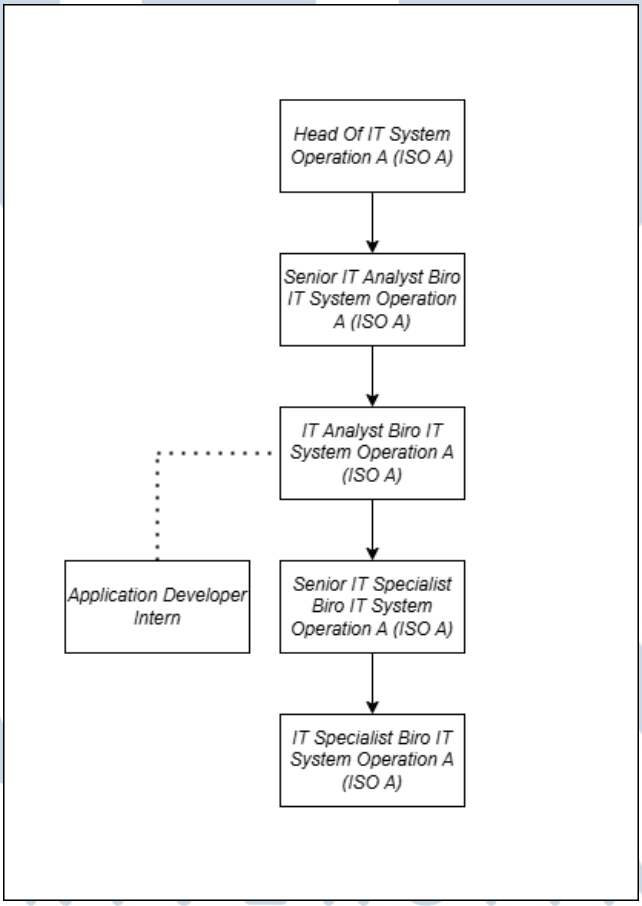
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Program kerja magang dilaksanakan di Biro ISO A, salah satu sub-unit yang berada di bawah koordinasi *IT System Operation* (ISO). Peserta magang menempati posisi sebagai *Application Developer Intern* yang memiliki peran mendukung aktivitas teknis dalam pengembangan sistem di lingkungan BCA.



Gambar 3.1. Kedudukan Intern di Biro ISO A pada PT Bank Central Asia Tbk

Sumber : [11]

Berdasarkan struktur organisasi yang ditampilkan pada Gambar 3.1, posisi *intern* ditempatkan sebagai anggota tim teknis yang berperan dalam menunjang proses pengembangan aplikasi *fullstack*. Walaupun tidak berada dalam jalur struktural utama, *intern* berhubungan langsung dengan *IT Analyst* selaku pembimbing teknis dan mentor harian. Hubungan ini digambarkan melalui garis

putus-putus pada bagan, menandakan bahwa fungsi pembimbingan bersifat non-struktural.

Sepanjang masa magang, arahan teknis sehari-hari diberikan oleh Galuh Dhian Wibowo (*IT Analyst*) melalui penugasan, peninjauan kode (*code review*), serta pemberian masukan terhadap hasil pekerjaan. Selain itu, pengarahan strategis diberikan oleh Suryanto Chandra (*Senior IT Analyst*), sementara pengawasan formal berada di bawah tanggung jawab *Head of IT System Operation A*. Dengan demikian, meskipun bukan bagian struktural inti, *intern* tetap bekerja terintegrasi dalam sistem kerja tim.

Dalam kegiatan harian, koordinasi dengan staf ISO A dilakukan melalui *Microsoft Teams*, yang digunakan untuk komunikasi langsung, rapat tim, maupun pembaruan status *project*. Sementara itu, pengelolaan kode dilakukan dengan menggunakan *GitLab* sebagai platform *version control*. Setiap kontribusi *intern* ditinjau oleh mentor sebelum diintegrasikan, sehingga kualitas dan konsistensi kode dapat terjaga.

Pengalaman ini menunjukkan bahwa meskipun berstatus sebagai *intern*, peran yang dijalankan tetap penting dalam mendukung proses pengembangan sistem. Kegiatan magang tidak hanya memberikan kesempatan untuk mempelajari aspek teknis, tetapi juga pengalaman bekerja secara kolaboratif dalam lingkungan profesional yang terstruktur.

3.2 Tugas yang Dilakukan

Selama pelaksanaan magang di PT Bank Central Asia Tbk., peserta magang berfokus pada implementasi *Content Management System* (CMS) untuk Aplikasi *Incident Report* yang dibangun menggunakan *Spring Boot* di sisi *backend* dan *Next.js* di sisi *frontend*, sebagai bagian dari ekosistem *microservice* internal perusahaan. Tugas yang dilakukan melibatkan pemrograman, analisis sistem, dokumentasi teknis, serta kolaborasi dengan tim *development* dalam rangka mendukung pembangunan sistem *gateway notification* yang terintegrasi.

Adapun tugas-tugas yang dilakukan antara lain:

1. Mempelajari dan Menganalisis Arsitektur Sistem Notifikasi
   Peserta magang ditugaskan untuk memahami rancangan arsitektur aplikasi notifikasi yang terdiri atas berbagai layanan (*service*) seperti *auth service*, *notification service*, dan lain sebagainya. Analisis dilakukan terhadap alur

komunikasi antar layanan dalam ekosistem *microservice* untuk memastikan integrasi CMS dapat berjalan dengan baik.

2. Pengembangan Layanan *Backend* Menggunakan *Spring Boot*
   Peserta magang mengimplementasikan *backend* berbasis *Spring Boot* untuk menangani proses *routing request notification* dari CMS menuju *service* terkait. *Backend* tidak menyimpan data secara independen, melainkan berfungsi sebagai penghubung agar CMS dapat melakukan *fetch data* ke layanan yang bersangkutan.

