BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama periode kerja magang, penulis ditempatkan pada posisi *Web Development Intern* di dalam divisi *Business Controlling / Business Intelligence*. Dalam divisi ini, penulis merupakan satu-satunya personel yang berfokus pada aspek teknis pengembangan web. Oleh karena itu, tanggung jawab yang diberikan bersifat *full-stack* dan mencakup tiga area utama: pertama, pengembangan fitur baru untuk menambah fungsionalitas sistem; kedua, pemeliharaan berkelanjutan untuk memperbaiki *bug* dan menjaga akurasi data; serta ketiga, optimalisasi untuk meningkatkan performa dan kualitas antarmuka. Seluruh pekerjaan ini dilakukan pada sistem *dashboard* internal yang dibangun menggunakan *framework* Laravel dan Tailwind CSS.

Dalam menjalankan peran tersebut, penulis berfungsi sebagai penghubung teknis yang memastikan alur data berjalan lancar dari sumber hingga ke pengguna akhir. Alur kerja dan koordinasi penulis bersifat dua arah dan melibatkan interaksi erat dengan beberapa tim. Dari sisi *input*, proses kerja dimulai dengan koordinasi bersama tim *Data Mining* untuk menerima kueri *SQL* yang menjadi dasar pengambilan data. Penulis kemudian bertanggung jawab untuk melakukan adaptasi dan integrasi kueri tersebut ke dalam logika *back-end* aplikasi, memastikan kueri berjalan secara efisien di dalam *framework* Laravel.

Setelah data berhasil diproses di *back-end*, hasilnya disajikan melalui antarmuka *front-end* kepada tim Analis BI. Tim Analis BI, sebagai pengguna akhir (*end-user*), memegang peran krusial dalam tahap validasi, di mana mereka memastikan bahwa data yang ditampilkan sudah akurat dan sesuai dengan kebutuhan bisnis. Selain itu, dari sisi *feedback*, penulis secara aktif menerima laporan masalah, *bug*, atau permintaan perbaikan langsung dari tim Analis BI, yang menjadi dasar untuk siklus pemeliharaan selanjutnya. Seluruh alur kerja ini berada di bawah pengawasan dan arahan strategis dari seorang *supervisor* yang memastikan setiap tugas sejalan dengan prioritas divisi.

3.2 Tugas yang Dilakukan

Selama melaksanakan magang di PT Sumber Trijaya Lestari (Aksesmu), penulis mengemban tanggung jawab utama dalam pengembangan dan pemeliharaan sistem *dashboard* internal berbasis web yang dilakukan guna mendukung kebutuhan analisis data pada divisi *Business Intelligence*. Seluruh lingkup tugas tersebut diklasifikasikan ke dalam empat kategori utama yang saling terkait dan disajikan secara sistematis sebagai berikut:

1. **Pengembangan Fitur (*Feature Development*)**
   Tugas ini mencakup pengembangan *dashboard* dari analisis kebutuhan hingga implementasi kode, meliputi perancangan solusi teknis di sisi *back-end* (Laravel) dan *front-end*. Pengembangan fitur mengikuti prinsip *Feature-Driven Development*, dengan setiap fitur dikembangkan dalam siklus iteratif yang jelas [4].

2. **Pengembangan *Dashboard* dan Visualisasi Data (*Dashboard Development and Data Visualization*)**
   Penulis bertanggung jawab merancang dan membangun halaman *dashboard* baru untuk memvisualisasikan data strategis perusahaan. Tugas ini melibatkan penyusunan kueri (*query*) basis data yang kompleks untuk agregasi data, serta merancang antarmuka visual yang informatif.

3. **Pemeliharaan Korektif (*Corrective Maintenance*)**
   Fokus utama dari tugas ini adalah menangani laporan *bug* atau ketidakakuratan data yang ditemukan oleh tim Analis. Penulis melakukan perbaikan melalui proses *debugging* yang mendalam, mulai dari penelusuran alur *route* dan *controller*, hingga menganalisis kueri SQL pada basis data untuk mengidentifikasi dan memperbaiki akar permasalahan.

4. **Optimalisasi & Pemeliharaan Perfektif (*Perfective Maintenance*)**
   Tugas proaktif untuk meningkatkan kualitas sistem *dashboard* melalui dua area:
   a. Optimalisasi kinerja menggunakan *tools* diagnostik (Clockwork, Chrome DevTools) untuk mempercepat waktu muat halaman.
   b. Optimalisasi *User Interface & User Experience* melalui *refactoring* kode *front-end* (Tailwind CSS) untuk meningkatkan konsistensi visual dan kemudahan pemeliharaan kode.

3.3 Uraian Pelaksanaan Magang

Bagian ini memaparkan beberapa studi kasus representatif dari tugas-tugas yang telah diselesaikan selama magang. Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

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

Minggu Ke -	**Pekerjaan yang dilakukan**
1	Memahami proses bisnis dan alur kerja, melakukan *setup tools* (Jenkins, GitLab, VS Code, DBeaver), serta memperbaiki *bug* fitur *export to* Excel dan memperbarui antarmuka *Dashboard Admin* agar lebih responsif.
2	Melakukan optimasi kueri SQL untuk tabel performa (Net Sales, Margin), memperbaiki responsivitas *grid* menu, dan melakukan *refactoring* kode *styling* agar lebih rapi dan aman.
3	Menginstalasi *package* Clockwork untuk *debugging* dan *profiling*, memisahkan konfigurasi *environment* (*staging*/*production*), serta melakukan migrasi komponen UI dari Bootstrap ke Tailwind CSS dan Alpine.js.
4	Menyesuaikan konsistensi *styling* pada berbagai *Dashboard* (Laba Rugi, IKDC, DSI), menangani *build error* akibat dependensi Composer di Jenkins, dan mengimplementasikan notifikasi SweetAlert.
5	Mengembangkan fitur filter *export* Excel dengan *multiple choices* untuk kode PLU[1], menambahkan validasi data input, dan mengamankan konfigurasi *endpoint API* di *.env*.
6	Mengimplementasikan *Unit Testing* otomatis, memperbaiki logika *middleware* (*CheckRole*), dan mengembangkan fitur *Role Management* pada *Dashboard Superadmin*.
7	Melakukan optimasi kueri validasi PLU untuk mencegah *timeout*, melakukan *refactor* struktur *routes* Laravel agar lebih terorganisir, dan memulai pengembangan fitur *Dashboard Grading Member*.

[1] *Price Label Unit* adalah kode unik yang dibuat oleh perusahaan yang berfungsi sebagai identitas setiap barang dagangan yang dijual di *Stock Point*.

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

Minggu Ke -	Pekerjaan yang dilakukan
8	Merombak sistem RBAC (*Role-Based Access Control*) dengan struktur *routes* berbasis *array*, serta mengembangkan fitur detail member pada *Dashboard Grading Member* menggunakan Alpine.js dan Livewire.
9	Menyesuaikan tampilan UI agar konsisten pada *Dashboard* IKDC dan Report Lapse, serta melanjutkan implementasi kueri basis data untuk *Dashboard Grading Member*.
10	Melakukan penyesuaian nama tabel pada *controller* agar sesuai dengan perubahan skema basis data dan memulai pengembangan modul baru *Dashboard Sales BI*.
11	Melanjutkan pengembangan fitur utama *Dashboard Sales BI* dan menyesuaikan logika tampilan serta kueri untuk daftar item terlaris pada *Dashboard Grading Member*.
12	Menambahkan animasi *loading state* pada komponen Livewire, filter periode bulanan, dan visualisasi *Vertical Bar Chart* untuk data member pada *Dashboard Grading Member*.
13	Mengembangkan tampilan daftar seluruh member per level, menggabungkan fitur-fitur yang telah selesai (*merge branch*) ke *master branch*, dan memastikan responsivitas tampilan di berbagai perangkat.
14	Melakukan penyesuaian tata letak menu utama *Dashboard BI* dan melakukan konsultasi intensif dengan *user* untuk evaluasi akhir pengembangan *Dashboard Grading Member*.

