime oleh sistem backend melalui pemanggilan service, yang memungkinkan fleksibilitas dalam mencetak laporan berdasarkan input pengguna atau data transaksi tertentu. Sementara itu, bagian tabel menggunakan field seperti $F{itemCode}, $F{itemDesc}, $F{orderQty}, $F{uom}, $F{totalAmount}, dan $F{discount} yang secara otomatis akan diisi berdasarkan hasil query data transaksi yang dikirim dari backend. Mekanisme ini memungkinkan laporan mencetak daftar barang yang dikirim dengan format tabular yang rapi dan sistematis.

Pada bagian bawah laporan juga disediakan ruang untuk tanda tangan berbagai pihak yang terlibat, seperti bagian gudang, pengirim, dan penerima. Hal ini mendukung kebutuhan legalitas dan pelacakan fisik atas pengiriman barang, serta memperkuat aspek akuntabilitas dalam proses logistik. Dengan adanya fitur ini, sistem tidak hanya mampu menyimpan data digital, tetapi juga menyediakan representasi dokumen resmi yang siap dicetak, disimpan, atau dibagikan ke

pihak eksternal. Penerapan JasperReports dalam konteks ini menunjukkan peran penting laporan digital dalam mengintegrasikan dokumentasi bisnis dengan sistem informasi yang terotomatisasi.

Kerumitan sistem ini tidak hanya terletak pada hubungan antar modul, tetapi juga pada penerapan alur bisnis yang sesuai dengan standar operasional perusahaan serta kebijakan distribusi yang bervariasi di tiap wilayah. Oleh karena itu, pengembangan halaman-halaman dalam sistem DMS memerlukan desain query yang efisien, struktur data yang saling terhubung, serta validasi berlapis agar seluruh proses bisnis dapat berjalan secara konsisten dan terintegrasi dengan baik.

Salah satu langkah penting dalam mendukung integrasi sistem yang kompleks ini adalah pelaksanaan migrasi database. Migrasi bertujuan untuk memastikan bahwa seluruh struktur, skema, dan relasi data dapat menyesuaikan dengan kebutuhan sistem berbasis arsitektur microservices, serta mendukung berbagai fitur tingkat lanjut seperti integrasi antarmodul dan penyusunan laporan yang dinamis. Proses ini tidak terbatas pada pemindahan data antar lingkungan, tetapi juga mencakup normalisasi entitas, penyesuaian relasi antartabel, serta reformulasi format penyimpanan agar sesuai dengan praktik pengembangan sistem modern.

Langkah normalisasi dilakukan untuk menghindari redundansi data dan memastikan efisiensi dalam eksekusi query, terutama pada modul-modul utama seperti *User Management*, *Master Data*, *Purchase Order*, dan *Delivery Order*. Dengan struktur basis data yang lebih optimal dan modular, sistem menjadi lebih stabil, scalable, serta siap diintegrasikan dengan sistem eksternal seperti ERP atau SAP. Proses ini juga menjadi fondasi penting dalam menjamin keberhasilan implementasi sistem DMS secara menyeluruh dan berkelanjutan.

Sebagai bagian dari tahapan akhir dalam pengembangan, dilakukan proses migrasi dari lingkungan *development* ke lingkungan *production*. Tahap ini berperan penting dalam mentransfer skema dan data yang sebelumnya diuji secara

lokal ke server operasional yang digunakan langsung oleh pengguna akhir. Lingkungan *development* berfungsi sebagai ruang eksperimental untuk pengujian logika bisnis dan simulasi alur data, sedangkan *production* memerlukan konfigurasi yang lebih matang dari sisi keamanan, performa, dan ketahanan terhadap beban kerja tinggi.

Proses migrasi ini mencerminkan prinsip dalam Software Deployment Lifecycle, khususnya pada fase transisi sistem dari pengembangan ke implementasi penuh (*development to production handover*) [20]. Tahapan ini mencakup validasi akhir struktur data, penghapusan data dummy, penyesuaian indeks dan relasi, serta pengujian performa dan integritas data. Selain itu, konfigurasi environment seperti koneksi database (JDBC), pembagian hak akses pengguna, serta aktivasi fitur pendukung seperti logging dan backup otomatis juga dilakukan pada tahap ini.

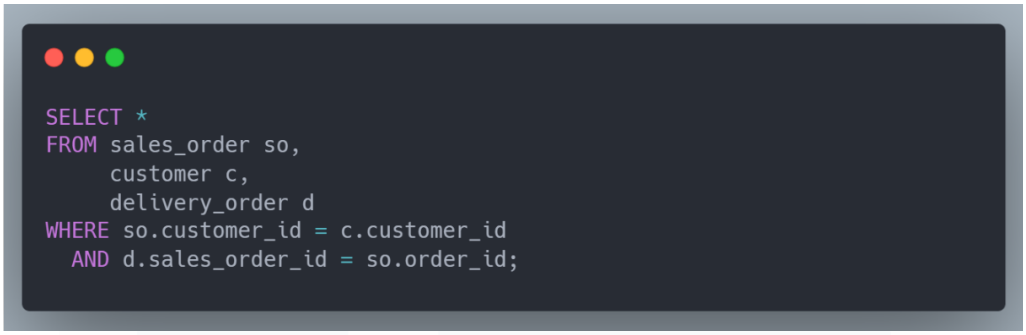
Untuk memastikan performa sistem dalam jangka panjang, dilakukan pula proses optimasi query, yaitu dengan mengidentifikasi dan memperbaiki kueri-kueri yang tidak optimal agar menjadi lebih efisien dan sesuai dengan standar performa sistem produksi. Optimasi ini mencakup perbaikan struktur *JOIN*, penghindaran penggunaan SELECT *, penerapan filter yang spesifik melalui WHERE clause, serta pemanfaatan indeks dan relasi antar tabel untuk mempercepat proses eksekusi data. Selain itu, beberapa query yang bersifat kompleks dan tidak efisien dipecah menjadi bagian-bagian yang lebih modular, atau dikonversi menjadi *stored procedures* dan *materialized views* untuk mengurangi beban komputasi berulang. Pendekatan ini tidak hanya meningkatkan kecepatan pemrosesan, tetapi juga mengurangi penggunaan sumber daya sistem secara berlebihan seperti memori dan CPU.

Dari perspektif teoritis, praktik ini sejalan dengan prinsip dalam Database Performance Tuning, yang menyatakan bahwa desain query yang buruk merupakan salah satu penyebab utama penurunan performa sistem berbasis data [21].

[21] menekankan pentingnya optimasi kueri sebagai bagian dari *logical database design*, karena eksekusi query yang tidak efisien dapat menurunkan throughput, memperlambat respons sistem, serta meningkatkan risiko bottleneck pada saat sistem berada dalam beban kerja tinggi [21]. Oleh karena itu, dalam konteks migrasi dari lingkungan pengembangan ke produksi, evaluasi dan optimasi query menjadi langkah strategis untuk menjamin bahwa sistem dapat beroperasi secara cepat, efisien, dan dapat diskalakan seiring dengan pertumbuhan data dan pengguna.

Dengan dilakukannya optimasi ini, sistem tidak hanya sekadar berfungsi secara fungsional, tetapi juga siap menangani skenario penggunaan nyata yang membutuhkan efisiensi akses data, konsistensi performa, serta stabilitas pada tingkat sistem produksi. Maka dari itu, optimasi query tidak dapat dipisahkan dari tahapan migrasi database dan deployment, karena keduanya saling terkait dalam mendukung keberhasilan implementasi sistem informasi yang responsif dan andal.

Salah satu bentuk penerapan langsung dari upaya optimasi tersebut terdapat pada modul monitoring *Sales Order (SO)*, di mana data pesanan pelanggan harus ditampilkan secara real-time untuk mendukung kebutuhan operasional distribusi. Query yang digunakan dalam modul ini bertugas menampilkan informasi penting seperti nomor pesanan, tanggal order, nama pelanggan, dan status pengiriman dari *Delivery Order (DO)* yang terkait. Informasi ini digunakan oleh tim operasional dan logistik untuk melacak pesanan yang telah masuk, memverifikasi pelanggan, dan menentukan apakah pesanan sudah siap dikirim atau masih dalam tahap persiapan. Karena data ini bersifat dinamis dan akan terus bertambah setiap hari, maka performa query sangat berpengaruh terhadap respons sistem secara keseluruhan, terutama saat digunakan oleh banyak pengguna secara bersamaan.

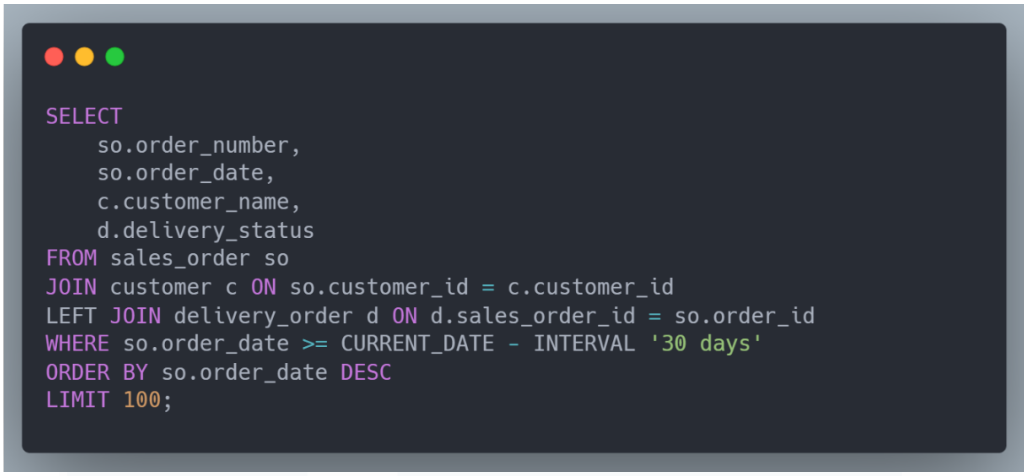


```
SELECT *
FROM sales_order so,
     customer c,
     delivery_order d
WHERE so.customer_id = c.customer_id
  AND d.sales_order_id = so.order_id;
```

Gambar 3.19 Contoh Kueri Yang Belum Dioptimasi

Sebagai contoh, pada Gambar 3.19 ditampilkan sebuah potongan query yang belum dioptimasi, yang digunakan untuk mengambil data dari tiga tabel utama, yaitu sales_order, customer, dan delivery_order. Query ini ditulis menggunakan sintaks *comma-separated join*, di mana relasi antar tabel hanya ditentukan dalam klausa WHERE. Selain itu, seluruh kolom diambil menggunakan SELECT *, tanpa memperhatikan relevansi atau efisiensi data yang dimuat. Pendekatan seperti ini memiliki sejumlah kelemahan, di antaranya adalah beban pemrosesan yang tinggi karena mengambil semua atribut dalam tabel, potensi konflik kolom dengan nama serupa, serta rendahnya keterbacaan dan skalabilitas query saat diterapkan dalam sistem produksi yang memiliki skala data besar.