3. Implementasi *Content Management System* (CMS) Menggunakan *Next.js*
   Peserta magang membangun antarmuka CMS menggunakan *Next.js*, dengan fitur utama berupa tampilan data detail notifikasi, daftar transaksi *incident*, serta beberapa data lainnya.

4. Integrasi *Frontend* dan *Backend*
   CMS yang dikembangkan diintegrasikan dengan *backend Spring Boot* melalui *API gateway*. Proses ini mencakup implementasi *fetch data*, validasi *response*, serta pengelolaan status notifikasi.

5. Penerapan Prinsip *Separation of Concerns*
   Dalam pengembangan aplikasi, peserta magang menerapkan prinsip *Separation of Concerns* untuk memastikan pemisahan tanggung jawab antara lapisan *controller*, *service*, dan *integration*. Penerapan ini mendukung keteraturan sistem sekaligus memudahkan pemeliharaan dalam jangka panjang.

6. Dokumentasi Teknis dan *Deployment Support*
   Peserta magang menyusun dokumentasi teknis terkait implementasi CMS dan *backend*, mencakup desain *endpoint*, alur integrasi, dan panduan penggunaan. Selain itu, peserta magang turut berkontribusi dalam proses persiapan *deployment* aplikasi ke lingkungan *staging* dan *testing* yang dikelola melalui platform *OpenShift* bersama tim *DevOps*.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Menyelesaikan permasalahan teknis dan menambahkan fitur pada *EMS Service*, serta melakukan uji coba implementasi.
2	Membuat *RestAPI* untuk *Telegram Service* dengan *Spring Boot*, mempelajari alur bisnis, menyiapkan *environment GitLab*, dan memisahkan kode *Gaia Service*.
3	Mengembangkan *Telegram Service* awal dan merancang *UI/UX CMS Gateway (login, dashboard,* halaman layanan), serta mengimplementasikan *EMS Service*.
4	Merancang *UI/UX CMS Incident Report (login, group, user, audit log, detail page)* dan melakukan uji coba aplikasi notifikasi.
5	Menyempurnakan desain *CMS Incident Report*, mempelajari *Next.js*, menyiapkan *requirement* project, instalasi kebutuhan pengembangan, serta melakukan *slicing UI*.
6	Mengembangkan fungsionalitas *CMS Incident Report,* implementasi *login middleware,* pembuatan *service* dan *controller*, integrasi *React Query,* dan membuat *sidebar, header*, serta tabel interaktif.
7	Mengembangkan *CMS Incident Report* dengan *service* dan *controller* tambahan, integrasi *React Query* untuk berbagai data, dan membuat tabel interaktif dengan fitur *search* dan *sort*.
8	Menyelesaikan modul *Detail Incident CMS Incident Report,* membuat *service* dan *controller*, integrasi *React Query*, serta mengembangkan *backend service* dan *mutation* untuk fitur edit data.
9	Mengembangkan fitur *CRUD CMS Incident Report* dengan *mutation* untuk *entity Users, Groups, Group Member, Job, Incident*, dan Detail *Incident*, serta integrasi ke *frontend*.
10	Menyelesaikan fitur *create Detail Incident*, menerapkan *soft delete* pada entitas tertentu, menambahkan *reload fetch data*, serta eksplorasi *Email Service.*
11	Mengembangkan *CMS Gateway (login*, *service* dan *controller*, tabel data EMS) dan memperbaiki logika *Email Service* agar terintegrasi.

Minggu Ke -	Pekerjaan yang dilakukan
12	Memperbaiki *bug* pada *CMS Incident Report* dan *CMS Gateway*, melakukan *deployment* ke *development*, serta mengembangkan fitur *summary* dan pencarian pada *Email Service*.
13	Mengembangkan CMS *Incident Report* dan CMS *Gateway* dengan *session cookies* untuk otorisasi, memperbaiki UI/responsivitas, membuat *dashboard notifikasi* dengan filter dan *summary* 7 hari, serta melakukan *deployment* ke *DEV*.

Pada minggu pertama pelaksanaan magang, kegiatan diawali dengan pengenalan lingkungan kerja dan pembekalan terkait budaya kerja di biro ISO A. Kegiatan koordinasi dilakukan secara rutin melalui *meeting daily commitment* setiap pagi untuk menyampaikan progres pekerjaan dan rencana harian. Pada hari-hari awal, peserta magang mempersiapkan pembuatan laporan kinerja bulanan serta melakukan diskusi bersama mentor mengenai *proyek* yang akan di-*handle* selama bulan tersebut. Selanjutnya, dilakukan penyusunan *requirement project* sebagai tahap persiapan awal. Selain itu, kegiatan juga mencakup perbaikan *bug* dan penambahan fitur *Request Time Update* pada *EMS Service*, diikuti dengan sesi diskusi tim untuk mempersiapkan implementasi *EMS Service*. Penutup minggu diisi dengan keterlibatan langsung dalam tahap *uji coba EMS Service* sebelum implementasi, sebagai bagian dari proses pembelajaran teknis dan integrasi kerja tim.