3.3.1 Catatan tentang Tools dan Metodologi

Dalam pelaksanaan tugas-tugas teknis yang telah diuraikan sebelumnya, penulis memanfaatkan berbagai perangkat lunak dan metodologi pengembangan untuk mendukung efisiensi kerja. Berikut adalah rincian teknologi utama yang digunakan:

1. **Laravel dan Livewire**

   Laravel adalah *framework* PHP yang menyediakan komponen untuk menghasilkan kode *frontend* [5]. Dalam proyek ini, Laravel diintegrasikan

dengan Livewire, yaitu teknologi yang memungkinkan pengembangan antarmuka pengguna berbasis PHP secara dinamis tanpa memerlukan pengetahuan *JavaScript* yang mendalam [6]. Kombinasi ini mempercepat proses pengembangan fitur interaktif pada *dashboard*.

2. **Tailwind CSS**

   Tailwind CSS merupakan *framework* CSS dengan pendekatan *utility-first* yang populer untuk membangun antarmuka pengguna (UI) kustom secara cepat. Berbeda dengan *UI kit* tradisional seperti Bootstrap, Tailwind tidak menyediakan komponen dengan gaya bawaan, melainkan memungkinkan pengembangan situs web modern dengan menulis CSS langsung di dalam *markup* serta menawarkan fleksibilitas kustomisasi yang tinggi [7].

3. **Jenkins**

   Jenkins adalah sistem otomasi *open-source* yang diadopsi secara luas untuk mengimplementasikan jalur kerja (*pipeline*) CI/CD dikarenakan fleksibilitas dan kemampuan ekstensinya [8]. Perangkat ini digunakan oleh penulis untuk mengotomatisasi proses *build*, pengujian (*testing*), dan penyebaran (*deployment*), guna memastikan setiap perubahan kode terintegrasi dengan baik dan siap dirilis ke lingkungan produksi.

4. **Unit Testing**

   Unit Testing adalah metode verifikasi perangkat lunak di mana setiap unit kode diuji secara terisolasi untuk memastikan kebenaran logika pemrogramannya [9]. Penerapan praktik ini menjadi sangat krusial dalam menjaga stabilitas aplikasi, terutama di tengah siklus pengembangan yang cepat dan berulang.

5. **Role-Based Access Control (RBAC)**

   RBAC adalah model keamanan yang mengatur hak akses pengguna berdasarkan peran spesifik mereka di dalam sistem [10]. Implementasi RBAC pada sistem *dashboard* ini bertujuan untuk menjamin keamanan data dengan memastikan setiap pengguna hanya dapat mengakses fitur dan informasi yang relevan dengan tanggung jawab jabatannya.

6. **Agile Methodology**

   Agile adalah metodologi pengembangan perangkat lunak yang bersifat *incremental* (rilis cepat dengan siklus singkat), *cooperative* (kolaborasi berkelanjutan dengan pengguna), *straightforward* (mudah dipelajari dan

dimodifikasi), dan *adaptive* (responsif terhadap perubahan kebutuhan) [4]. Metodologi ini dipilih karena memungkinkan *feedback* berkelanjutan dari pengguna, memastikan hasil akhir sesuai dengan kebutuhan bisnis yang dinamis.

7. **Black Box Testing**

   Black box *testing* adalah metodologi pengujian yang memvalidasi fungsionalitas sistem berdasarkan spesifikasi kebutuhan, tanpa menganalisis struktur internal atau kode sumber [11]. Penguji berinteraksi dengan antarmuka pengguna menggunakan teknik seperti partisi ekuivalensi dan analisis nilai batas untuk memverifikasi output sesuai ekspektasi. Metode ini memastikan setiap fitur *dashboard* berfungsi dengan benar dari perspektif pengguna akhir.

8. ***User-Centered Design* (UCD)**

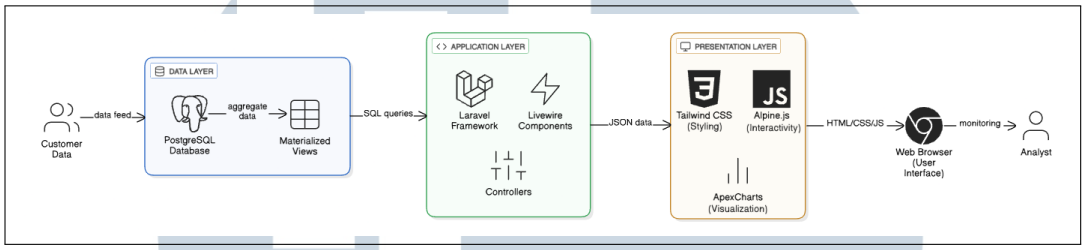
   User-Centered Design adalah pendekatan desain yang menempatkan pengguna sebagai pusat dari seluruh proses pengembangan sistem [12]. UCD terdiri dari empat fase utama: (1) *Specify the context of use*, yaitu mengidentifikasi siapa pengguna akhir dan bagaimana mereka akan menggunakan sistem; (2) *Specify requirements*, yaitu menentukan kebutuhan bisnis dan pengguna; (3) *Create design solutions*, yaitu membangun desain dari konsep kasar hingga desain lengkap secara iteratif; dan (4) *Evaluate designs*, yaitu melakukan evaluasi kegunaan idealnya melalui pengujian dengan pengguna nyata. Pendekatan ini dipilih dalam pengembangan *Dashboard Grading Member* karena memastikan bahwa solusi yang dibangun benar-benar sesuai dengan kebutuhan dan ekspektasi pengguna akhir, yaitu tim analis *Business Intelligence*. Berbeda dengan pendekatan *Genius Design* yang mengandalkan intuisi perancang atau *Activity-Centered Design* yang berfokus pada aktivitas, UCD lebih tepat diterapkan karena keberhasilan *dashboard* sangat bergantung pada kemudahan penggunaan dan relevansi informasi yang disajikan bagi pengguna.

3.3.2 Pengembangan *Dashboard* Grading Member

Sistem manajemen loyalitas member memerlukan alat visualisasi yang mampu memberikan wawasan mendalam terhadap distribusi dan perkembangan member berdasarkan tingkat loyalitas. Pada bagian ini, penulis akan memaparkan proses pengembangan *dashboard* untuk sistem *grading* member yang berbasis XP.

A Arsitektur Sistem

Sistem *Dashboard Grading Member* dirancang dengan arsitektur berlapis (*layered architecture*) yang memisahkan tanggung jawab setiap komponen untuk meningkatkan *maintainability* dan skalabilitas sistem.



Gambar 3.1. Arsitektur Lengkap dan Alur Data Dashboard Grading Member

Sebagaimana ditunjukkan pada Gambar 3.1, arsitektur sistem terdiri dari empat lapisan yang saling terhubung:

1. ***Data Layer***
   Lapisan ini bertanggung jawab atas penyimpanan dan pra-pemrosesan data. PostgreSQL *Database* menyimpan data transaksional member yang kemudian diagregasi menjadi *Materialized Views*. Penggunaan *Materialized View* mengurangi waktu eksekusi kueri kompleks.

2. ***Application Layer***
   Lapisan ini menangani logika bisnis dan komunikasi antara *database* dengan antarmuka pengguna. Laravel *Framework* bertindak sebagai *backbone* aplikasi, dengan `GradingDashboardController` sebagai *entry point* yang menerima *HTTP request*. Livewire *Components* (`MemberList.php`, `ItemList.php`) mengelola *state* aplikasi dan melakukan kueri ke *database* melalui *Query Builder*. *Controllers* bertugas mengambil data agregat untuk grafik dan statistik *dashboard*.

3. ***Presentation Layer***
   Lapisan ini bertanggung jawab merender antarmuka pengguna yang responsif dan interaktif. Tailwind CSS menyediakan *utility-first styling* untuk desain yang konsisten, Alpine.js menangani interaktivitas di sisi klien seperti *modal*, *dropdown*, dan filter tanpa *page reload*, sedangkan ApexCharts menghasilkan visualisasi grafik interaktif berdasarkan data JSON yang diterima dari *controller*.

4. ***Client Layer***

   Lapisan ini adalah *web browser* pengguna yang menampilkan halaman *dashboard* dan mengirimkan *event* pengguna (seperti klik, pencarian, filter) kembali ke server melalui direktif Livewire (`wire:model`, `wire:click`). Analis BI menggunakan antarmuka ini untuk melakukan monitoring dan analisis data member.