Dalam pengembangan sistem informasi berbasis database, query semacam ini sangat tidak disarankan karena dapat menurunkan performa sistem secara signifikan, terutama jika dijalankan tanpa filter pembatas waktu atau jumlah data. Oleh karena itu, perbaikan terhadap struktur query menjadi bagian penting dalam proses optimasi untuk mendukung kebutuhan performa dan keterandalan sistem yang akan digunakan dalam lingkungan operasional yang sebenarnya.



```
SELECT
    so.order_number,
    so.order_date,
    c.customer_name,
    d.delivery_status
FROM sales_order so
JOIN customer c ON so.customer_id = c.customer_id
LEFT JOIN delivery_order d ON d.sales_order_id = so.order_id
WHERE so.order_date >= CURRENT_DATE - INTERVAL '30 days'
ORDER BY so.order_date DESC
LIMIT 100;
```

Gambar 3.20 Contoh Kueri Yang Sudah Dioptimasi

Sebagai tindak lanjut dari evaluasi terhadap query yang belum optimal, kemudian disarankan untuk menggunakan struktur query seperti yang ditunjukkan pada Gambar 3.20. Query tersebut dirancang dengan pendekatan yang lebih efisien dan terstruktur, yaitu hanya memilih kolom yang relevan seperti order_number, order_date, customer_name, dan delivery_status, alih-alih mengambil seluruh kolom menggunakan SELECT *. Pemilihan kolom secara spesifik ini membantu mengurangi beban transfer data antara basis data dan aplikasi, sehingga mempercepat waktu respon sistem.

Selain itu, penggunaan JOIN eksplisit antara tabel sales_order, customer, dan delivery_order memperjelas hubungan antar entitas, serta mendukung keterbacaan kode bagi pengembang lain di kemudian hari. Penerapan LEFT JOIN memastikan bahwa data Sales Order tetap dapat ditampilkan meskipun belum memiliki data Delivery Order terkait, suatu kebutuhan penting dalam proses pemantauan distribusi.

Filter waktu yang diterapkan menggunakan WHERE so.order_date >= CURRENT_DATE - INTERVAL '30 days' membatasi hasil query hanya pada transaksi 30 hari terakhir, sehingga menghindari pemrosesan data historis yang tidak relevan. Tambahan ORDER BY so.order_date DESC memastikan data

terbaru muncul terlebih dahulu, sedangkan LIMIT 100 menjaga agar hasil yang ditampilkan tetap manageable, terutama dalam konteks pagination.

Dengan struktur seperti ini, query tidak hanya memenuhi kebutuhan fungsional pengguna, tetapi juga mencerminkan penerapan prinsip-prinsip dasar dalam query optimization, seperti selektivitas data, efisiensi pemanfaatan sumber daya, serta keterbacaan dan kejelasan logika. Desain query yang efisien menjadi komponen penting dalam tahapan implementasi sistem berbasis data, khususnya pada sistem berskala menengah hingga besar seperti DMS yang menangani data transaksi dalam jumlah besar dan berelasi antar modul. Namun, agar efisiensi query dapat tercapai secara konsisten, fondasi struktur basis data juga harus didukung oleh rancangan skema yang baik. Salah satu pendekatan yang digunakan untuk mendukung hal tersebut adalah proses normalisasi tabel, yang bertujuan untuk merancang struktur data yang bebas dari duplikasi, menjaga integritas referensial, dan memfasilitasi ekspansi sistem di masa mendatang. Pembahasan berikutnya akan menjelaskan bagaimana normalisasi diterapkan pada proyek ini sebagai bagian dari penyempurnaan desain basis data yang terintegrasi dan efisien.

Dalam implementasinya, proses normalisasi tabel dilakukan dengan membagi data ke dalam entitas-entitas terpisah yang memiliki satu fokus spesifik, kemudian menghubungkannya melalui relasi yang jelas menggunakan kunci primer dan kunci asing (*primary key* dan *foreign key*) [22]. Pada proyek DMS ini, normalisasi diterapkan pada sejumlah modul utama seperti *User Management*, *Master Customer*, *Sales Order*, hingga *Delivery Order*. Sebagai contoh, data pelanggan yang sebelumnya tersebar pada beberapa tabel transaksi dipindahkan ke tabel customer, sehingga setiap entri transaksi cukup mereferensikan customer_id. Hal ini tidak hanya mengurangi duplikasi data, tetapi juga memudahkan proses pembaruan dan menjaga konsistensi informasi di seluruh modul sistem.

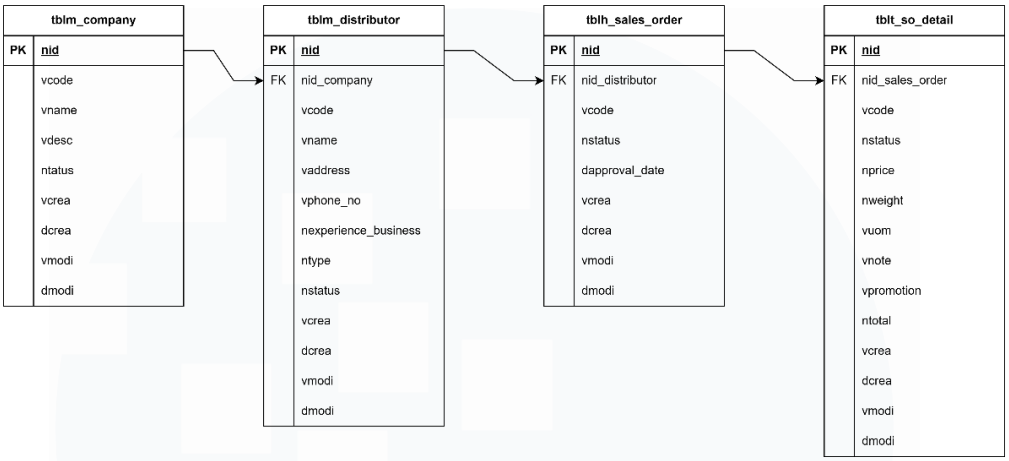
Pendekatan ini mengacu pada prinsip proses normalisasi dilakukan melalui beberapa tahapan bentuk normal (Normal Forms), umumnya hingga bentuk ketiga atau *Third Normal Form (3NF)* [22]. Menurut [22], normalisasi bertujuan menghilangkan anomali penyisipan, pembaruan, dan penghapusan yang sering terjadi dalam basis data yang tidak terstruktur dengan baik [22]. Dalam konteks sistem berskala menengah hingga besar seperti DMS, penerapan normalisasi memberikan keuntungan jangka panjang dalam hal efisiensi ruang penyimpanan, performa query, dan fleksibilitas integrasi modul. Dengan struktur basis data yang telah dinormalisasi, proses pengembangan menjadi lebih terkontrol, dan sistem yang dibangun dapat lebih mudah dikembangkan maupun dipelihara di masa depan.

id	order_code	distributor	address	weight	order_date
1	ORDER01	PT Bakery	Jakarta	1000	2025-06-01
2	ORDER02	PT Bakery	Jakarta	1200	2025-10-01

Tabel 3.2 Contoh Table Order Yang Tidak Normalisasi

Tabel 3.2 menunjukkan struktur tabel order yang belum dinormalisasi. Pada tabel tersebut, setiap baris mencatat informasi pesanan beserta data pelanggan secara berulang. Misalnya, untuk dua pesanan berbeda (ORDER01 dan ORDER02), data pelanggan yang sama *PT Bakery* di *Jakarta* dicatat ulang, meskipun entitas pelanggannya identik. Hal ini menimbulkan redundansi data, yang tidak hanya boros ruang penyimpanan, tetapi juga rentan terhadap inkonsistensi saat terjadi pembaruan informasi pelanggan.

Selain itu, struktur seperti ini menyulitkan dalam membangun relasi antar entitas seperti pelanggan, pesanan, dan barang. Oleh karena itu, diperlukan proses normalisasi, yakni pemisahan data ke dalam tabel-tabel yang merepresentasikan satu entitas logis, serta penggunaan kunci relasional (foreign key) untuk menghubungkan antar entitas tersebut. Dengan normalisasi, struktur data menjadi lebih modular, efisien, dan mudah dipelihara di masa depan.



Gambar 3.21 Contoh Table Yang Sudah Normalisasi

Struktur tabel hasil normalisasi yang ditampilkan dalam gambar 3.21 tersebut menggambarkan penerapan prinsip desain basis data relasional pada sistem Sales Order (SO) dalam proyek DMS. Normalisasi dilakukan dengan tujuan utama untuk menghindari redundansi data, menjaga konsistensi antar entitas, serta meningkatkan efisiensi pengelolaan transaksi dalam sistem yang kompleks dan saling berelasi. Diagram tersebut menunjukkan pembagian struktur menjadi empat tabel utama, yakni tblm_company, tblm_distributor, tblh_sales_order, dan tblt_so_detail, yang masing-masing merepresentasikan entitas yang terpisah namun saling terhubung secara logis melalui foreign key.