Pada minggu kedua pelaksanaan magang, fokus kegiatan diarahkan pada persiapan dan pengembangan awal *project* baru bernama *Telegram Service*. Kegiatan dimulai dengan *meeting* bersama mentor untuk membahas *requirement* dan *flow business process* yang akan digunakan dalam *project* tersebut. Selanjutnya dilakukan *daily commitment* bersama tim untuk menyampaikan progres pekerjaan dan rencana harian. Pada tahap persiapan, peserta magang melakukan *request GitLab*, menyiapkan folder *Spring Boot*, serta menyiapkan kebutuhan teknis lainnya. Proses pengembangan dimulai dengan pembuatan *business logic* dan *endpoint* untuk *Telegram Service*. Selain itu, dilakukan pemisahan *Gaia Service* dari *MyGateway Service* sehingga *Gaia Service* menjadi layanan terpisah. Pada akhir minggu, dibuat pula *Technical Documentation* untuk *EMS Service* sebagai bagian dari proses pendokumentasian pengembangan sistem dan memastikan kesesuaian implementasi dengan *requirement* yang telah ditetapkan.

Pada minggu ketiga pelaksanaan magang, kegiatan difokuskan pada eksplorasi dan pengembangan awal *Telegram Service* sekaligus perancangan *UI/UX* untuk *CMS Gateway*. Awal minggu dimulai dengan eksplorasi konsep *Telegram Service*, termasuk pembuatan *request* dan *response data*, serta pembuatan *service Telegram*, meskipun pengiriman data masih menunggu *API Telegram*. Selanjutnya, dilakukan *daily commitment* bersama tim untuk menyampaikan progres harian. Peserta magang kemudian melakukan diskusi dengan mentor terkait pengembangan desain *UI/UX CMS Gateway* dan memulai eksplorasi rancangan antarmuka. Proses perancangan mencakup pembuatan halaman *login*, *dashboard*, dan halaman data *Gateway CMS*. Selain itu, dilakukan perancangan halaman layanan untuk *Telegram Service*, *EMS Service*, dan *Email Service*. Menutup minggu, peserta magang melanjutkan proses desain *UI/UX CMS Gateway* dan mengikuti implementasi *EMS Service* bersama tim hingga malam hari.

Pada minggu keempat pelaksanaan magang, fokus kegiatan diarahkan pada perancangan *UI/UX* untuk *CMS Incident Report* serta persiapan implementasi *Notification Application*. Awal minggu dimulai dengan diskusi bersama mentor untuk memahami kebutuhan desain *UI/UX CMS Incident Report* dan mencari ide rancangan yang sesuai. Selanjutnya, dilakukan *daily commitment* bersama tim untuk menyampaikan progres harian. Proses perancangan *UI/UX* mencakup pembuatan halaman *login*, serta desain awal untuk *CMS Incident Report*. Peserta magang juga mempersiapkan tahap implementasi aplikasi notifikasi dan mengikuti sesi *testing* aplikasi tersebut. Pada hari-hari berikutnya, kegiatan dilanjutkan dengan pembuatan desain *UI/UX* untuk berbagai halaman, termasuk *Group Page*, *User Page*, *Audit Log Page*, serta halaman *detail* (*Detail Page 1* hingga *Detail Page 5*). Penutup minggu diisi dengan implementasi *Incident Report Application* dan penyempurnaan desain *UI/UX CMS Incident Report* agar dapat mendukung fungsionalitas aplikasi secara optimal.

Pada minggu kelima pelaksanaan magang, kegiatan difokuskan pada penyempurnaan desain *UI/UX CMS Incident Report* serta persiapan tahap pengembangan. Awal minggu diisi dengan perancangan *UI/UX* untuk *CMS Incident Report* pada halaman *detail* (*Detail Page 6* hingga *Detail Page 8*) dan eksplorasi *Next.js* sebagai teknologi pendukung proses *development*. Selanjutnya dilakukan *daily commitment* bersama tim untuk menyampaikan progres pekerjaan. Peserta magang melanjutkan perbaikan detail pada desain *UI/UX CMS Incident Report* agar lebih sesuai dengan kebutuhan sistem. Pada pertengahan minggu, dibuat *requirement project* untuk *CMS Incident Report*, melakukan instalasi kebutuhan

development, serta mempersiapkan seluruh *requirement* yang diperlukan untuk tahap implementasi. Minggu ini ditutup dengan pembuatan *slicing UI/UX* pada *CMS Incident Report* sebagai persiapan awal proses *coding* dan integrasi sistem.

Pada minggu keenam pelaksanaan magang, fokus kegiatan diarahkan pada pengembangan fungsionalitas *CMS Incident Report*. Awal minggu dimulai dengan pembuatan *slicing UI dan UX* pada *CMS Incident Report* serta eksplorasi penggunaan *React Query*, termasuk *useQuery*, *useMutation*, dan *infiniteQuery* untuk pengelolaan data. Selanjutnya dilakukan *daily commitment* bersama tim untuk menyampaikan progres pekerjaan. Pada tahap ini, peserta magang mengembangkan sistem login aplikasi menggunakan *middleware*, termasuk pembuatan *request* dan *response schema* serta integrasi *React Query* untuk proses login. Proses pengembangan berlanjut dengan pembuatan *service* dan *controller* untuk pengambilan data *Group*, *Group Member*, dan *Job*, lengkap dengan pembuatan *query* di *frontend* menggunakan *React Query*. Pada bagian *frontend*, peserta magang menambahkan komponen *sidebar* untuk logout dan navigasi halaman, *header* untuk menampilkan nama pengguna yang login, serta tabel interaktif menggunakan *Shadcn* dengan fitur pencarian dan sortir kolom pada data *Job*.