A.1 Alur Data

Alur data dalam sistem dimulai ketika data transaksi pelanggan masuk ke PostgreSQL *Database*. Data ini kemudian diagregasi menjadi *Materialized Views* untuk optimalisasi performa. Saat pengguna mengakses *dashboard*, *Application Layer* mengeksekusi kueri SQL ke *Materialized Views* dan mengonversi hasilnya menjadi format JSON. Data JSON ini kemudian dikirim ke *Presentation Layer* yang merender grafik menggunakan ApexCharts dan tabel data menggunakan Livewire. Seluruh proses ini terjadi secara reaktif, setiap perubahan filter atau pencarian oleh pengguna langsung memicu *re-query* dan *re-render* tanpa perlu *refresh* halaman penuh.

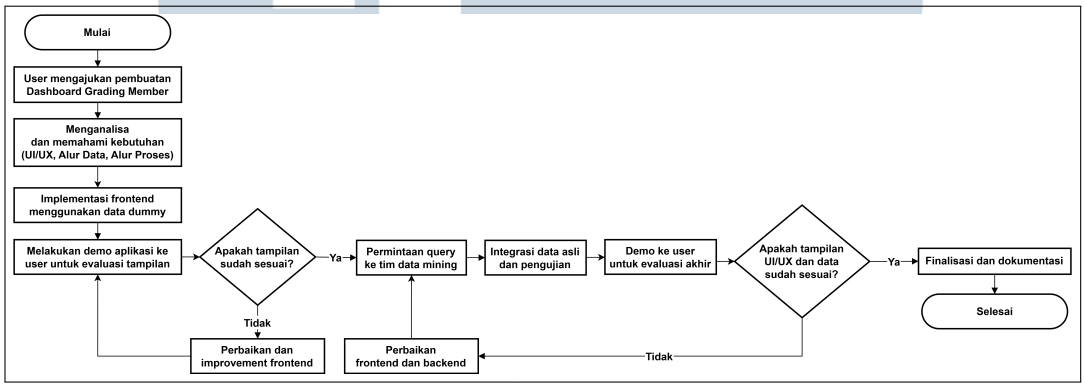
B Analisis dan Perancangan Solusi

Pengembangan *dashboard* ini menggunakan pendekatan *User-Centered Design* (UCD) yang dikombinasikan dengan metodologi Agile. Pendekatan UCD dipilih karena keberhasilan *dashboard* sangat bergantung pada kesesuaian antara sistem yang dibangun dengan kebutuhan pengguna akhir, yaitu tim analis *Business Intelligence*. Prinsip-prinsip UCD yang diterapkan meliputi:

1. *Specify the context of use*: Mengidentifikasi bahwa pengguna akhir adalah tim analis BI yang membutuhkan visualisasi data member untuk keperluan pelaporan.
2. *Specify requirements*: Menentukan kebutuhan fungsional melalui sesi diskusi dengan pengguna, seperti grafik tren, pencarian member, dan detail transaksi.
3. *Create design solutions*: Membangun tampilan *front-end* dengan data *dummy* terlebih dahulu untuk mendapatkan persetujuan visual sebelum integrasi data asli.

4. *Evaluate designs*: Melakukan *demo* berkala kepada pengguna dan mengiterasi desain berdasarkan *feedback* yang diterima.

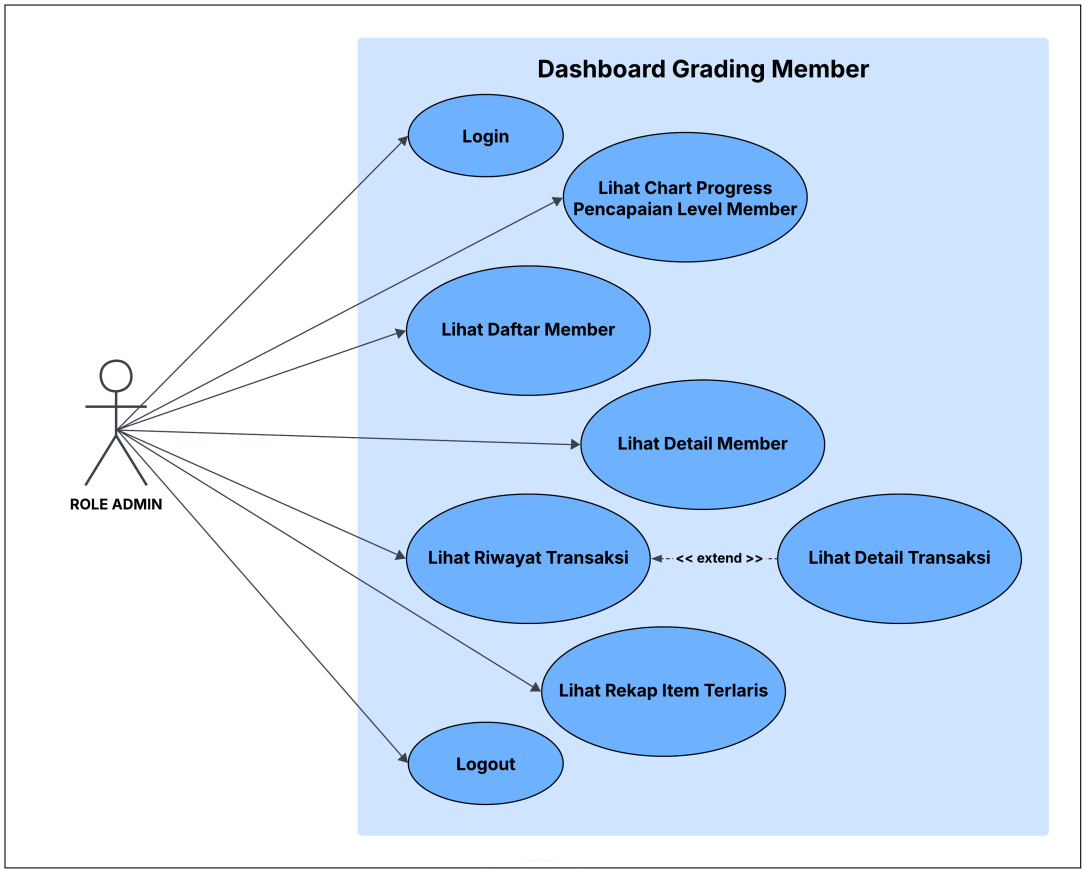
Alur pengembangan ditunjukkan pada Gambar 3.2. Proses dimulai dengan analisis kebutuhan bersama pengguna, dilanjutkan dengan implementasi *front-end* dengan data *dummy*, kemudian *demo* ke *user* untuk evaluasi. Jika tampilan belum sesuai, dilakukan perbaikan dan iterasi kembali sesuai prinsip UCD. Setelah tampilan disetujui, penulis meminta *query* data asli ke tim *data mining* untuk melakukan integrasi dan pengujian, kemudian *demo* akhir ke *user*. Proses berulang hingga UI/UX dan data sudah sesuai, baru dilakukan finalisasi dan dokumentasi.



Gambar 3.2. *Flowchart* Proses Pengembangan *Dashboard Grading Member*

Kebutuhan fungsional diidentifikasi dan dimodelkan dalam *Use Case Diagram* pada Gambar 3.3. Aktor utama dalam sistem adalah Admin yang dapat melakukan beberapa aksi berikut:

1. *Login* ke sistem untuk mengakses *dashboard*.
2. Melihat *chart progress* pencapaian level member selama enam bulan terakhir.
3. Melihat daftar member dengan fitur *filter*.
4. Melihat detail member yang mencakup informasi profil dan juga riwayat transaksi selama tiga bulan terakhir.
5. Melihat detail transaksi yang menampilkan rincian barang.
6. *Logout* dari sistem.