Tabel tblm_company menyimpan data terkait perusahaan pusat, seperti nama perusahaan, alamat, dan informasi identitas lainnya. Ini merupakan tabel referensi utama yang digunakan untuk mengaitkan setiap distributor yang terdaftar dalam sistem. Tabel tblm_distributor berisi informasi tentang distributor atau pelanggan, dengan kolom nid_company sebagai relasi ke tblm_company. Dengan adanya pemisahan ini, informasi perusahaan tidak perlu diulang di setiap entri distributor, sehingga mengurangi duplikasi data dan memudahkan perubahan informasi jika sewaktu-waktu ada pembaruan pada perusahaan induk.

Selanjutnya, tabel tblh_sales_order bertindak sebagai entitas header untuk transaksi penjualan. Tabel ini mencatat informasi umum dari sebuah sales order, seperti nomor dokumen (vcode), tanggal approval (vcrea), status, serta identitas distributor yang melakukan pemesanan. Data dalam tabel ini mencerminkan satu transaksi penjualan secara keseluruhan, dan akan memiliki beberapa baris detail tergantung pada jumlah item yang dipesan.

Detail transaksi barang tercatat pada tabel tblt_so_detail, yang memiliki relasi many-to-one dengan tblh_sales_order melalui nid_sales_order. Setiap baris di tabel ini menggambarkan satu item dalam transaksi, dengan rincian seperti harga, satuan berat, dan kuantitas, promosi, catatan, dan subtotal. Pemisahan ini memungkinkan sistem untuk memproses dan menganalisis data penjualan secara lebih fleksibel, baik untuk keperluan pencetakan invoice, laporan penjualan, maupun integrasi dengan modul pengiriman atau modul ERP.

Penerapan struktur ini sesuai dengan tahapan normalisasi hingga bentuk normal ketiga (3NF), di mana setiap entitas hanya memuat atribut yang saling bergantung secara fungsional terhadap primary key, dan tidak ada ketergantungan transitif antar kolom. Menurut teori yang dikemukakan oleh [22], desain seperti ini memfasilitasi efisiensi penyimpanan, menjaga integritas data antar modul, dan mempercepat proses pemrosesan query karena data tersimpan secara modular dan relasional. Struktur seperti ini juga krusial untuk mendukung integrasi lintas modul seperti modul pengiriman (DO), pembayaran, dan pelaporan (reporting & dashboard) yang membutuhkan keterkaitan data secara akurat dan efisien.

Sebagai bagian integral dari proses migrasi database dalam proyek ini, dilakukan juga penyesuaian struktur skema melalui perubahan nama tabel dan kolom agar lebih konsisten, terstruktur, dan mudah dipahami baik oleh tim pengembang frontend maupun backend. Penamaan tabel disesuaikan berdasar-

kan klasifikasi fungsionalnya, yaitu: awalan tblm_ digunakan untuk entitas *master data* seperti tblm_company dan tblm_distributor, awalan tblh_ untuk entitas *header* transaksi seperti tblh_sales_order, dan tblt_ untuk entitas *detail* dari transaksi seperti tblt_so_detail. Konvensi penamaan ini selaras dengan prinsip *database schema clarity* yang di mana struktur dan hierarki tabel yang jelas dapat meningkatkan keterbacaan dan mempercepat pemahaman arsitektur data bagi pengembang baru atau tim lintas fungsi.

Selain itu, kolom-kolom di dalam tabel juga mengikuti pola penamaan berdasarkan tipe datanya. Misalnya, awalan v pada kolom seperti vcode, vname, dan vcrea menandakan tipe data *varchar* atau *string/text*, n seperti pada nweight atau nstatus menunjukkan tipe numerik (*numeric*, *decimal*, atau *float*), sedangkan d seperti dorder_date atau dcrea merujuk pada tipe data tanggal. Pendekatan ini mengacu pada praktik *Hungarian Notation* yang disesuaikan untuk skema basis data, bertujuan untuk mempermudah identifikasi tipe data dan mempercepat debugging, pemrosesan query, serta pengembangan program antarmuka [23].

Dalam setiap tabel, secara konsisten terdapat dua kolom unik utama yaitu nid dan vcode. nid digunakan sebagai primary key (PK) bertipe integer auto-increment yang menjamin identitas unik setiap baris dalam database, sedangkan vcode berfungsi sebagai identifier unik yang bersifat fleksibel dan lebih manusiawi untuk kebutuhan operasional sistem, seperti penelusuran dokumen melalui frontend, integrasi antarmuka API, serta proses debugging pada sisi backend.

Lebih lanjut, setiap tabel juga dilengkapi dengan kolom pelacak status (nstatus) dan metadata perubahan (vcrea, dcrea, vmodi, dan dmodi). nstatus digunakan sebagai indikator aktif/tidak aktifnya data, mendukung pendekatan *soft delete* yang lebih aman dibanding *hard delete* karena memungkinkan data untuk tetap disimpan secara historis tanpa muncul di tampilan pengguna. Dalam tabel-tabel kompleks seperti tblh_sales_order, status dapat mencakup lebih dari

dua nilai, misalnya 0 (inactive), 1 (active), 2 (waiting approval), 3 (hold), 4 (waiting delivery), dan 5 (complete), yang mencerminkan fase dalam siklus transaksi. Sementara kolom vcrea, dcrea, vmodi, dan dmodi digunakan untuk mencatat jejak pengguna dan waktu perubahan data, yang penting dalam praktik *data audit trail* dan *accountability system*.

Praktik ini sejalan dengan prinsip-prinsip *data governance* dan *database maintainability* dalam sistem enterprise modern, yang menekankan pentingnya standar penamaan, auditabilitas, dan modularitas untuk mendukung keberlanjutan sistem jangka panjang. Dengan adanya struktur dan standar seperti ini, tidak hanya pengembangan menjadi lebih terarah, tetapi juga proses debugging, dokumentasi teknis, hingga migrasi data di masa mendatang dapat dilakukan dengan lebih efisien dan minim risiko.

Dengan fondasi struktur database yang sudah dirancang secara sistematis dan modular, langkah selanjutnya dalam proses pengembangan sistem adalah membangun mekanisme autentikasi dan otorisasi yang aman dan fleksibel. Tahapan ini menjadi krusial karena seluruh fitur dalam sistem DMS, termasuk modul-modul transaksi dan monitoring, membutuhkan kontrol akses berbasis peran (role-based access control/RBAC) untuk menjamin keamanan data serta membatasi hak akses pengguna sesuai dengan tanggung jawab dan fungsi operasionalnya. Implementasi fitur seperti login, register, dan manajemen akun distributor tidak hanya menjadi pintu masuk utama bagi pengguna ke dalam sistem, tetapi juga menjadi elemen fundamental dalam menjaga integritas dan privasi data yang dikelola oleh aplikasi. Oleh karena itu, pembahasan selanjutnya akan mengulas proses pengembangan modul autentikasi beserta integrasinya dengan fitur manajemen user dan monitoring distributor.

Sebagai bagian dari sistem authentication pada sistem DMS, proses registrasi dalam sistem DMS dirancang secara bertahap untuk memastikan bahwa struktur distribusi terorganisir dan sesuai dengan alur bisnis yang berlaku. Tahap pertama

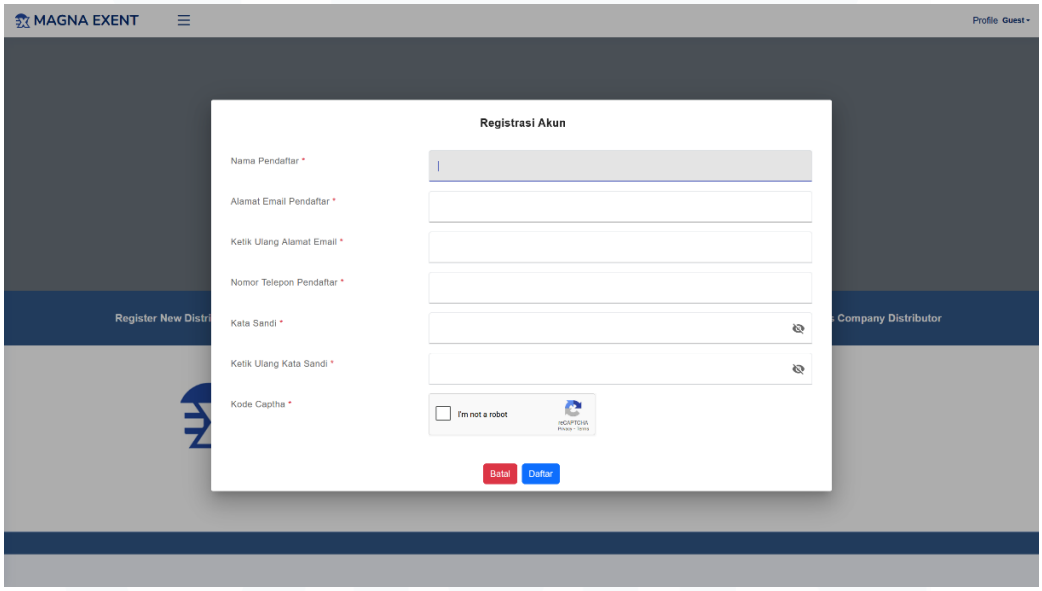
dimulai dari registrasi awal pada halaman publik, di mana entitas perusahaan dapat mengajukan pendaftaran sebagai calon distributor. Pada tahap ini, perusahaan wajib mengisi data utama seperti nama perusahaan, alamat, kontak, serta dokumen pendukung yang diperlukan sebagai bagian dari proses verifikasi. Setelah akun perusahaan berhasil didaftarkan, sistem akan memberikan akses sebagai Company Distributor.

Tahap kedua memungkinkan Company Distributor yang telah terverifikasi untuk mendaftarkan distributor di bawah naungannya. Dalam struktur ini, satu perusahaan dapat memiliki banyak distributor cabang atau mitra yang tersebar di berbagai wilayah. Fitur ini memberikan fleksibilitas dalam membentuk hierarki distribusi dan mendukung manajemen wilayah distribusi secara lebih komprehensif. Meskipun Company Distributor memiliki wewenang untuk mendaftarkan entitas di bawahnya, sistem tetap memberlakukan pengendalian akses melalui proses validasi.