Pada minggu ketujuh pelaksanaan magang, kegiatan difokuskan pada pengembangan lanjutan *CMS Incident Report* dengan perluasan fungsionalitas data. Awal minggu dimulai dengan pembuatan *service* dan *controller* untuk pengambilan data *Token*, *User*, *Audit Log*, *Incident*, dan *Detail Incident* (*Detail Incident 1* dan *2*), lengkap dengan pembuatan *query* di *frontend* menggunakan *React Query*. Setiap data ditampilkan dalam bentuk tabel interaktif menggunakan *Shadcn*, dilengkapi fitur pencarian, sortir kolom, dan manajemen data lainnya. Selain itu, dilakukan *daily commitment* bersama tim untuk menyampaikan progres harian serta koordinasi dalam penyempurnaan implementasi data. Proses ini memperkuat integrasi *frontend-backend* dan mempersiapkan *CMS Incident Report* untuk mendukung fitur *CRUD* secara menyeluruh pada modul notifikasi.

Pada minggu kedelapan pelaksanaan magang, fokus kegiatan diarahkan pada penyelesaian modul *Detail Incident 3–8* pada *CMS Incident Report*. Kegiatan dimulai dengan pembuatan *service* dan *controller* untuk pengambilan data *Detail Incident 3* dan *Detail Incident 4*, termasuk pembuatan *query* di *frontend* menggunakan *React Query* serta pembuatan tabel interaktif dengan *Shadcn* yang dilengkapi fitur sortir kolom dan pencarian data. Selanjutnya, proses pengembangan dilanjutkan untuk *Detail Incident 5* hingga *Detail Incident 8* dengan metode

yang sama, sehingga setiap modul memiliki integrasi *backend* dan *frontend* yang konsisten. Selain itu, dilakukan pembuatan *service backend* untuk fitur *edit* pada *Groups*, *Group Member*, *Job*, *Users*, *Incident*, serta *Detail Incident 1–8*. Pada bagian *frontend*, dibuat *useMutation* beserta *request* dan *response schema* untuk *edit Groups* dan *Group Member*. Seluruh kegiatan didukung oleh *daily commitment* bersama tim untuk koordinasi progres dan memastikan target pengembangan modul dapat tercapai sesuai rencana.

Pada minggu kesembilan pelaksanaan magang, kegiatan difokuskan pada pengembangan fitur *CRUD* untuk *CMS Incident Report*, khususnya pada fitur *edit* dan *create* data. Kegiatan dimulai dengan pembuatan *useMutation*, *request*, dan *response schema* untuk *edit Users*, *Incident*, serta *Detail Incident 1–4*, termasuk implementasinya pada *frontend*. Selanjutnya, dibuat *useQueries* dan *useMutation* untuk *edit Detail Incident 5–8* serta penerapan fitur *edit* ke *frontend*, memastikan seluruh data dapat diubah sesuai kebutuhan sistem. Pada hari-hari berikutnya, dilakukan pembuatan *controller* dan *service backend* untuk fitur *create Groups*, *Group Member*, *Users*, *Incident*, serta *Detail Incident 1–8*. Di sisi *frontend*, dibuat *useMutation* beserta *request* dan *response schema* untuk *create Groups*, *Group Member*, *Job*, *Users*, *Incident*, dan *Detail Incident 1–4*, lengkap dengan implementasinya ke *frontend*. Seluruh proses pengembangan dilakukan dengan koordinasi *daily commitment* bersama tim, guna memastikan integrasi *backend* dan *frontend* berjalan lancar serta fitur *CRUD* dapat berfungsi secara optimal.

Pada minggu kesepuluh pelaksanaan magang, kegiatan difokuskan pada penyelesaian fitur *create Detail Incident 5–8*, penerapan mekanisme *soft delete* pada *CMS Incident Report*, serta pengembangan awal *Email Service*. Pekerjaan meliputi pembuatan *useMutation*, *request*, dan *response schema* untuk *create* data serta implementasinya ke *frontend*, perancangan *logic bisnis soft delete*, pembuatan fitur *create Group Member* tanpa duplikasi, serta fitur *delete Users*, *Group*, dan *Group Member* dengan logika *active/inactive* untuk *single* maupun *multi selected* data, lengkap dengan *reload fetch data* menggunakan *TanStack Query*.

Pada minggu kesebelas pelaksanaan magang, kegiatan difokuskan pada pengembangan awal *CMS Gateway* dan penyempurnaan *Email Service*. Pekerjaan meliputi pembuatan *service Email* menggunakan *Spring Boot* untuk mencatat data transaksi, perbaikan sistem login *CMS Incident Report*, persiapan folder dan instalasi requirement untuk *CMS Gateway*, pembuatan *service* dan *controller* backend, serta perbaikan *database* dan *business logic* pada *Email Service*. Selain itu, dilakukan pengembangan sistem login *CMS Gateway*, pembuatan tabel dan fitur

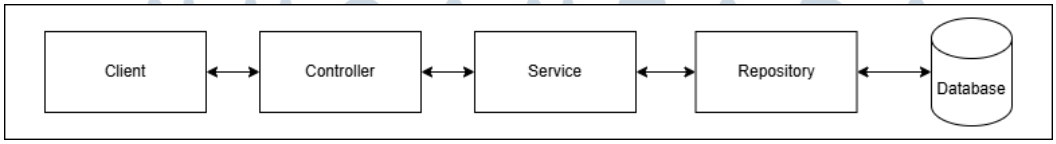
fetching data *EMS* dengan *search* dan *filter*, serta pembuatan *UI* dan *backend* untuk menampilkan data transaksi berdasarkan *ID* pilihan pengguna.

Pada minggu kedua belas pelaksanaan magang, kegiatan difokuskan pada penyempurnaan *CMS Incident Report*, *CMS Gateway*, dan *Email Service*. Pekerjaan meliputi perbaikan bug pada *detail view service EMS* untuk *CMS Gateway*, pembuatan *configmap* dan pengujian *Email Service* di lingkungan *development*, serta perbaikan beberapa fitur pada *service EMS*. Selain itu, dibuat *summary* untuk *Email Service* beserta *service* dan *controller* untuk fitur *search summary* dan *search by ID*, perbaikan bug *CMS Incident Report* akibat *ESLINT error*, dan *deployment CMS Incident Report* ke *development*. Pada akhir minggu, dilakukan dukungan *deployment* untuk *Email Service*.