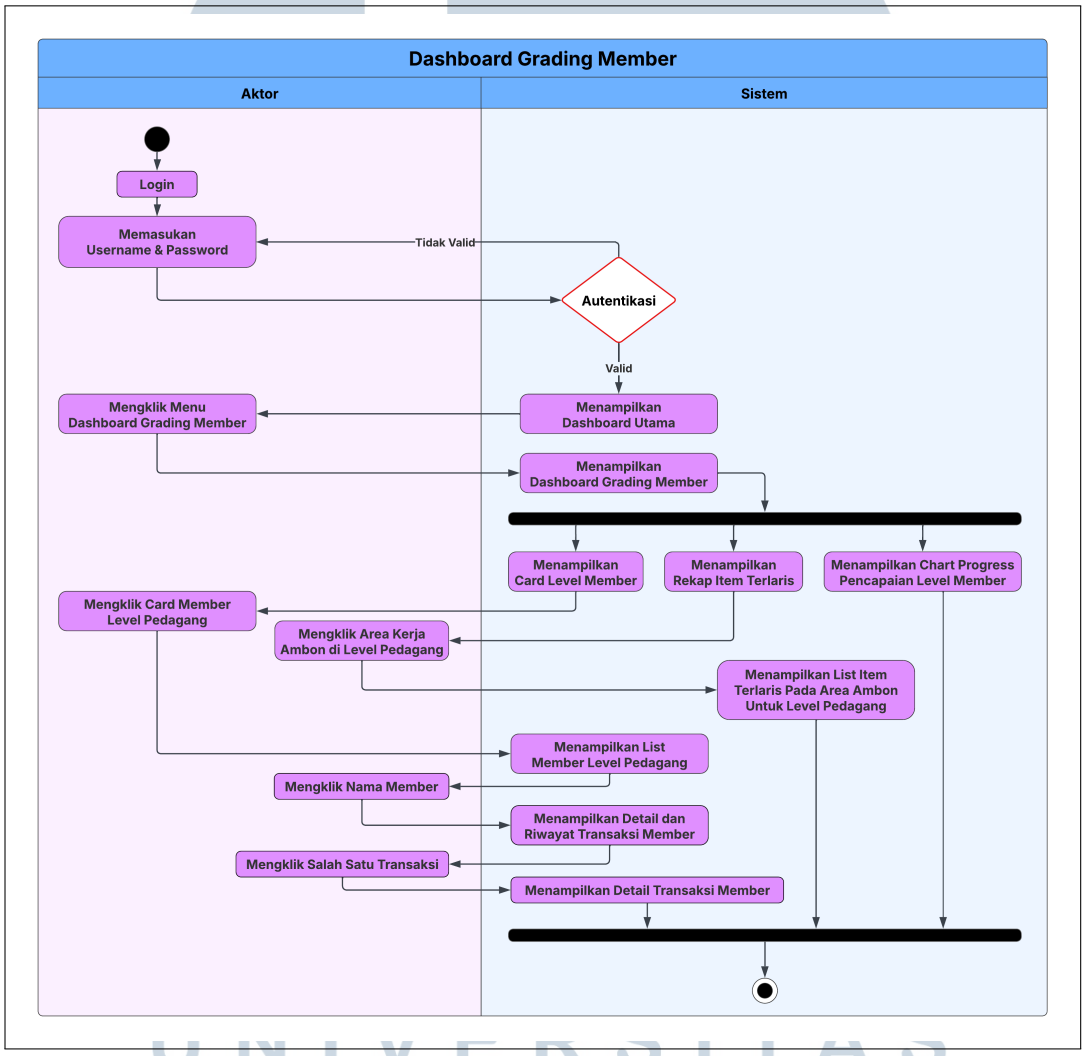
Gambar 3.3. *Use Case Diagram Dashboard* Grading Member

Diagram menunjukkan tujuh *use case* utama yang tersedia pada *dashboard*. Setiap *use case* mewakili satu fungsionalitas yang dapat diakses oleh Admin untuk mengelola dan menganalisis data member. Hubungan *extend* antara *use case* Lihat Riwayat Transaksi dan Lihat Detail Transaksi menunjukkan bahwa detail transaksi merupakan fungsionalitas lanjutan yang dipicu ketika Admin memilih transaksi tertentu dari daftar riwayat. Semua *use case* diakses melalui sistem setelah Admin berhasil melakukan *login*, dan sesi berakhir ketika Admin melakukan *logout*

Alur interaksi pengguna dengan sistem dimodelkan dalam *Activity Diagram* pada Gambar 3.4. Diagram ini mengilustrasikan perjalanan pengguna mulai dari proses autentikasi, akses menu utama *dashboard*, hingga interaksi dengan berbagai fitur sistem.

Proses dimulai dengan autentikasi pengguna melalui mekanisme validasi *username* dan *password*. Setelah validasi berhasil, pengguna dapat mengakses halaman utama *dashboard* yang menampilkan ringkasan data dan opsi menu. Dari halaman utama, pengguna dapat melakukan beberapa aktivitas paralel, termasuk

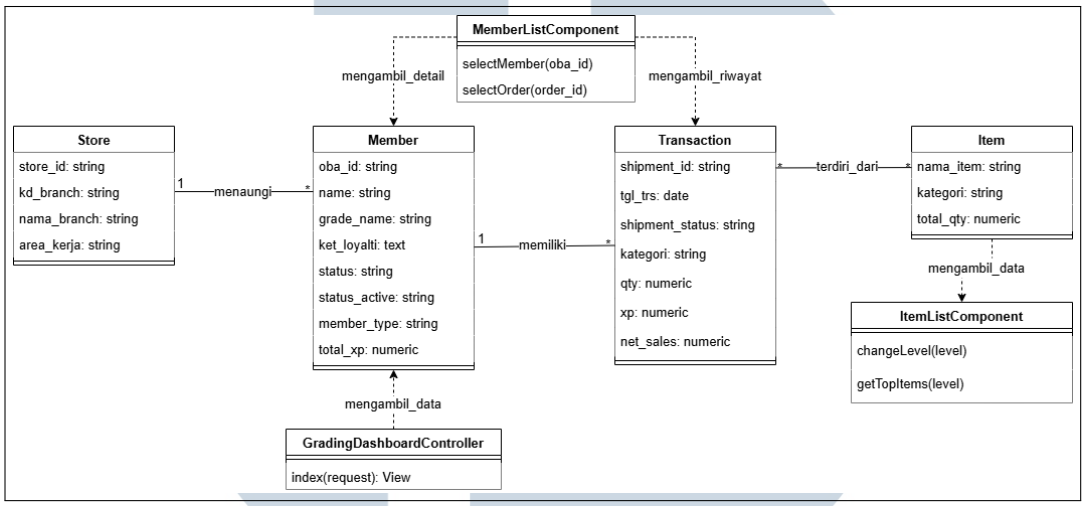
melihat kartu ringkasan level member, mengakses rekap item terlaris berdasarkan area kerja, atau melihat grafik *progress* pencapaian level member selama enam bulan terakhir. Pengguna juga dapat memilih nama member untuk melihat detail dan riwayat transaksi, dengan opsi untuk melakukan *filter* berdasarkan level, melakukan pencarian spesifik, atau melihat detail transaksi individual. Alur berakhir ketika pengguna selesai mengakses informasi dan keluar dari sistem.



Gambar 3.4. *Activity Diagram Dashboard* Grading Member

Gambar 3.5 mengilustrasikan struktur statis sistem yang dibangun menggunakan arsitektur *Model-View-Controller* (MVC). Arsitektur ini dipadukan dengan komponen Livewire untuk mendukung interaksi data secara dinamis tanpa memuat ulang halaman secara keseluruhan. Proses pengambilan dan pengolahan data dipusatkan pada *controller* dan komponen Livewire, yang

kemudian berinteraksi dengan *domain class* (Store, Member, Transaction, dan Item) untuk menyajikan informasi yang relevan pada *Dashboard Grading Member*. Relasi asosiasi antar kelas menggambarkan keterkaitan entitas data yang menjamin integritas referensial dalam operasi sistem.



Gambar 3.5. *Class Diagram Dashboard* Grading Member

Berdasarkan diagram di atas, berikut adalah deskripsi kelas-kelas utama yang terlibat dalam sistem:

1. **Store**

   Merepresentasikan entitas toko atau cabang dengan atribut utama meliputi `store_id`, `kd_branch`, `nama_branch`, dan `area_kerja`. Kelas ini memiliki relasi asosiasi *one-to-many* dengan kelas Member, yang mengindikasikan bahwa satu cabang dapat menaungi banyak anggota (*member*).

2. **Member**

   Merepresentasikan data pelanggan yang terdaftar dengan atribut seperti `oba_id`, `name`, `grade_name`, `ket_loyalti`, `status`, `status_active`, `member_type`, dan `total_xp`. Kelas ini memiliki relasi *one-to-many* dengan Transaction, yang berarti setiap anggota dapat memiliki riwayat transaksi yang beragam.