Pada tahap ketiga, setiap distributor yang didaftarkan oleh Company Distributor akan masuk ke dalam daftar permohonan yang memerlukan persetujuan (approval) dari administrator sistem. Distributor yang diusulkan harus melengkapi data-data penting seperti profil distributor, wilayah operasional, jenis produk, dan informasi legal lainnya. Setelah seluruh data tervalidasi dan diverifikasi oleh admin, maka status distributor akan diubah menjadi aktif, dan akun akses ke sistem DMS akan diberikan secara resmi. Proses bertahap ini bertujuan untuk menjaga kualitas jaringan distributor, memastikan data yang masuk telah diverifikasi, serta meminimalkan risiko penyalahgunaan akses dalam sistem distribusi yang kompleks.

Sistem registrasi bertahap yang diterapkan dalam proyek ini mengacu pada prinsip *Identity Lifecycle Management (ILM)*, yang menekankan pentingnya pengelolaan identitas digital secara sistematis mulai dari proses pendaftaran

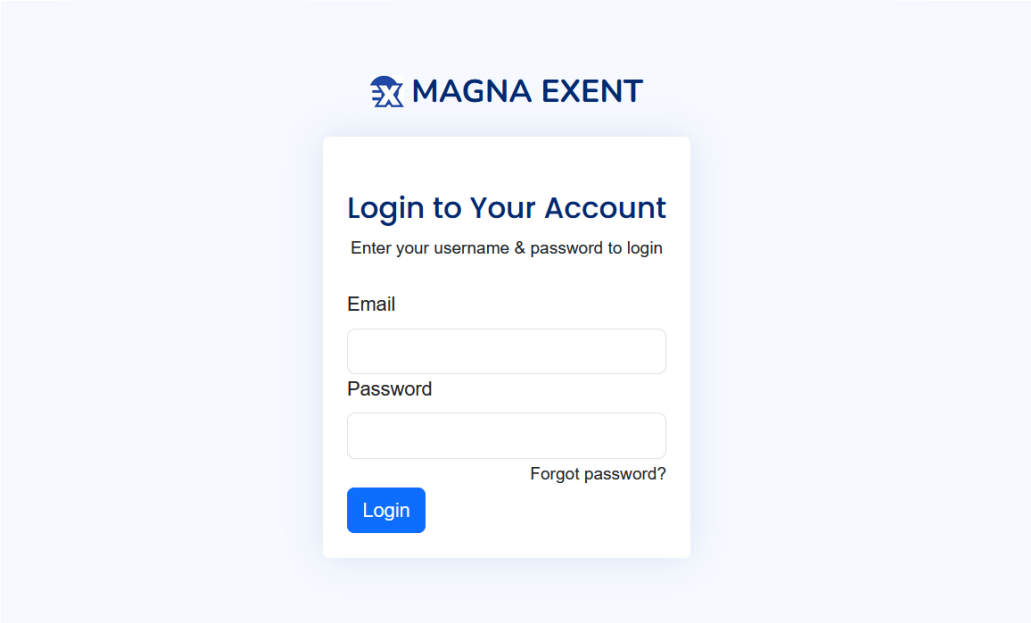
hingga pemberian hak akses sesuai peran. Dalam konteks ini, pendekatan bertahap mendukung penerapan *Role-Based Access Control (RBAC)*, di mana hak akses pengguna ditentukan berdasarkan struktur organisasi dan fungsi operasionalnya. Selain itu, konsep validasi berlapis pada setiap tahap registrasi mencerminkan penerapan *Access Governance*, yang bertujuan memastikan bahwa setiap entitas dalam sistem memperoleh akses hanya setelah melalui proses otorisasi dan verifikasi yang sesuai kebijakan keamanan sistem. Pendekatan ini tidak hanya memperkuat keamanan data dan integritas sistem, tetapi juga selaras dengan praktik terbaik dalam pengembangan aplikasi enterprise yang mengedepankan akuntabilitas, skalabilitas, dan kontrol akses yang terstruktur [24].



Gambar 3.22 Form Registrasi Halaman Depan

Gambar 3.22 menampilkan antarmuka halaman registrasi untuk *Company Distributor*, di mana proses pendaftaran dimulai dengan pengisian formulir oleh perusahaan yang ingin bergabung ke dalam sistem. Setelah formulir dikirimkan, sistem secara otomatis akan mengirimkan email verifikasi ke alamat email yang terdaftar. Apabila proses verifikasi berhasil diselesaikan, perusahaan akan

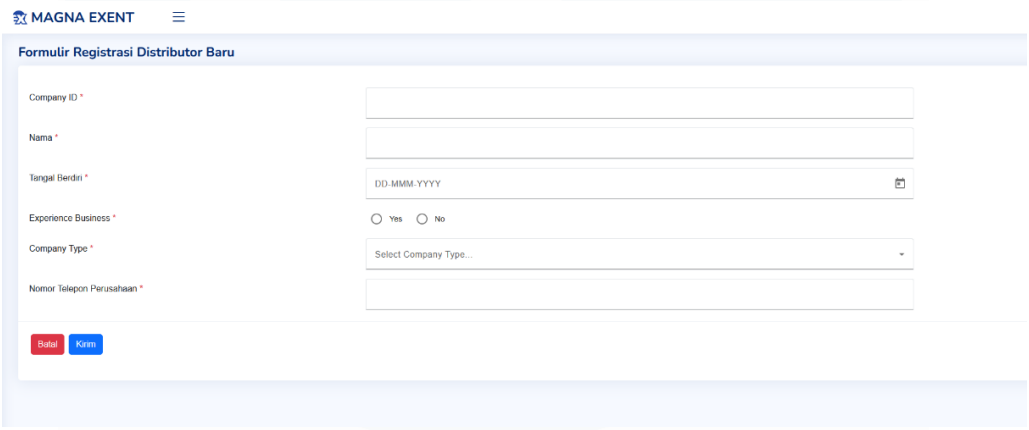
diberikan akses untuk masuk (login) ke dalam aplikasi. Dalam sistem ini, terdapat dua jenis akun login, yaitu akun *Company Distributor* dan akun *Staff*. Akun *Company Distributor* digunakan oleh perusahaan utama yang telah berhasil mendaftar dan terverifikasi, sedangkan akun *Staff* diperuntukkan bagi pengguna atau karyawan yang berada di bawah naungan distributor tersebut. Pembagian tipe akun ini dimaksudkan untuk mengatur hak akses, otorisasi fitur, serta tanggung jawab pengguna sesuai dengan struktur organisasi distribusi yang berlaku. Adapun Gambar 3.23 menampilkan antarmuka halaman login yang digunakan untuk mengakses sistem berdasarkan jenis akun masing-masing.



Gambar 3.23 Tampilan Halaman Login

Pada halaman login, sistem juga menyediakan fitur lupa kata sandi (*forgot password*) sebagai bagian dari mekanisme pemulihan akun yang aman dan user-friendly. Fitur ini memungkinkan pengguna yang tidak dapat mengakses akunnya karena lupa kata sandi untuk melakukan permintaan reset melalui tautan khusus yang tersedia di halaman login. Setelah pengguna memasukkan alamat email yang terdaftar, sistem akan mengirimkan email verifikasi berisi tautan untuk melakukan proses pengaturan ulang kata sandi. Tautan ini memiliki masa

berlaku tertentu demi menjaga keamanan, dan hanya dapat digunakan satu kali. Setelah proses verifikasi berhasil, pengguna akan diarahkan ke halaman untuk membuat kata sandi baru. Pendekatan ini tidak hanya memudahkan pemulihan akun, tetapi juga merupakan bagian dari penerapan prinsip *account recovery flow* yang umum diterapkan dalam sistem autentikasi modern. Dengan adanya fitur ini, sistem menjadi lebih inklusif, mendukung keberlanjutan akses pengguna, serta tetap menjaga keamanan dan integritas akun secara keseluruhan.

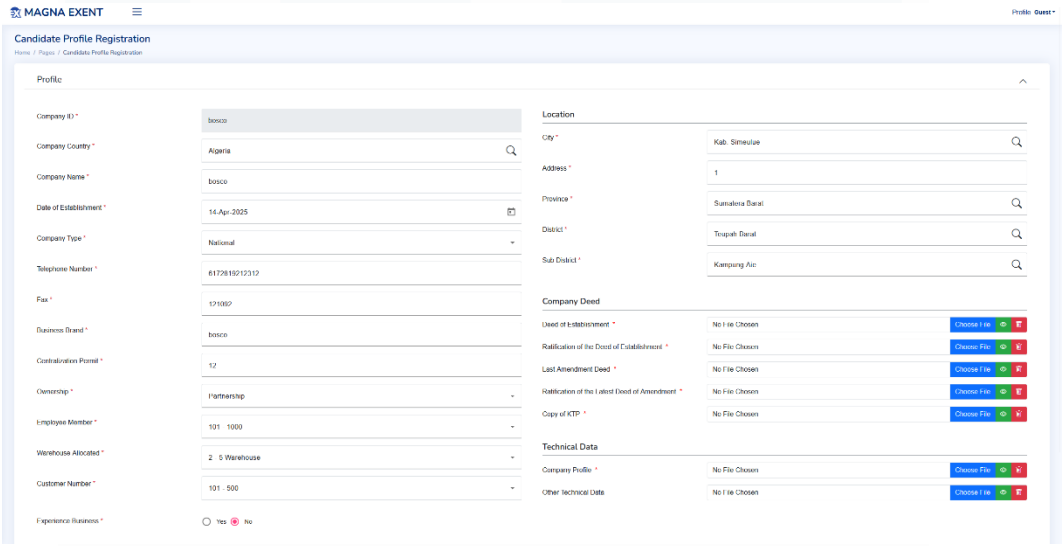


Gambar 3.24 Tampilan Halaman Registrasi Kandidat Distributor

Ketika sebuah *Company Distributor* ingin mendaftarkan distributor di bawah naungannya, proses tersebut akan dimulai melalui halaman seperti yang ditampilkan pada Gambar 3.24. Pada halaman ini, *Company Distributor* dapat mengisi formulir pendaftaran distributor dan mengirimkan permintaan registrasi. Setelah formulir dikirim, sistem akan mengirimkan email verifikasi kepada kandidat distributor sebagai bagian dari proses validasi awal. Setelah email berhasil diverifikasi, status distributor akan tercatat sebagai kandidat yang masih dalam tahap evaluasi.