Pada minggu ketiga belas pelaksanaan magang, kegiatan difokuskan pada pengembangan dan penyempurnaan *CMS Incident Report* dan *CMS Gateway*. Pekerjaan meliputi penambahan *session cookies* pada *CMS Gateway* untuk menyimpan *session token* sebagai *authorization* saat *fetching data*, perbaikan *endpoint* dan *UI detail view* di *CMS Incident Report*, peningkatan *responsiveness*, serta perancangan dan pembuatan halaman *dashboard* dengan fitur *summary 7 hari terakhir*. Selain itu, dibuat *controller* dan *service* untuk *filter data incident* berdasarkan *status* dan *severity*, serta dilakukan *deployment CMS Incident Report* ke *development*. Seluruh proses dilakukan melalui koordinasi *daily commitment* bersama tim untuk memastikan integrasi fitur berjalan optimal.

3.3.1 *Architecture* dan *Framework* yang Digunakan

Proyek ini dikembangkan menggunakan *framework Spring Boot*, yang dipilih karena kemampuannya untuk memulai proyek secara cepat dan memiliki konfigurasi yang sederhana. Dalam arsitektur *microservices*, *Spring Boot* memungkinkan setiap layanan dibuat secara independen namun tetap dapat saling berinteraksi melalui mekanisme komunikasi antar-layanan seperti *REST API*. Framework ini digunakan untuk membangun bagian *backend* dari sistem *Application Incident Report*.



Gambar 3.2. Arsitektur *Layered* Aplikasi *Back-end*

Sumber : [12]

Aplikasi yang dibuat menggunakan arsitektur *layered*, dengan alur kerja seperti yang ditunjukkan pada Gambar 3.2:

1. *Client*: *Client* merupakan sistem atau aplikasi yang mengirimkan permintaan (*request*) dan menerima tanggapan (*response*) dari sistem melalui *controller*. *Client* dapat berupa aplikasi *frontend*, layanan dari pihak ketiga, atau sistem lain yang berinteraksi dengan *backend* menggunakan protokol *HTTP* atau *Restful API*.

2. *Controller*: *Controller* berfungsi sebagai titik masuk utama (*entry point*) untuk semua permintaan dari *client*. Komponen ini bertanggung jawab menerima *request*, memvalidasi data bila diperlukan, serta meneruskan *request* tersebut ke lapisan *service* yang sesuai untuk diproses lebih lanjut.

3. *Service*: Lapisan *service* berisi logika bisnis inti dari aplikasi. Pada lapisan ini, data diproses termasuk validasi, perhitungan, serta pengambilan keputusan yang relevan. *Service* juga dapat berinteraksi dengan *repository* atau memanggil *service* lain apabila arsitektur aplikasi lebih kompleks.

4. *Repository*: *Repository* adalah lapisan yang bertanggung jawab atas akses dan pengelolaan data. Ia menangani komunikasi dengan database dengan menggunakan operasi CRUD (*Create, Read, Update, Delete*). Pada aplikasi ini, akses data dilakukan menggunakan *JDBC Template* yang memungkinkan pelaksanaan perintah SQL secara langsung dan terstruktur.

5. *Database*: *Database* merupakan media penyimpanan utama yang berfungsi menyimpan seluruh data aplikasi. Data yang tersimpan dapat diakses, dimodifikasi, atau dihapus sesuai kebutuhan aplikasi melalui lapisan *repository*.

3.3.2 *Application Incident Report*

Application Incident Report adalah Aplikasi *mobile* dengan arsitektur *microservice* berbasis *Spring Boot* yang berfungsi sebagai sistem aplikasi pengiriman notifikasi kejadian (*incident*) kepada pihak manajer, dengan tujuan utama memastikan pihak manajer dapat memantau kondisi gangguan atau anomali yang terjadi secara *realtime*. Layanan ini mengelola data terkait *incident*, termasuk *summary*, waktu respons (*response time*), dampak (*impact*), serta detail lainnya, sehingga memudahkan proses pengambilan keputusan.

Application Incident Report beroperasi dalam ekosistem *microservices* yang terdiri dari layanan-layanan seperti *Auth-Service, Job Sync Service, Notification Service*, dan CMS. Data *incident* yang diterima dari ELK akan diproses dan dikirimkan ke *Firebase Notification Service*, kemudian diteruskan ke aplikasi *mobile Notification*.

Layanan ini dibangun dengan arsitektur *layered* yang terdiri dari *controller*, *service*, *repository*, dan *integration layer*. Pendekatan ini menjamin modularitas, pemisahan tanggung jawab, serta integrasi yang terukur, sekaligus mengurangi ketergantungan pada proses manual.

A Konsep Dasar

Application Incident Report dibangun menggunakan arsitektur *microservices*, yaitu suatu pendekatan pengembangan sistem di mana aplikasi dibagi menjadi komponen-komponen layanan yang kecil (*services*) dan berdiri sendiri, namun tetap saling berinteraksi melalui protokol komunikasi ringan seperti HTTP.

Microservices merupakan metode arsitektur yang membagi aplikasi menjadi sekumpulan layanan independen, di mana setiap layanan menangani bagian tertentu dari logika bisnis. Pendekatan ini merupakan evolusi dari arsitektur *monolithic*, dengan tujuan agar setiap layanan dapat berjalan secara mandiri, dikembangkan, serta *deploy* secara terpisah [13].