3. **Transaction**

   Menyimpan data riwayat transaksi anggota dengan atribut `shipment_id`, `tgl_trs`, `shipment_status`, `kategori`, `qty`, `xp`, `net_sales`, dan `gm`. Kelas ini memiliki relasi *many-to-many* dengan kelas Item (biasanya dijembatani

oleh tabel detail transaksi), yang menunjukkan bahwa satu transaksi dapat terdiri dari banyak *item*, dan satu jenis *item* dapat muncul dalam banyak transaksi berbeda.

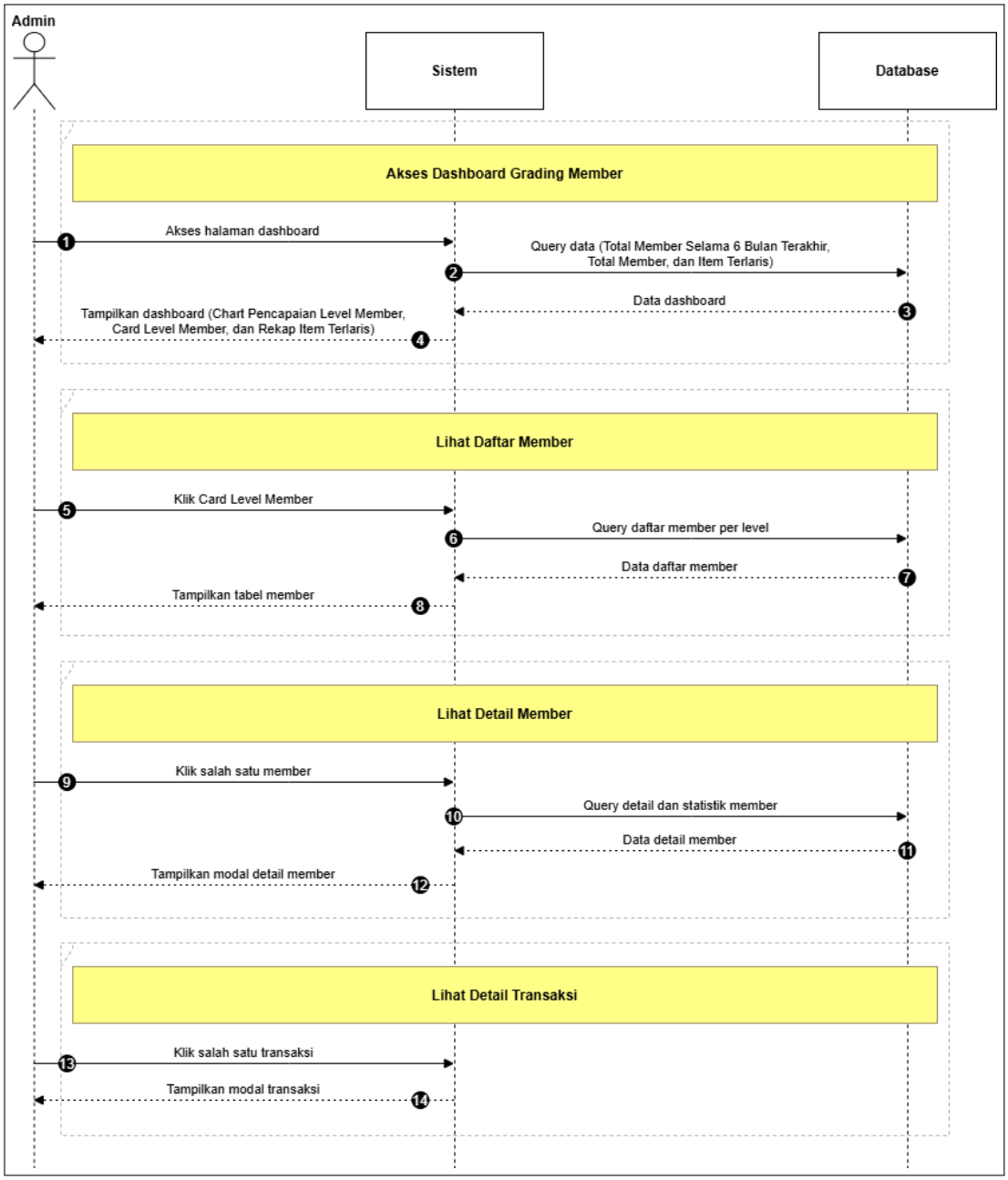
4. **Item**
   Merepresentasikan katalog produk dengan atribut `nama_item` dan `kategori`. Kelas ini esensial untuk mendukung fitur analisis, seperti rekapitulasi produk terlaris berdasarkan level anggota maupun area kerja.

5. **GradingDashboardController**
   Berperan sebagai *controller* utama yang menangani logika bisnis untuk halaman depan *Dashboard Grading Member*. Kelas ini bertanggung jawab melakukan agregasi data, seperti perhitungan untuk *chart* progres anggota dan ringkasan populasi anggota per level, sebelum data tersebut dikirimkan ke *view*.

6. **MemberListComponent**
   Merupakan komponen Livewire yang berfungsi sebagai pengendali untuk interaksi data daftar anggota secara dinamis. Komponen ini mengelola logika pencarian, penyaringan berdasarkan tipe loyalitas, serta paginasi data. Method `selectMember(oba_id)` digunakan untuk mengambil detail profil dan riwayat transaksi anggota, sedangkan `selectOrder(order_id)` menampilkan rincian transaksi yang dipilih.

7. **ItemListComponent**
   Merupakan komponen Livewire yang mengelola tampilan rekap item terlaris per level anggota. Komponen ini menyediakan method `changeLevel(level)` untuk mengubah level yang ditampilkan dan `getTopItems(level)` untuk mengambil data produk terlaris berdasarkan level dan area kerja tertentu.

Interaksi antar komponen saat pengguna menggunakan *Dashboard* Grading Member dimodelkan dalam *Sequence Diagram* pada Gambar 3.6. Diagram ini menggambarkan alur komunikasi antara Pengguna, Sistem, dan *database* untuk empat skenario utama.



Gambar 3.6. *Sequence Diagram Dashboard* Grading Member

Berdasarkan diagram tersebut, terdapat empat alur interaksi utama:

1. **Akses *Dashboard***
   Pengguna membuka halaman *dashboard*, kemudian sistem mengambil data grafik dan total member dari *database*. Sistem memproses data untuk ditampilkan dalam bentuk grafik *progress* enam bulan terakhir dan ringkasan jumlah member per level *grading*.

2. **Lihat Daftar Member**

   Pengguna menerapkan *filter* atau melakukan pencarian pada tabel member. Sistem memproses permintaan tersebut dengan melakukan *query* ke *database* untuk mengambil daftar member yang sesuai kriteria, kemudian menampilkan hasil dalam bentuk tabel dengan paginasi.

3. **Lihat Detail Member**

   Pengguna mengklik baris member pada tabel untuk melihat detail. Sistem memanggil fungsi `selectMember()` yang mengambil informasi lengkap member beserta riwayat transaksinya dari *database*. Data yang diperoleh meliputi profil member, statistik XP, dan daftar pesanan terakhir. Hasil ditampilkan dalam modal detail member.

4. **Lihat Detail Transaksi**

   Pengguna mengklik salah satu transaksi pada modal detail member. Pada alur ini, sistem tidak melakukan *query* ulang ke *database* karena data transaksi sudah dimuat saat proses sebelumnya. Sistem cukup menggunakan data yang sudah tersimpan di *state* komponen untuk menampilkan modal detail transaksi. Pendekatan ini mengoptimalkan performa dengan mengurangi jumlah *request* ke *database*.