Selanjutnya, kandidat distributor akan diminta untuk melengkapi informasi profil secara menyeluruh. Pengisian data profil dilakukan melalui dua tahapan, yaitu halaman data profil umum seperti yang ditampilkan pada Gambar 3.25, dan halaman data finansial seperti yang ditunjukkan pada Gambar 3.26. Kedua

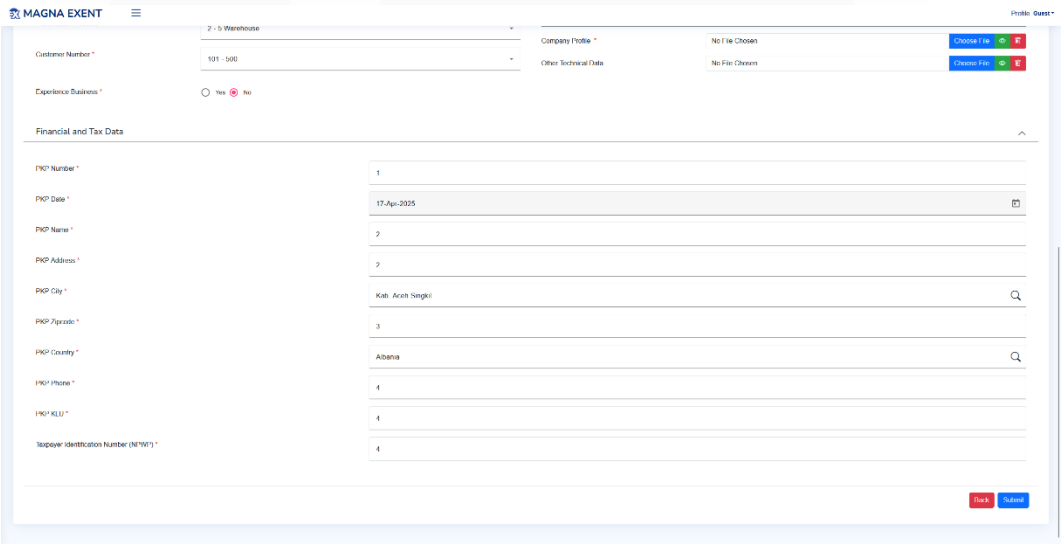
bagian ini mencakup informasi penting terkait identitas distributor, alamat operasional, dokumen legal, serta data keuangan dan rekening pembayaran. Setelah seluruh data terisi dengan lengkap dan valid, tim administrator akan melakukan evaluasi terhadap kelayakan kandidat berdasarkan informasi yang telah diberikan. Apabila hasil evaluasi dinyatakan sesuai dan memenuhi kriteria, maka kandidat tersebut akan resmi terdaftar sebagai distributor aktif dan memperoleh akses login ke dalam sistem DMS.



Gambar 3.25 Halaman Profile Kandidat Distributor

Gambar 3.25 menampilkan halaman "Candidate Profile Registration" pada sistem DMS, yang digunakan untuk mengisi data profil kandidat distributor. Halaman ini terdiri dari beberapa bagian utama. Di bagian kiri terdapat informasi umum perusahaan seperti ID perusahaan, negara asal, nama perusahaan, tanggal pendirian, jenis perusahaan, nomor telepon, faks, dan data dasar lainnya seperti bidang usaha, format konstitusi, dan bentuk kepemilikan. Selain itu, terdapat juga data jumlah karyawan, jumlah gudang, nomor pelanggan, dan pengalaman dalam hal pelelangan. Sementara itu, di bagian kanan halaman terdapat informasi lokasi perusahaan yang mencakup kota, alamat, provinsi, distrik, dan sub distrik. Di bawahnya, tersedia bagian untuk mengunggah dokumen legal seperti

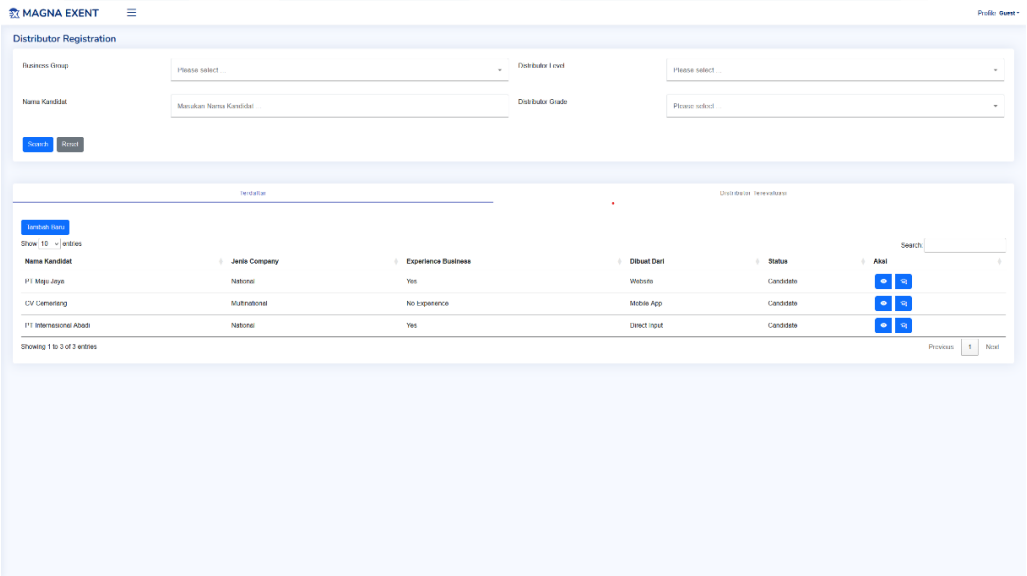
akta pendirian, pernyataan perubahan, dan salinan NPWP. Bagian terakhir adalah “Technical Data” yang memungkinkan pengguna mengunggah profil perusahaan dan data teknis lainnya. Semua data tersebut bersifat wajib diisi untuk melengkapi pendaftaran sebagai kandidat distributor.



Gambar 3.26 Halaman Profile Kandidat Distributor Bagian Financial

Gambar 3.26 ini masih merupakan bagian dari halaman "Candidate Profile Registration", lebih tepatnya pada bagian Financial and Tax Data atau data keuangan dan perpajakan calon distributor. Pada bagian ini, pengguna diminta untuk mengisi informasi terkait PKP (Pengusaha Kena Pajak) dan data perpajakan lainnya. Beberapa kolom yang harus diisi meliputi PKP Number, PKP Date (tanggal terdaftar sebagai PKP), PKP Name, dan PKP Address. Selanjutnya terdapat kolom untuk PKP City, PKP District, PKP Country, PKP Phone, serta PKP KLU (Kode Lapangan Usaha sesuai klasifikasi pajak). Bagian terakhir adalah kolom untuk Taxpayer Identification Number (NPWP), yang merupakan nomor identifikasi wajib pajak perusahaan. Di bagian bawah terdapat tombol navigasi Back dan Submit untuk menyimpan atau kembali ke halaman sebelumnya. Informasi ini penting untuk memastikan bahwa calon distributor memiliki legalitas dan kewajiban pajak yang jelas dan sesuai dengan peraturan.

Halaman ini berfungsi sebagai formulir registrasi lengkap bagi calon distributor yang ingin didaftarkan oleh Company. Melalui halaman ini, admin dapat mengumpulkan data administratif, legalitas, lokasi, teknis, hingga informasi perpajakan dari kandidat distributor secara terstruktur dan terdokumentasi dengan baik. Tujuannya adalah untuk melakukan proses seleksi, verifikasi, dan evaluasi terhadap kelayakan distributor secara transparan dan efisien sebelum menjalin kerja sama lebih lanjut.



Gambar 3.27 Halaman Monitoring Registrasi Distributor

Gambar 3.27 menampilkan halaman Distributor Registration yang berfungsi sebagai panel monitoring bagi admin untuk melihat dan memantau seluruh kandidat distributor yang telah melakukan proses registrasi. Setelah kandidat mengisi dan mengirimkan formulir registrasi secara lengkap, data mereka akan secara otomatis masuk ke halaman ini. Pada halaman ini terdapat fitur pencarian dan filter berdasarkan *Business Group*, *Distributor Level*, *Nama Kandidat*, dan *Distributor Grade* untuk memudahkan admin dalam mengelola data. Bagian bawah halaman menampilkan tabel daftar kandidat yang mencakup informasi seperti nama kandidat, jenis perusahaan, pengalaman bisnis, asal data (misalnya

dari website atau input langsung), serta statusnya yang saat ini masih sebagai "Candidate".

Halaman ini dibagi menjadi dua bagian utama, yaitu Terdaftar dan Distributor Terevaluasi. Semua kandidat baru yang belum melalui proses evaluasi akan muncul di tab "Terdaftar", seperti yang terlihat dalam gambar 3.27. Admin dapat meninjau data setiap kandidat, menilai kelayakan mereka, dan memproses lebih lanjut untuk menjadi distributor resmi. Setelah proses evaluasi selesai dan kandidat dinyatakan lolos, status mereka akan berpindah ke bagian Distributor Terevaluasi. Hal ini memungkinkan pemisahan yang jelas antara kandidat yang masih dalam tahap seleksi dan yang sudah resmi bekerja sama sebagai distributor. Dengan adanya halaman ini, proses seleksi menjadi lebih terorganisir, transparan, dan mudah dikendalikan oleh pihak yang berwenang.

Setelah proses evaluasi terhadap distributor diselesaikan dan status kandidat diperbarui menjadi distributor resmi, seluruh data dan aktivitas yang berkaitan dengan registrasi, validasi, serta pengelolaan distributor akan terdokumentasi secara sistematis dalam sistem. Fitur ini tidak hanya mendukung akuntabilitas, tetapi juga menjadi bagian dari *audit trail* yang penting untuk menjamin transparansi, keterlacakan proses, dan pengawasan operasional. Dengan demikian, halaman manajemen distributor tidak hanya berfungsi sebagai alat administratif, melainkan juga sebagai fondasi untuk pengelolaan relasi bisnis yang profesional dan terstruktur.