Keunggulan arsitektur *microservices* meliputi kemudahan dalam pemeliharaan karena setiap fungsi utama terisolasi dalam modul terpisah, peningkatan keandalan sistem karena kegagalan pada satu layanan tidak mempengaruhi layanan lainnya, serta kemampuan *scaling* yang fleksibel untuk memenuhi kebutuhan performa sistem [13].

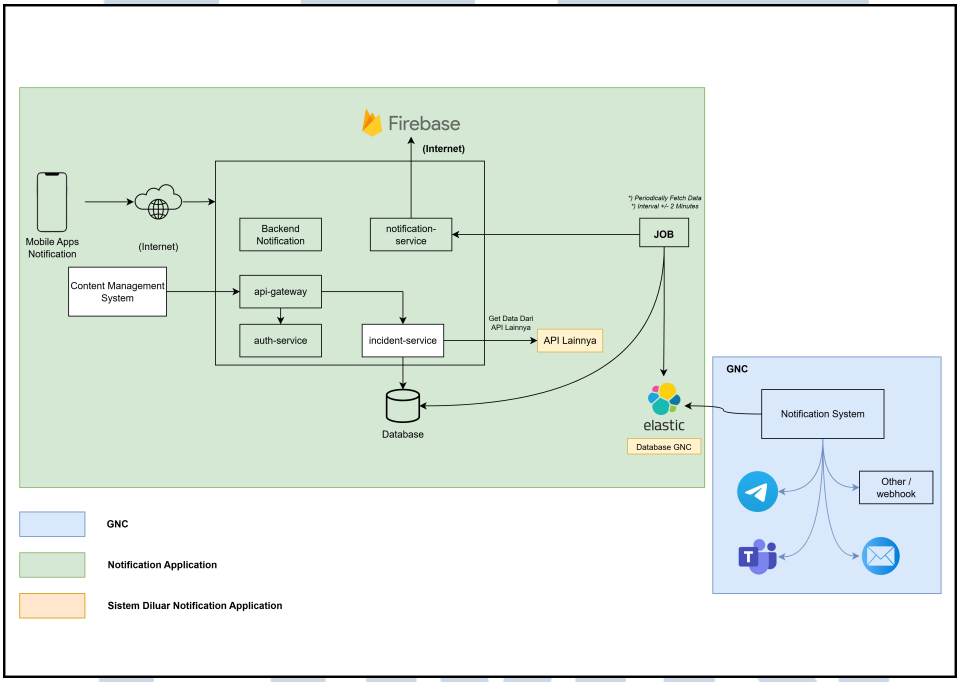
Selama masa magang, pengembangan difokuskan pada *Content Management System* (CMS) dan *Incident-Service*, yang berperan sebagai pusat pengelolaan data *incident* serta penghubung antara sistem CMS dan layanan lainnya dalam arsitektur*Application Incident Report*.

CMS berfungsi sebagai antarmuka bagi *administrator* untuk melakukan operasi *create, read, update*, dan *delete* (CRUD) terhadap data *incident, group, user,* dan *member*. CMS juga menampilkan data *incident* selama periode tertentu melalui halaman *dashboard*, sehingga memudahkan pemantauan aktivitas sistem. Sementara itu, *Incident-Service* menyediakan *endpoint* untuk pengambilan dan

pengelolaan data *incident* yang digunakan oleh CMS dan layanan lain, dengan mengatur proses interaksi terhadap *database internal*.

Layanan lain seperti *Auth-Service, Job Sync, Notification-Service, dan Firebase* telah dikembangkan sebelumnya dan menjadi bagian dari ekosistem *Application Incident Report* yang sudah berjalan. Seluruh layanan ini saling terintegrasi melalui *API Gateway* sebagai *single entry point* yang menangani *routing*, autentikasi, *logging*, dan keamanan data.

Berikut adalah gambaran dari arsitektur sistem *Application Incident Report* yang telah dikembangkan:



Gambar 3.3. Arsitektur*Application Incident Report* dalam *Microservices*

Sumber : [11]

Pada Gambar 3.3, setiap komponen dalam *Application Incident Report* memiliki fungsi tersendiri yang saling terhubung, yaitu:

1. *Content Management System* (CMS)
   CMS merupakan aplikasi yang menjadi antarmuka bagi *Administrator* untuk mengelola data *incident*. Melalui CMS, admin dapat membuat, dan mengubah *data incident*, *detail data incident, group, users, group member*, dan lain lannya yang nantinya akan diteruskan ke backend. Serta admin juga dapat memonitoring *data incident* selama 7 hari sebelumnya melalui halaman *dashboard*

2. *API Gateway*

   Seluruh permintaan dari CMS ataupun aplikasi *mobile* akan terlebih dahulu melewati *API Gateway*. *Gateway* ini berfungsi sebagai pintu masuk tunggal (*single entry point*) yang melakukan *routing* ke *service* terkait, sekaligus menangani kebutuhan *logging*, *rate limiting*, dan keamanan dasar.

3. *Auth-Service*

   Sebelum *request* diproses lebih jauh, *API Gateway* akan mengarahkan ke *Auth-Service* untuk proses autentikasi. Layanan ini bertugas memverifikasi kredensial dan menghasilkan *token* (JWT) yang digunakan sebagai identitas akses agar hanya pengguna yang sah dapat melakukan *request*.

4. *Incident-Service*

   menyediakan *endpoint* untuk operasi *fetch* dan *CRUD* data *incident* yang digunakan oleh CMS dan *client* lain. *Service* ini berinteraksi dengan *database* internal untuk menyimpan dan mengambil data *incident*.