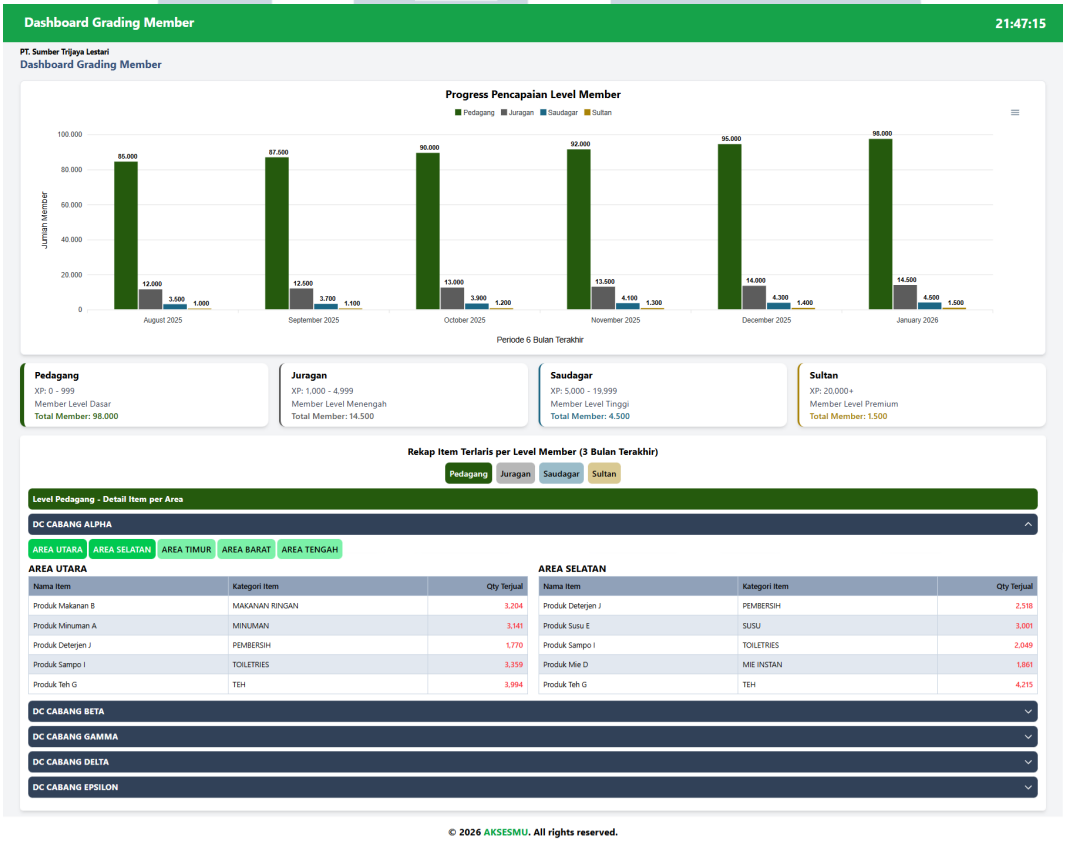
Poin penting dari desain ini adalah penggunaan *caching* pada level komponen Livewire. Ketika pengguna membuka detail member, semua data termasuk riwayat transaksi sudah diambil sekaligus. Sehingga ketika pengguna ingin melihat detail transaksi tertentu, tidak diperlukan *query* tambahan ke *database*. Hal ini membuat interaksi terasa lebih responsif bagi pengguna.

C Implementasi

Implementasi dilakukan secara iteratif sesuai dengan metodologi Agile. Proses dimulai dengan pengembangan *front-end* menggunakan data *dummy*, dilanjutkan dengan integrasi data asli setelah mendapat persetujuan tampilan dari pengguna. *Dashboard* Grading Member terdiri dari tiga tampilan utama yang saling terintegrasi, yaitu halaman utama *dashboard*, halaman detail member, dan halaman detail transaksi. Masing-masing tampilan dijelaskan pada bagian berikut.

C.1 Halaman Utama *Dashboard*

Halaman utama berfungsi sebagai pusat informasi yang menyajikan ringkasan data member secara visual dan komprehensif. Tampilan ini dirancang untuk memberikan gambaran menyeluruh mengenai kondisi keanggotaan perusahaan dalam satu pandangan. Elemen utama yang ditampilkan adalah *chart* bar yang memvisualisasikan jumlah member per level selama enam bulan terakhir, memungkinkan tim analis untuk mengidentifikasi tren pertumbuhan atau penurunan member pada setiap tingkatan loyalitas.



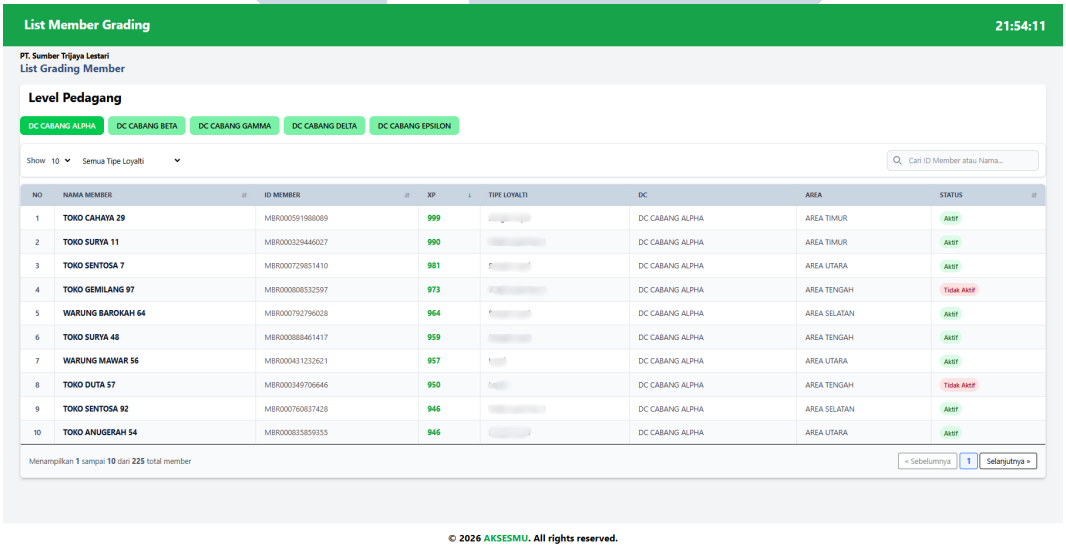
Gambar 3.7. Halaman Utama pada *Dashboard* Grading Member

Halaman ini juga menampilkan empat kartu ringkasan yang menunjukkan total member aktif per level dengan kode warna yang berbeda untuk memudahkan identifikasi visual. Setiap kartu bersifat interaktif dan dapat diklik untuk membuka halaman detail member pada level yang bersangkutan. Selain itu, terdapat tampilan rekap produk yang paling banyak dibeli oleh member pada setiap level, yang disajikan dalam bentuk tabel terstruktur. Data dikelompokkan berdasarkan cabang dan area kerja sehingga tim analis dapat melihat preferensi produk yang berbeda

di setiap wilayah, membantu dalam pengambilan keputusan strategis terkait alokasi stok dan promosi produk. Tampilan halaman utama dapat dilihat pada Gambar 3.7.

C.2 Halaman List Member

Halaman *list* member menampilkan daftar seluruh member berdasarkan level yang dipilih dari halaman utama. Halaman ini dirancang untuk mengakomodasi kebutuhan tim analis dalam melakukan eksplorasi data member secara mendalam. Tabel data dilengkapi dengan berbagai fitur interaktif, termasuk pencarian berdasarkan nama atau ID member, *filter* berdasarkan tipe loyalitas (seperti Pedagang Baru, Pedagang Lama, atau Pedagang Reaktivasi), pengurutan kolom secara dinamis, dan paginasi untuk memudahkan navigasi data dalam jumlah besar. Pengguna dapat memilih cabang melalui tombol *filter* yang tersedia di bagian atas halaman, sehingga analisis dapat difokuskan pada wilayah operasional tertentu. Tampilan halaman *list* member dapat dilihat pada Gambar 3.8.