Keberadaan modul-modul ini menandai bahwa sebagian besar proses pengembangan inti telah terselesaikan. Namun, sebelum sistem dapat dianggap siap untuk digunakan secara penuh, diperlukan tahapan finalisasi yang mencakup kegiatan bug fixing, unit testing, dan penyusunan dokumentasi teknis. Bug fixing dilakukan untuk memperbaiki kesalahan fungsional atau tampilan yang ditemukan selama pengujian internal, sementara unit testing bertujuan untuk menguji setiap komponen secara terpisah guna memastikan bahwa logika

dan output yang dihasilkan sesuai dengan spesifikasi yang telah ditentukan. Bersamaan dengan itu, dokumentasi sistem disusun untuk mencatat struktur modul, alur data, integrasi antarmuka, serta hasil pengujian yang telah dilakukan.

Tahapan finalisasi ini menjadi langkah krusial dalam memastikan bahwa sistem tidak hanya berfungsi secara teoritis, tetapi juga siap dioperasikan dalam lingkungan produksi yang sesungguhnya. Subbab berikutnya akan mengulas secara rinci proses pengujian teknis, penanganan bug, serta penyusunan dokumentasi sebagai penutup dari siklus pengembangan aplikasi DMS ini.

3.2.1.3 Tahap Final & Dokumentasi

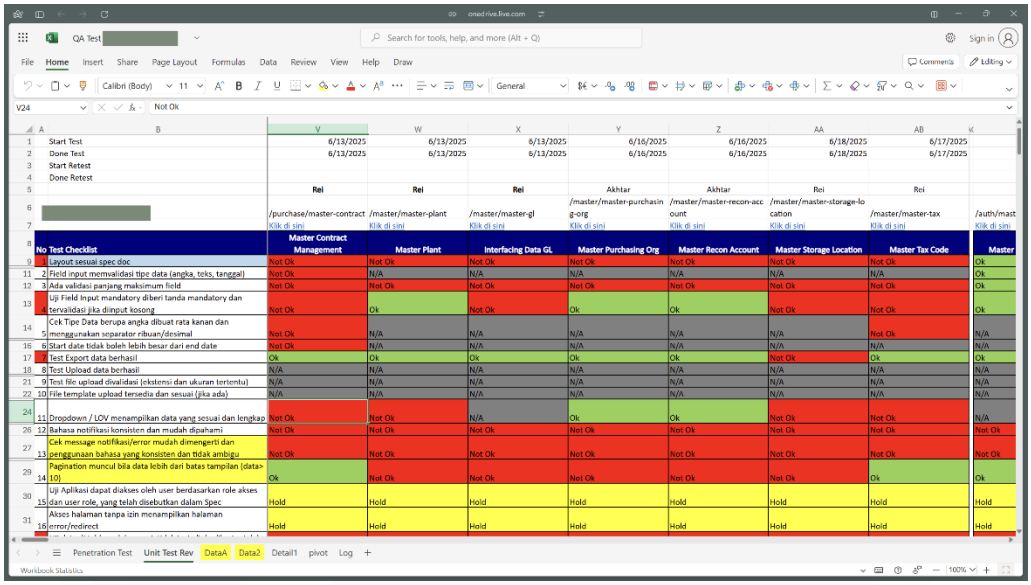
Sebagai kelanjutan dari tahapan pengembangan dan evaluasi fitur-fitur utama dalam sistem DMS, proses berikutnya difokuskan pada tahap akhir yaitu finalisasi dan dokumentasi. Setelah seluruh modul mulai dari registrasi, autentikasi, hingga manajemen distributor berhasil dikembangkan dan diuji secara fungsional, diperlukan langkah-langkah penutup untuk memastikan bahwa sistem siap diimplementasikan secara menyeluruh. Tahap ini mencakup proses perbaikan terhadap bug yang ditemukan selama pengujian, pelaksanaan unit test secara menyeluruh, serta penyusunan dokumentasi teknis yang akan menjadi acuan penting bagi pengembang selanjutnya maupun pihak operasional. Finalisasi ini menjadi penanda bahwa sistem telah melewati siklus pengembangan inti dan siap dipindahkan menuju lingkungan produksi dengan tingkat keandalan yang sesuai standar sistem informasi enterprise.

Proses *unit testing* menjadi bagian yang sangat penting untuk memastikan bahwa setiap komponen dalam aplikasi berfungsi sesuai dengan spesifikasi yang telah ditetapkan. Pada proyek ini, pelaksanaan unit test dilakukan secara kolaboratif, di mana sebagian besar proses pengujian ditangani oleh tim Business Analyst (BA) dan Quality Assurance (QA) yang memiliki pemahaman mendalam terhadap kebutuhan bisnis dan alur sistem, serta didampingi oleh tim *Fullstack Developer* untuk mendukung aspek teknis, terutama dalam verifikasi logika dan

respons antarmuka. Unit test yang dilakukan masih bersifat manual, yakni dengan cara menjalankan skenario-skenario penggunaan secara langsung pada antarmuka aplikasi, kemudian mencatat hasil pengujian tersebut ke dalam dokumen uji sebagai bukti validasi sistem.

Unit testing merupakan salah satu pendekatan dasar dalam pengujian perangkat lunak, yang bertujuan untuk memverifikasi apakah satuan kode terkecil (unit), seperti fungsi modul, atau komponen bekerja dengan benar secara terisolasi [25]. Menurut [25], unit test membantu mendeteksi kesalahan pada tahap awal sebelum modul-modul digabungkan dalam skala yang lebih besar, sehingga mengurangi biaya perbaikan dan meningkatkan reliabilitas sistem. Meskipun belum menggunakan otomasi, pelaksanaan unit test secara sistematis dalam bentuk manual tetap memiliki kontribusi besar terhadap stabilitas aplikasi, khususnya dalam sistem DMS yang memiliki banyak alur transaksi dan keterkaitan antar modul. Hasil dari pengujian ini juga digunakan sebagai dasar untuk menyusun dokumentasi teknis, yang menjadi landasan penting bagi tim pengembang dan pihak pengguna dalam proses implementasi serta pemeliharaan sistem di masa depan.

Dokumen dibuat menggunakan google spreadsheet seperti yang ditampilkan pada gambar 3.28 merupakan bagian dari proses Unit testing dalam bentuk spreadsheet, yang digunakan untuk mengevaluasi dan memastikan bahwa berbagai modul dalam sistem telah berfungsi sesuai dengan standar dan spesifikasi yang ditentukan sebelum sistem digunakan secara resmi. Pengujian ini dilakukan terhadap beberapa semua modul penting. Masing-masing modul diuji oleh tim yang berbeda, sebagaimana ditunjukkan oleh nama-nama penanggung jawab seperti *Rei*, *Akhtar*, dan *AA*, beserta path modul yang diuji.

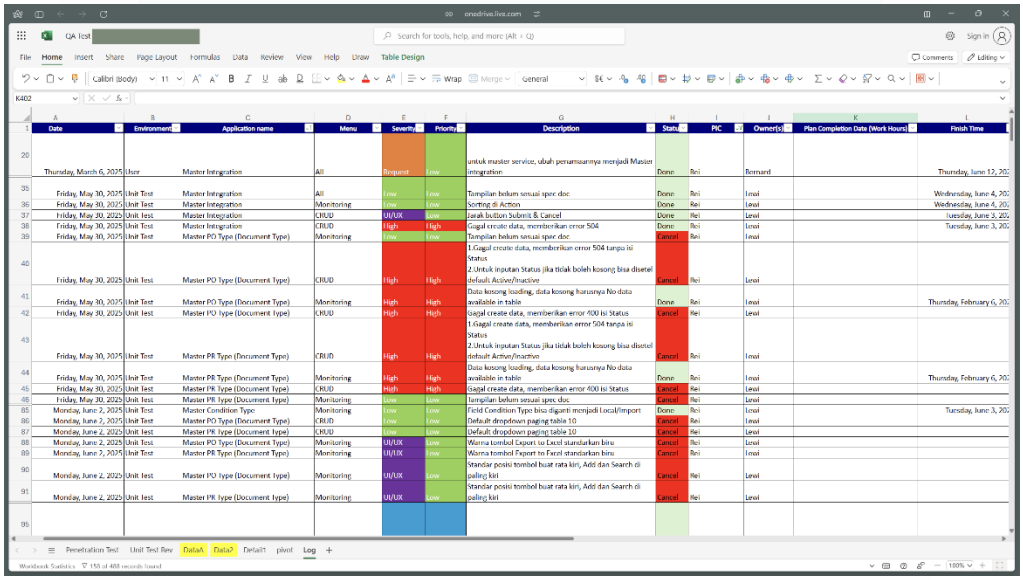


Gambar 3.28 Ilustrasi Document Testing

Bagian atas dokumen mencantumkan tanggal dimulainya pengujian, tanggal selesainya, serta waktu pelaksanaan retest jika diperlukan. Sementara itu, bagian utama dokumen berisi tabel yang terdiri dari checklist pengujian pada kolom pertama dan hasil pengujian untuk tiap modul pada kolom-kolom berikutnya. Checklist tersebut meliputi 29 poin pengujian yang mencakup aspek validasi input (misalnya angka, teks, dan tanggal), keharusan pengisian field mandatory, tampilan antarmuka, kesesuaian dropdown atau list of value (LOV), serta fungsi tombol-tombol penting seperti simpan dan hapus. Selain itu, pengujian juga mencakup kriteria seperti penamaan field yang konsisten, hasil upload yang sesuai, pengurutan data, dan aspek keamanan seperti pembatasan akses berdasarkan role.

Hasil dari masing-masing pengujian ditandai dengan kode warna yang mudah dibaca, yaitu "Not OK" (merah) untuk modul yang gagal dalam pengujian, "OK" (hijau) untuk modul yang lolos, "N/A" (abu-abu) untuk poin yang tidak berlaku, dan "Hold" (kuning) untuk pengujian yang tertunda atau belum dilaksanakan. Pendekatan visual ini memudahkan tim QA dan pengembang dalam mengiden-

tifikasi area yang bermasalah, melakukan tindak lanjut perbaikan, serta memastikan bahwa sistem dalam kondisi stabil dan siap digunakan. Secara keseluruhan, dokumen ini berfungsi sebagai alat dokumentasi dan kontrol kualitas yang esensial dalam proses pengembangan sistem, guna menjaga integritas dan keandalan aplikasi sebelum digunakan oleh end user.