5. *Elastic* (ELK)

   Elastic berperan sebagai tempat pengindeksan dan penyimpanan *log*/hasil *indexing* dari GNC. Sistem GNC atau *pipeline logging* menulis *data* ke *Elastic*, sehingga *data incident* dapat dicari dan diambil secara efektif.

6. *Job Sync*

   Job Sync adalah proses terjadwal (periodik) yang melakukan *query* ke *Elastic* untuk mengambil data *incident* terbaru. Setelah mengambil data, *Job Sync* melakukan transformasi/*parsing* (memakai *regex*) membentuk *object* notifikasi, menyimpan *record* ke *database*, dan menyiapkan *payload* untuk dikirim ke *Notification-Service*.

7. *Notification-Service*

   Layanan ini menerima *object*/*payload* dari *Job Sync*, memastikan format *payload* sesuai, lalu meneruskan permintaan pengiriman ke *Firebase* sehingga notifikasi dapat dikirimkan ke perangkat *mobile*.

8. Firebase

   Sebagai penyedia layanan *push notification* (FCM), *Firebase* menerima *payload* dari *Notification-Service* dan menyalurkannya sebagai *push notification* ke aplikasi *mobile* pengguna.

9. GNC (*Notification Center*)
   GNC merupakan aplikasi internal yang digunakan oleh tim operasional untuk melakukan pengisian data *incident* diinput di GNC akan disimpan ke *Elastic* sehingga dapat ditarik oleh sistem lain, termasuk *Application Incident Report*.

B Metode Pengembangan Sistem

Metode perancangan yang digunakan dalam pengembangan *Content Management System* (CMS) *Incident Report* adalah *Agile Software Development Life Cycle* (SDLC). *Agile* merupakan pendekatan pengembangan perangkat lunak yang menekankan fleksibilitas, kolaborasi, serta kemampuan beradaptasi terhadap perubahan kebutuhan secara cepat. Pendekatan ini terbukti efektif dalam meningkatkan efisiensi proses kerja, mempercepat pengambilan keputusan, serta mendukung penyelesaian proyek melalui siklus pengembangan yang iteratif dan adaptif [14].

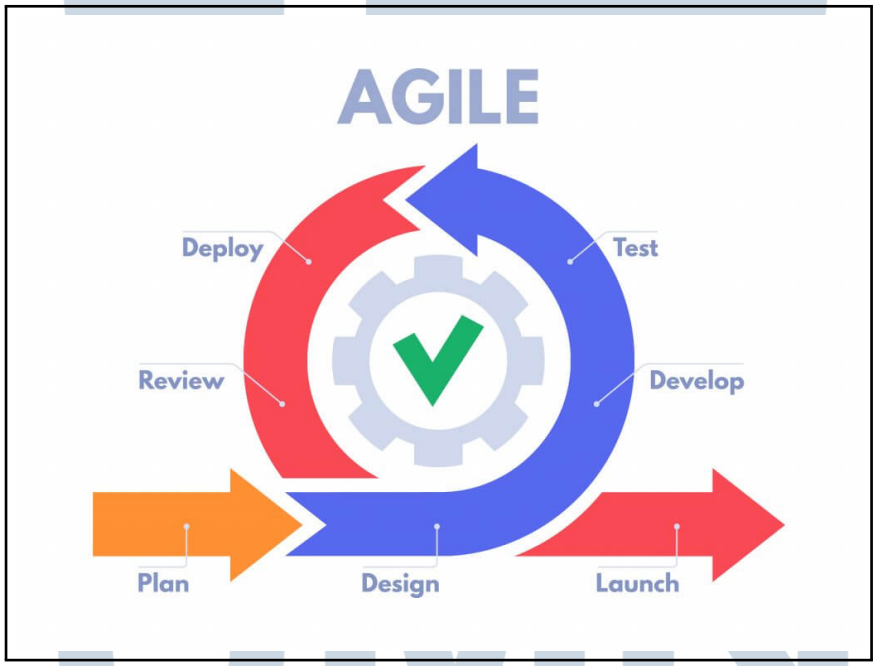
Agile SDLC terdiri atas siklus pengembangan berulang (*iterative cycle*) yang menghasilkan *increment* berupa fitur yang siap diuji, dievaluasi, dan dikembangkan lebih lanjut. Tahapan utama dalam *Agile* meliputi:

1. *Planning*: proses merumuskan kebutuhan sistem serta menyusun *backlog* prioritas.
2. *Design*: pembuatan rancangan arsitektur, alur proses, *API*, dan antarmuka pengguna.
3. *Development*: implementasi fitur inti secara bertahap pada setiap *sprint*.
4. *Testing*: verifikasi kualitas fitur melalui uji unit, integrasi, dan penyempurnaan hasil pengujian.
5. *Launch*: persiapan rilis fitur ke *environment* berikutnya.
6. *Review*: evaluasi capaian *sprint* dan identifikasi peningkatan yang perlu dilakukan.
7. *Deployment*: penerapan fitur menggunakan pipeline GitLab dan OpenShift sesuai standar DevOps.

Pada proyek *CMS Incident Report*, metode *Agile* diterapkan melalui pembagian *sprint* mingguan sebagaimana tercantum pada Tabel 3.1. Setiap *sprint*

memiliki sasaran kerja (*sprint goal*), cakupan fitur, serta target keluaran yang jelas. Selain itu, proses *daily commitment* dilakukan setiap hari untuk memantau perkembangan pekerjaan dan mengatasi hambatan teknis yang muncul. Seluruh kode juga melalui proses *code review* melalui GitLab guna memastikan kualitas implementasi. Hal ini sesuai dengan pandangan Omonije (2024) yang menyatakan bahwa *Agile* mampu meningkatkan efektivitas kolaborasi tim dan memastikan pengiriman nilai bisnis secara bertahap dan terukur [14].