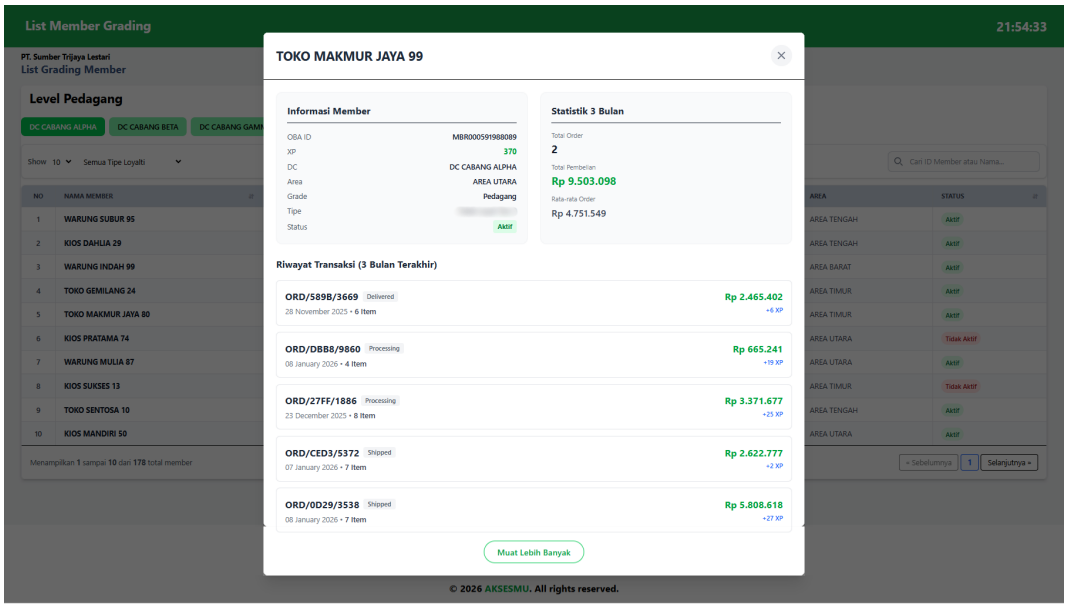
Gambar 3.8. Tampilan List Member pada *Dashboard* Grading Member

Setiap baris pada tabel menampilkan informasi penting seperti nama member, ID member, total XP yang dikumpulkan, tipe loyalitas, cabang terdaftar, area kerja, dan status keanggotaan (aktif atau tidak aktif). Desain tabel menggunakan kode warna untuk membedakan status keanggotaan, di mana member aktif ditandai dengan warna hijau dan member tidak aktif ditandai dengan warna merah. Kolom-kolom tersebut dapat diurutkan secara *ascending* maupun *descending* dengan mengklik *header* kolom, memungkinkan pengguna

untuk dengan cepat mengidentifikasi member dengan XP tertinggi atau terendah. Pengguna juga dapat mengklik baris member untuk membuka modal detail yang menampilkan informasi lebih lengkap mengenai member tersebut, termasuk riwayat transaksi dan statistik pembelian.

C.3 Halaman Detail Member

Halaman detail member dibangun menggunakan komponen Livewire untuk mendukung fitur interaktif yang memungkinkan pembaruan data secara real-time tanpa perlu memuat ulang halaman. Fitur-fitur yang tersedia meliputi pencarian cepat, *filter* berdasarkan tipe loyalitas, dan paginasi dengan opsi jumlah data per halaman yang dapat disesuaikan. Pengguna dapat memilih cabang melalui tombol *filter* dan melihat detail member melalui modal yang menampilkan informasi komprehensif. Modal tersebut terbagi menjadi tiga bagian utama: pertama, informasi profil member yang mencakup nama, ID, level, tipe loyalitas, dan tanggal bergabung; kedua, statistik transaksi tiga bulan terakhir yang meliputi total pembelian, rata-rata nilai transaksi, dan XP yang diperoleh; serta ketiga, riwayat pesanan yang menampilkan daftar transaksi terbaru.



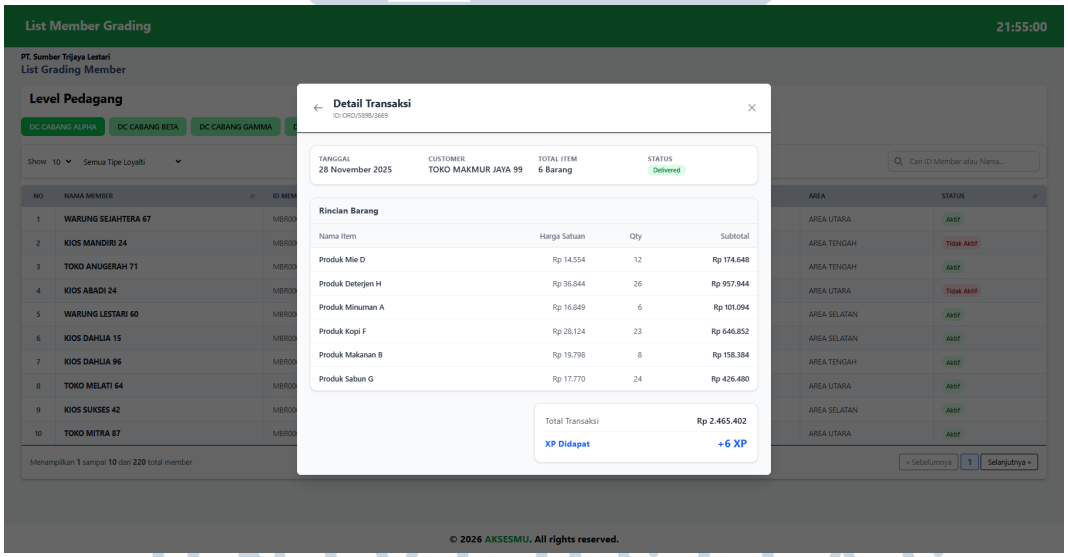
Gambar 3.9. Tampilan Detail Member pada *Dashboard* Grading Member

Modal detail member juga menampilkan daftar pesanan terakhir yang disajikan dalam format tabel ringkas, mencakup tanggal transaksi, jumlah item, dan nilai transaksi. Setiap baris pesanan dapat diklik untuk melihat rincian transaksi

lebih lanjut. Fitur "Muat Lebih Banyak" tersedia di bagian bawah daftar pesanan untuk mengambil riwayat transaksi tambahan tanpa menutup modal. Desain antarmuka dibuat responsif dengan menggunakan teknik *scrollable container* sehingga pengguna dapat dengan mudah menavigasi data dalam jumlah besar tanpa kehilangan konteks informasi profil di bagian atas. Tampilan halaman detail member dapat dilihat pada Gambar 3.9.

C.4 Halaman Detail Transaksi

Halaman detail transaksi menampilkan rincian lengkap dari setiap transaksi yang dilakukan oleh member, menyajikan informasi pada tingkat *item-level* untuk analisis yang lebih granular. Informasi yang ditampilkan mencakup identitas transaksi (*shipment ID*), tanggal dan waktu transaksi, status pengiriman (seperti Delivered, Pending, atau Cancelled), kategori barang yang dibeli, jumlah item per kategori, XP yang diperoleh dari transaksi tersebut, *net sales* (nilai penjualan bersih setelah diskon), dan *gross margin* (selisih antara harga jual dan harga pokok).



Gambar 3.10. Tampilan Detail Transaksi pada *Dashboard* Grading Member

Data transaksi disajikan dalam bentuk tabel yang terstruktur dengan pengelompokan berdasarkan kategori produk, sehingga memudahkan pengguna dalam membaca dan menganalisis komposisi pembelian. Setiap baris pada tabel mewakili satu kategori produk yang dibeli dalam transaksi tersebut, lengkap dengan metrik finansial yang relevan. Fitur ini memungkinkan pengguna untuk melakukan analisis mendalam terhadap pola pembelian member, mengidentifikasi produk

yang sering dibeli, serta mengevaluasi kontribusi setiap kategori terhadap total nilai transaksi dan margin keuntungan. Informasi ini sangat berharga bagi tim *merchandising* untuk menyusun strategi *cross-selling* dan *up-selling* yang lebih efektif. Tampilan halaman detail transaksi dapat dilihat pada Gambar 3.10.

D Algoritma Sistem

Algoritma sistem menggambarkan logika proses yang berjalan di balik *Dashboard Grading Member*. Algoritma ini direpresentasikan dalam bentuk *pseudocode* untuk memudahkan pemahaman alur kerja sistem tanpa terikat pada sintaks bahasa pemrograman tertentu. Terdapat tiga proses utama yang menjadi inti fungsionalitas sistem, yaitu menampilkan *chart* dan daftar member, menampilkan detail member, serta menampilkan rekap item terlaris.