Gambar 3.29 Ilustrasi Dokumen Log Fixing Bugs

Gambar 3.29 yang ditampilkan adalah bug fixing log dalam bentuk spreadsheet yang digunakan selama proses finalisasi untuk mencatat dan memantau progress perbaikan atas temuan bug atau isu yang muncul selama pengujian sistem. Log ini merupakan alat penting dalam siklus pengembangan perangkat lunak karena memberikan visibilitas yang jelas terkait jenis masalah, status penanganannya, serta siapa yang bertanggung jawab atas penyelesaiannya. Dokumen ini mencatat informasi secara terstruktur seperti tanggal temuan, lingkungan pengujian (environment), nama aplikasi, menu modul yang terpengaruh, jenis bug (CRUD atau UI/UX), tingkat severity (dampak), prioritas penanganan, deskripsi masalah, status penyelesaian (done, hold, etc.), serta nama PIC dan owner. Juga terdapat kolom untuk mencatat estimasi waktu penyelesaian dan tanggal bug tersebut dinyatakan selesai.

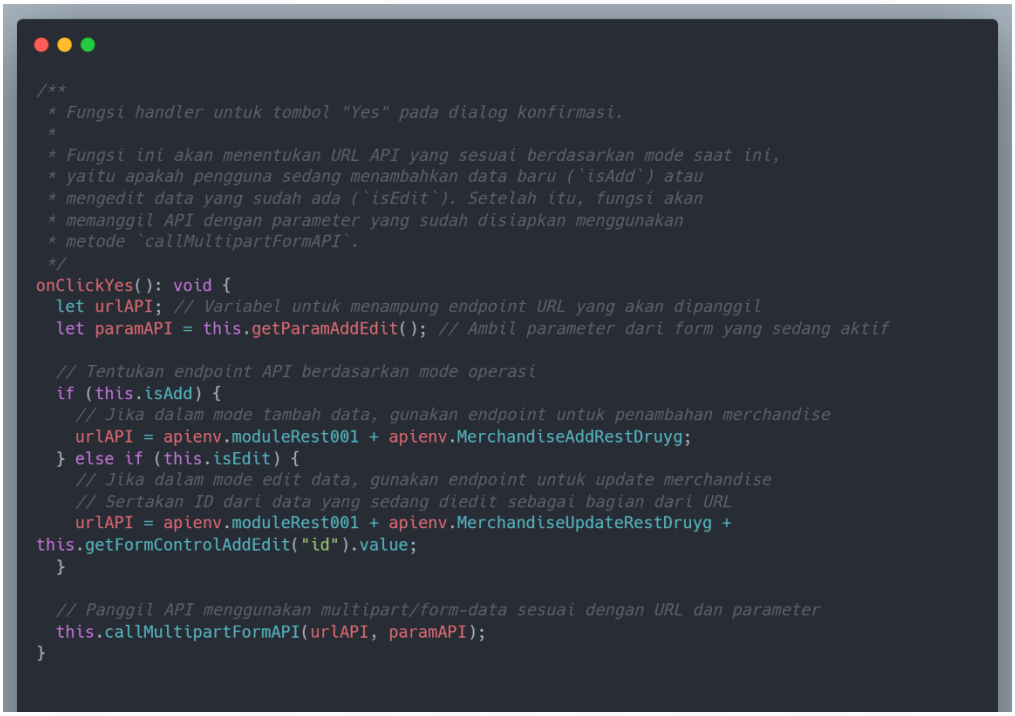
Pentingnya dokumentasi log seperti ini didukung oleh prinsip-prinsip manajemen proyek perangkat lunak, di mana pencatatan bug dan tracking perbaikannya merupakan bagian dari proses defect management yang sistematis. Proses pengelolaan bug harus terdokumentasi secara lengkap untuk memastikan bahwa setiap defect yang teridentifikasi selama pengujian ditindaklanjuti hingga tuntas, serta untuk memberikan akuntabilitas terhadap tim pengembang.

Dengan menggunakan bug log seperti pada gambar, tim QA dan pengembang dapat dengan mudah meninjau masalah-masalah yang masih aktif (open/hold), yang sudah diperbaiki (done), serta mengevaluasi efektivitas proses penanganan berdasarkan severity dan prioritas. Hal ini tidak hanya meningkatkan efisiensi tim dalam menyelesaikan bug, tetapi juga mendukung transparansi dan komunikasi yang baik antar tim, serta mencegah bug serupa terjadi kembali di masa depan.

Sebagai tahap akhir dari proses perbaikan bug dan penyempurnaan sistem, dokumentasi kode menjadi elemen yang sangat krusial untuk memastikan bahwa seluruh tim pengembang, baik saat ini maupun di masa depan, dapat memahami alur logika dan implementasi yang dilakukan. Dokumentasi kode biasanya mencakup penjelasan dalam bentuk komentar pada bagian-bagian penting dalam kode, seperti algoritma utama, validasi, struktur kontrol, serta alasan di balik pengambilan keputusan teknis tertentu. Komentar ini membantu memperjelas fungsi dari setiap bagian kode dan mengurangi risiko kesalahan saat dilakukan pemeliharaan, pengembangan lanjutan, atau refactoring. Dokumentasi yang baik secara langsung dalam kode mendorong pemahaman kolektif dan mempercepat proses debugging serta pengujian ulang.

Dokumentasi kode yang baik memungkinkan pengembang untuk menelusuri kembali riwayat perbaikan berdasarkan entri log yang ada. Ketika sebuah bug telah diperbaiki, maka sangat disarankan untuk menambahkan komentar

pada baris kode yang mengalami perubahan, yang menjelaskan alasan perubahannya dan referensi ke nomor tiket atau deskripsi log. Dengan demikian, log tidak hanya berfungsi sebagai catatan historis eksternal, tetapi juga tersambung erat ke dalam kode itu sendiri melalui dokumentasi internal. Pendekatan ini membantu menjaga *traceability* dan *accountability*, serta memperkuat kualitas dan keandalan sistem secara keseluruhan. Dokumentasi kode yang rapi juga menjadi bentuk komunikasi lintas waktu antar developer, terutama dalam proyek jangka panjang atau sistem berskala besar.



```
/**
 * Fungsi handler untuk tombol "Yes" pada dialog konfirmasi.
 *
 * Fungsi ini akan menentukan URL API yang sesuai berdasarkan mode saat ini,
 * yaitu apakah pengguna sedang menambahkan data baru (`isAdd`) atau
 * mengedit data yang sudah ada (`isEdit`). Setelah itu, fungsi akan
 * memanggil API dengan parameter yang sudah disiapkan menggunakan
 * metode `callMultipartFormAPI`.
 */
onClickYes(): void {
  let urlAPI; // Variabel untuk menampung endpoint URL yang akan dipanggil
  let paramAPI = this.getParamAddEdit(); // Ambil parameter dari form yang sedang aktif

  // Tentukan endpoint API berdasarkan mode operasi
  if (this.isAdd) {
    // Jika dalam mode tambah data, gunakan endpoint untuk penambahan merchandise
    urlAPI = apienv.moduleRest001 + apienv.MerchandiseAddRestDruyg;
  } else if (this.isEdit) {
    // Jika dalam mode edit data, gunakan endpoint untuk update merchandise
    // Sertakan ID dari data yang sedang diedit sebagai bagian dari URL
    urlAPI = apienv.moduleRest001 + apienv.MerchandiseUpdateRestDruyg +
this.getFormControlAddEdit("id").value;
  }

  // Panggil API menggunakan multipart/form-data sesuai dengan URL dan parameter
  this.callMultipartFormAPI(urlAPI, paramAPI);
}
```

Gambar 3.30 Contoh Dokumentasi Kode

Pada gambar 3.30 yang ditampilkan merupakan dokumentasi kode dari sebuah fungsi bernama onClickYes(), yang berfungsi sebagai handler (penangan aksi) ketika pengguna menekan tombol "Yes" pada sebuah dialog konfirmasi dalam aplikasi. Dokumentasi kode ini dibuat dengan sangat rapi dan terstruktur, menggunakan komentar blok di bagian atas untuk menjelaskan secara umum maksud dan fungsi utama dari metode tersebut, serta komentar baris per baris untuk menjelaskan logika spesifik dalam implementasinya.

Secara garis besar, fungsi onClickYes() bertugas untuk menentukan endpoint (alamat tujuan API) yang sesuai, berdasarkan mode aplikasi saat itu—apakah pengguna sedang melakukan penambahan data (isAdd) atau pengeditan data (isEdit). Bila pengguna berada dalam mode tambah, maka akan dipanggil endpoint untuk menambahkan merchandise; sedangkan jika dalam mode edit, maka endpoint yang dipanggil adalah untuk memperbarui data merchandise, dengan menyertakan ID dari data yang sedang diedit. Penjelasan teknis dalam dokumentasi ini sangat membantu pembaca kode, baik pengembang baru maupun tim QA, karena menjelaskan maksud setiap bagian logika secara mendalam. Misalnya, pemisahan deklarasi urlAPI dan paramAPI telah diberi keterangan yang jelas mengenai fungsinya. Bahkan, proses pemanggilan API di akhir fungsi juga diberikan komentar untuk memperjelas bahwa metode callMultipartFormAPI akan menjalankan permintaan API dengan format multipart/form-data menggunakan parameter yang sudah ditentukan sebelumnya.

Dokumentasi kode seperti ini mencerminkan praktik pengembangan perangkat lunak yang profesional dan berkelanjutan. Dengan gaya dokumentasi yang konsisten di seluruh fungsi, proyek menjadi lebih mudah dipelihara, dipahami, dan dikembangkan secara kolaboratif. Setiap anggota tim dapat menelusuri logika teknis tanpa harus membaca keseluruhan kode. Meskipun dokumentasi telah disusun dengan baik, proses implementasi tetap menghadapi berbagai kendala. Oleh karena itu, bagian selanjutnya akan membahas tantangan yang muncul serta solusi yang diterapkan untuk mengatasinya.