D.1 Algoritma Menampilkan *Chart* dan Daftar Member

Algoritma pertama menangani proses pengambilan dan penyajian data untuk halaman utama *dashboard*. Proses ini melibatkan pengambilan data *time series* untuk grafik enam bulan terakhir, ringkasan total member per level, serta daftar member yang dapat difilter berdasarkan berbagai kriteria.

```
ALGORITMA Tampil_Chart_dan_Daftar_Member
INPUT: level, branch, area, loyalty, search, page, sort
OUTPUT: Chart progres dan daftar member terfilter
BEGIN
    chartData <- FETCH_CHART_PROGRESS_USING_TIME_SERIES(parameters)
    summaryData <- FETCH_MEMBER_SUMMARY(parameters)
    RENDER(chartData, summaryData)
    filters <- BUILD_FILTER(level, branch, area, loyalty, search)
    memberList <- FETCH_MEMBER_LIST(filters, page, sort)
    RETURN memberList
END
```

Kode 3.1: *Pseudocode* Menampilkan *Chart* dan Daftar Member

D.2 Algoritma Menampilkan Detail Member

Algoritma kedua menangani proses pengambilan informasi detail member ketika pengguna mengklik salah satu baris pada tabel. Proses ini mencakup

pengambilan profil member, statistik transaksi selama tiga bulan terakhir, dan riwayat transaksi terbaru. Data yang berhasil diambil kemudian ditampilkan dalam modal detail member.

```
ALGORITMA Tampil_Detail_Member
INPUT: oba_id
OUTPUT: Detail member dan riwayat transaksi
BEGIN
    member <- FETCH_MEMBER_PROFILE(oba_id)
    IF member IS NULL THEN
        RETURN error
    END IF
    stats <- FETCH_MEMBER_STATS(oba_id, periode_tertentu)
    transactions <- FETCH_MEMBER_TRANSACTIONS(oba_id, N_terbaru)
    TAMPILKAN {member, stats, transactions}
END
```

Kode 3.2: *Pseudocode* Menampilkan Detail Member

D.3 Algoritma Menampilkan Rekap Item Terlaris

Algoritma ketiga menangani proses pengambilan dan penyajian data item terlaris per level member. Algoritma ini menerapkan mekanisme *caching* untuk mengoptimalkan performa, di mana sistem akan memeriksa ketersediaan data di *cache* terlebih dahulu sebelum melakukan *query* ke *database*. Jika data tidak ditemukan di *cache*, sistem akan mengambil data dari *database*, mengelompokkannya berdasarkan cabang dan area kerja, kemudian menyimpannya ke *cache* untuk penggunaan berikutnya.

```
ALGORITMA Tampil_Rekap_Item_Terlaris
INPUT: level, branch, area
OUTPUT: Rekap item terlaris
BEGIN
    key <- BUILD_CACHE_KEY(level, branch, area)
    IF CACHE_EXISTS(key) THEN
        RETURN GET_CACHE(key)
    END IF
    results <- FETCH_TOP_ITEMS(level, branch, area)
    groupedData <- GROUP_BY_BRANCH_AND_AREA(results)
    SET_CACHE(key, groupedData, durasi_tertentu)
    TAMPILKAN groupedData
```

```
13 END
```

Kode 3.3: *Pseudocode* Menampilkan Rekap Item Terlaris

E Pengujian

Pengujian dilakukan menggunakan metode *black box testing* yang berfokus pada validasi fungsionalitas sistem dari perspektif pengguna tanpa memperhatikan struktur internal kode. Metode ini dipilih karena sesuai untuk menguji aplikasi berbasis web di mana pengguna berinteraksi melalui antarmuka tanpa perlu mengetahui logika pemrograman di baliknya.

Pengujian mencakup seluruh fitur utama *Dashboard Grading Member*, meliputi akses halaman *dashboard*, navigasi antar level member, fungsi pencarian dan *filter*, serta tampilan modal detail member dan transaksi. Setiap skenario pengujian dijalankan secara manual dengan memverifikasi kesesuaian antara aksi pengguna dan respons sistem yang diharapkan. Tabel 3.2 menyajikan skenario pengujian beserta hasilnya.

Tabel 3.2. *Black Box Testing Dashboard* Grading Member

No	Skenario Pengujian	Langkah Pengujian	Hasil yang Diharapkan	Status
1	Menampilkan grafik tren member	Membuka halaman utama *dashboard*	Grafik bar menampilkan data member per level selama enam bulan terakhir	Berhasil
2	Menampilkan kartu ringkasan	Membuka halaman utama *dashboard*	Empat kartu menampilkan total member per level sesuai data *database*	Berhasil
3	Navigasi ke halaman detail level	Mengklik salah satu kartu level member	Halaman detail member terbuka sesuai level yang dipilih	Berhasil

No	Skenario Pengujian	Langkah Pengujian	Hasil yang Diharapkan	Status
4	Filter berdasarkan cabang	Mengklik salah satu cabang pada halaman detail member	Data member berubah sesuai cabang yang dipilih	Berhasil
5	Pencarian member berdasarkan ID	Memasukkan ID member pada kolom pencarian	Tabel menampilkan member dengan ID yang sesuai	Berhasil
6	Pencarian member berdasarkan nama	Memasukkan nama member pada kolom pencarian	Tabel menampilkan member dengan nama yang sesuai	Berhasil
7	Filter berdasarkan tipe loyalitas	Memilih tipe loyalitas pada *dropdown filter*	Tabel hanya menampilkan member dengan tipe loyalitas yang dipilih	Berhasil
8	Pengurutan data kolom	Mengklik *header* kolom pada tabel member	Data diurutkan berdasarkan kolom yang diklik secara *ascending/descending*	Berhasil
9	Paginasi tabel member	Mengubah jumlah data per halaman dan navigasi halaman	Tabel menampilkan jumlah data sesuai pilihan dan navigasi berfungsi	Berhasil
10	Menampilkan modal detail member	Mengklik baris member pada tabel	Modal terbuka dan menampilkan profil, statistik transaksi, serta riwayat pesanan	Berhasil
11	Menampilkan modal detail transaksi	Mengklik salah satu pesanan pada modal detail member	Modal detail transaksi terbuka dan menampilkan rincian item yang dibeli	Berhasil

No	Skenario Pengujian	Langkah Pengujian	Hasil yang Diharapkan	Status
12	Memuat lebih banyak riwayat transaksi	Mengklik tombol "Muat Lebih Banyak" pada modal detail member	Riwayat pesanan bertambah dengan data pesanan berikutnya	Berhasil
13	Pergantian level pada rekap item	Mengklik tombol level pada bagian rekap item terlaris	Data item terlaris berubah sesuai level yang dipilih	Berhasil
14	*Toggle* area pada rekap item	Mengklik salah satu area pada bagian rekap item terlaris	Tabel item terlaris untuk area tersebut ditampilkan atau disembunyikan	Berhasil

Berdasarkan hasil pengujian pada Tabel 3.2, seluruh skenario berjalan sesuai ekspektasi. Pengujian dilakukan dalam dua iterasi. Iterasi pertama menghasilkan perbaikan pada tata letak kartu dan warna grafik agar lebih konsisten dengan identitas visual level member. Iterasi kedua berfokus pada penyesuaian tampilan data setelah integrasi dengan *query* asli dari *database*.

F Hasil

Implementasi *Dashboard Grading Member* memberikan dampak positif bagi tim bisnis dan operasional. Pertama, tim bisnis kini dapat memantau distribusi dan tren member secara langsung melalui visualisasi grafik yang intuitif tanpa perlu mengolah data secara manual di Excel. Kedua, informasi detail member termasuk riwayat transaksi dan statistik pembelian tersedia dalam satu tampilan terintegrasi sehingga tim dapat dengan cepat mengidentifikasi member dengan potensi tinggi. Ketiga, rekap produk terlaris per level dan area membantu tim *merchandising* dalam menyusun strategi promosi yang lebih tepat sasaran berdasarkan preferensi setiap segmen member.

3.4 Kendala dan Solusi yang Ditemukan

Selama proses pengembangan sistem dan implementasi berbagai fitur, penulis menghadapi beberapa kendala teknis dan non-teknis yang memerlukan

solusi adaptif. Kendala-kendala tersebut beserta solusi yang diterapkan diuraikan sebagai berikut:

1. **Keterbatasan Performa pada Data dalam Jumlah Besar**
   Ketika jumlah member dan transaksi mencapai ratusan ribu *record*, halaman *dashboard* mengalami penurunan performa yang signifikan, terutama pada fitur *chart* dan tabel member.

   a. Menerapkan paginasi pada tabel daftar member dengan Livewire untuk membatasi jumlah data yang ditampilkan per halaman.
   b. Menggunakan *lazy loading* pada komponen *chart* sehingga data dimuat secara bertahap dan tidak membebani *loading* awal halaman.
   c. Menerapkan *caching* pada data yang jarang berubah seperti daftar cabang dan area kerja untuk mengurangi *query* ke *database*.