Sebagai hasil dari pelaksanaan program magang, telah dikembangkan sebuah aplikasi Distributor Management System (DMS) berbasis web yang dirancang untuk mengatasi berbagai tantangan distribusi yang selama ini dihadapi oleh perusahaan. Sistem ini tidak hanya bertujuan menggantikan proses manual yang tersebar, tetapi juga mendorong digitalisasi menyeluruh terhadap proses distribusi, mulai dari registrasi distributor hingga pelaporan transaksi. Pendekatan pengembangan menggunakan arsitektur microservices memungkinkan sistem dibangun secara

modular, mempermudah pengelolaan, dan memastikan skalabilitas seiring pertumbuhan kebutuhan bisnis.

Dari sisi kapabilitas, sistem memberikan enam kontribusi utama. Pertama, proses akuisisi distributor kini berjalan lebih cepat dan transparan. Melalui alur registrasi bertahap dan approval administratif, proses yang sebelumnya memerlukan 3 hingga 5 hari dapat diselesaikan dalam waktu kurang dari 24 jam. Hal ini tidak hanya mempercepat onboarding mitra baru, tetapi juga meningkatkan validitas data entitas yang masuk ke dalam sistem. Kedua, sistem otorisasi pengguna berbasis peran (role-based access control) memastikan bahwa setiap pengguna hanya dapat mengakses fitur sesuai tanggung jawabnya, sehingga mengurangi potensi penyalahgunaan dan menjaga integritas data internal.

Ketiga, visibilitas operasional meningkat secara signifikan. Seluruh proses distribusi, mulai dari pengelolaan data pelanggan, transaksi pemesanan, pengiriman barang, hingga penagihan dapat dipantau secara real-time melalui antarmuka yang terintegrasi. Dengan ini, proses pengambilan keputusan yang sebelumnya mengandalkan laporan mingguan atau bulanan kini dapat dilakukan dalam hitungan jam, karena seluruh informasi selalu diperbarui secara otomatis. Keempat, sinkronisasi data antar modul dan kompatibilitas ekspor data dalam format CSV atau API mendukung keterhubungan sistem secara lintas divisi maupun dengan sistem eksternal seperti ERP. Kondisi ini mengurangi beban pekerjaan manual dan meminimalkan risiko inkonsistensi data.

Kelima, efisiensi dokumentasi meningkat tajam melalui fitur pelaporan otomatis. Pengguna kini dapat menghasilkan dokumen seperti faktur, daftar pengiriman, dan bukti transaksi hanya dalam hitungan detik menggunakan JasperReport. Sebelumnya, dokumen-dokumen ini disiapkan secara manual dan membutuhkan waktu hingga satu hari kerja, serta melibatkan risiko kesalahan input. Keenam, arsitektur microservices yang diimplementasikan terbukti memberikan fleksibilitas tinggi dalam proses pengembangan dan pemeliharaan. Setiap modul dapat di-deploy, diuji,

dan disesuaikan secara terpisah tanpa mengganggu sistem utama, sehingga mempercepat iterasi fitur baru dan meminimalkan downtime sistem secara keseluruhan.

Berdasarkan hasil observasi dan simulasi proses kerja antara sistem lama dan sistem baru, ditemukan bahwa rata-rata waktu yang dibutuhkan untuk menyelesaikan alur kerja administratif, seperti proses validasi distributor, pembuatan laporan distribusi, dan penyusunan dokumen transaksi mengalami penurunan yang signifikan. Sebagai ilustrasi, dalam sistem sebelumnya, proses-proses tersebut memerlukan waktu antara 1 hingga 5 hari kerja, tergantung pada tingkat kompleksitas dan jumlah pihak yang terlibat. Setelah implementasi sistem Distributor Management System (DMS), proses yang sama dapat diselesaikan dalam rentang waktu 3 hingga 8 jam. Perbandingan durasi ini menunjukkan efisiensi waktu kerja administratif sebesar 73,5%, yang dihitung berdasarkan selisih rata-rata antara waktu proses manual dan waktu proses digital pada sistem baru. Pengukuran dilakukan dengan mencatat durasi aktual selama simulasi alur kerja yang dilaksanakan bersama tim pengembang dan pihak internal pengguna sistem pada tahap uji coba fungsional.

Pengukuran efisiensi tersebut diperkuat melalui simulasi terhadap beberapa proses bisnis inti yang sering menimbulkan keterlambatan dalam sistem sebelumnya. Misalnya, pada proses registrasi distributor, sebelumnya alur validasi dan penginputan data dilakukan secara manual dan terpisah, sehingga memerlukan waktu hingga tiga hari kerja. Setelah sistem DMS diterapkan, registrasi dapat diselesaikan dalam waktu kurang dari delapan jam berkat adanya integrasi form digital dan alur verifikasi otomatis, menghasilkan peningkatan efisiensi waktu sebesar 66,7%.

Proses lainnya yang turut diuji adalah pembuatan dan persetujuan Purchase Order (PO). Dalam sistem manual, dokumen PO dibuat terpisah, dikirim via email, dan membutuhkan validasi manual oleh bagian terkait, sehingga memakan waktu sekitar dua hari atau 16 jam kerja. Melalui sistem DMS, alur ini dapat diselesaikan

hanya dalam waktu empat jam karena seluruh proses dibuat terintegrasi dan berbasis workflow, mulai dari input hingga approval, dengan notifikasi internal yang mempercepat respons pengguna. Efisiensi proses ini tercatat sebesar 75%.

Selanjutnya, pengujian dilakukan pada proses penyusunan laporan Delivery Order (DO) yang sebelumnya membutuhkan waktu sekitar satu hari kerja karena harus merekap pesanan dan pengiriman secara manual menggunakan template Excel. Sistem baru memungkinkan laporan ini disusun secara otomatis dalam waktu tiga jam menggunakan JasperReport yang langsung menarik data dari modul terkait. Hal ini menghasilkan efisiensi waktu sebesar ±62,5%.

Proses yang paling signifikan dalam hal penghematan waktu adalah pembuatan faktur penagihan (invoice). Dalam sistem sebelumnya, proses ini sangat kompleks karena memerlukan validasi data dari berbagai sumber Delivery Order, Sales Order, diskon, dan pajak yang harus dicek secara manual. Total waktu yang dibutuhkan mencapai lima hari kerja atau 40 jam. Setelah implementasi sistem DMS, proses ini dapat dilakukan dalam waktu sekitar empat jam, menghasilkan efisiensi hingga 90%.

Temuan ini menunjukkan bahwa sistem DMS tidak hanya berdampak secara fungsional, tetapi juga secara kuantitatif dalam peningkatan produktivitas dan efisiensi kerja. Implementasi sistem ini terbukti mampu mengurangi waktu tunggu, mempercepat proses validasi data, dan mengurangi ketergantungan pada proses manual yang rawan kesalahan. Dengan demikian, sistem memberikan kontribusi nyata dalam optimalisasi operasi distribusi, serta meningkatkan responsivitas dan kualitas layanan terhadap mitra bisnis perusahaan secara menyeluruh.

3.3 Kendala yang Ditemukan

Selama pelaksanaan magang sebagai *Fullstack Developer* dalam pengembangan sistem Distributor Management System (DMS), ditemukan berbagai kendala teknis yang berdampak langsung terhadap proses implementasi sistem. Ken-

dala-kendala ini umumnya berkaitan dengan adaptasi terhadap teknologi baru, pemahaman arsitektur sistem, serta proses integrasi antarmuka antar modul. Adapun kendala utama yang dihadapi meliputi:

1. Ketidakterbiasaan dengan konsep-konsep seperti *dependency injection*, REST API, *observable*, dan *reactive form* menyebabkan hambatan saat mulai mengembangkan fitur awal.
2. Pada saat mengembangkan fitur Monitoring Incoterm, terjadi ketidaksesuaian antara data frontend dan backend akibat salah konfigurasi endpoint dan format data yang tidak konsisten.
3. Fitur login, registrasi, serta pengaturan hak akses pengguna memerlukan logika validasi yang kompleks dan harus konsisten di seluruh modul.
4. Kesulitan mencocokkan struktur data hasil input dengan format yang dibutuhkan untuk laporan menyebabkan keterlambatan dalam pengembangan fitur laporan.
5. Terjadi ketidaksesuaian ketika perubahan struktur data dilakukan pada satu modul, namun belum disesuaikan di modul lain, yang menyebabkan error saat integrasi.

3.4 Solusi atas Kendala yang Ditemukan

Untuk mengatasi kendala-kendala yang muncul selama masa magang, dilakukan sejumlah pendekatan dan strategi secara bertahap. Pendekatan ini terdiri dari pembelajaran mandiri, diskusi dengan mentor, dan pengembangan fitur secara iteratif. Berikut adalah solusi yang diterapkan untuk setiap kendala:

1. Mendalami dokumentasi resmi Spring Boot dan Angular, serta mengikuti tutorial dan studi kasus yang relevan. Fokus diarahkan pada struktur proyek, komunikasi HTTP, serta implementasi reactive form, sehingga mempercepat proses debugging dan pengembangan fitur.
2. Menggunakan Postman untuk pengujian API dan melakukan pengecekan manual terhadap *data binding* antara frontend dan backend. Dilakukan

juga *code review* berkala bersama tim untuk memastikan kesesuaian format data dan endpoint.

3. Implementasi autentikasi dilakukan secara bertahap, dimulai dari fungsi login dasar hingga validasi role. Setiap tahap diuji menggunakan akun dummy untuk memastikan konsistensi logika bisnis.
4. Memulai dengan pembuatan laporan sederhana sebelum melanjutkan ke format yang lebih kompleks. Mengandalkan dokumentasi Jasper dan bimbingan mentor teknis untuk memahami struktur XML dan kompatibilitas format.
5. Menjalankan komunikasi aktif dengan tim melalui Microsoft Teams serta melibatkan tim dalam *stand-up meeting* harian untuk memastikan setiap perubahan pada struktur data segera dikomunikasikan dan diimplementasikan ke seluruh modul yang terkait.