Model *Agile SDLC* yang digunakan dalam proyek ini ditunjukkan pada Gambar 3.4, yang menampilkan alur iteratif mulai dari tahap perencanaan hingga proses *deployment*.



Gambar 3.4. Metode Pengembangan *Agile*
Sumber : [14]

Sebagai bagian dari dokumentasi metodologi, rencana waktu pengembangan juga disusun dalam bentuk diagram *Gantt*. Diagram ini menunjukkan distribusi aktivitas selama dua belas minggu masa pengembangan berdasarkan pendekatan *Agile*, mulai dari tahap inisiasi, pengembangan modul tiap fitur, hingga tahap pengujian dan penyusunan laporan. Rincian rencana waktu tersebut ditunjukkan pada Tabel 3.2 berikut.

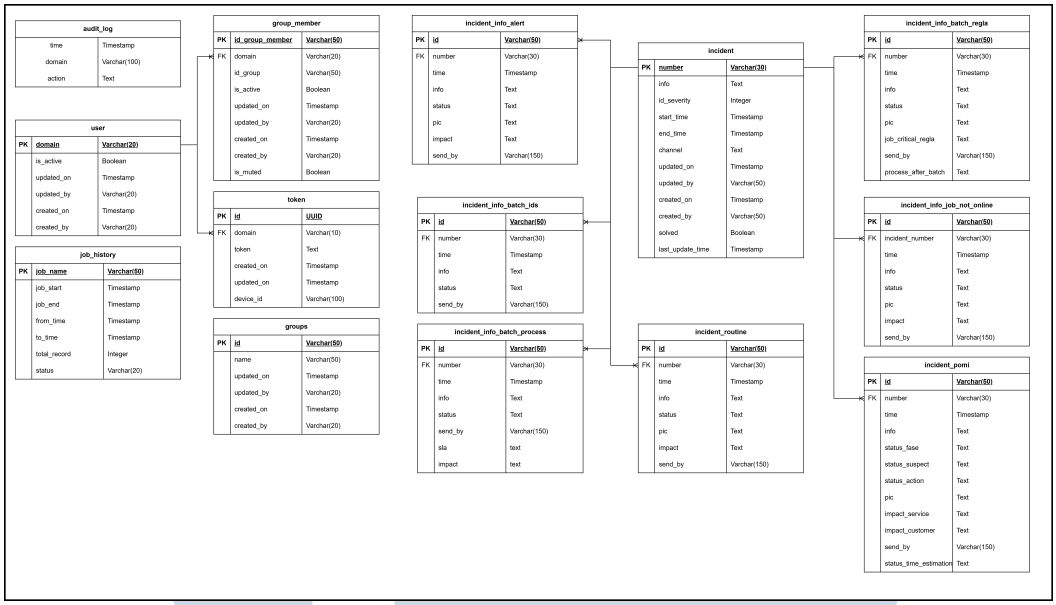
Tabel 3.2. Rencana Waktu Pengembangan CMS Incident Report

Kegiatan	Minggu ke-											
	1	2	3	4	5	6	7	8	9	10	11	12
Setup lingkungan, analisis kebutuhan, dan penyusunan backlog												
Pengembangan modul Login dan Authentication												
Pengembangan modul Groups dan Users (CRUD)												
Pengembangan Audit Log												
Pengembangan modul Incident (CRUD)												
Pengembangan modul Detail Incident												
Pengembangan Dashboard Incident Report												
Testing, Bug Fixing, dan Review Sprint												
Penyusunan laporan dan dokumentasi												

Dengan pendekatan *Agile SDLC* dan perencanaan waktu yang terstruktur, proses pengembangan *CMS Incident Report* dapat berjalan secara iteratif, adaptif, dan terkendali. Setiap *increment* fitur dapat diuji, dievaluasi, dan disempurnakan sebelum dirilis, sehingga kualitas sistem tetap terjaga dan sesuai dengan kebutuhan bisnis internal BCA.

C *Database*

Desain *database CMS Incident Report Application* dibangun menggunakan *PostgreSQL* sebagai sistem manajemen basis data relasional. Perancangan ini bertujuan untuk memastikan bahwa data *incident, user,* serta notifikasi dapat dikelola dengan baik, terstruktur, dan mendukung kebutuhan integrasi antarmodul.



Gambar 3.5. Desain Database *Incident Report Application*

Sumber : [11]

Pada Gambar 3.5, terlihat bahwa *database* terdiri dari beberapa tabel utama yang saling terhubung untuk mendukung proses pengelolaan *incident* dan pengiriman notifikasi. Penjelasan dari masing-masing tabel adalah sebagai berikut:

1. *User*

   Menyimpan data akun pengguna yang memiliki akses ke *Incident Report Application*. Informasi yang tercatat meliputi nama pengguna, status aktif, serta pencatatan waktu pembuatan dan pembaruan data.

2. *Groups*

   Menyimpan data unit atau tim tertentu di dalam sistem.

3. *Group Member*

   Menjadi tabel penghubung (*junction table*) antara *user* dan *groups*. Tabel ini menentukan keanggotaan pengguna dalam suatu grup, serta dapat menyimpan status keaktifan atau preferensi notifikasi.

4. *Token*

   Menyimpan *device token* yang digunakan untuk pengiriman *push notification* ke perangkat pengguna melalui *Firebase*. *Token* ini menjadi identitas perangkat agar notifikasi dapat diarahkan secara tepat.

5. *Job History*

   Mencatat riwayat eksekusi *job* sinkronisasi, termasuk informasi waktu mulai,

waktu selesai, jumlah data yang berhasil diproses, serta status hasil eksekusi (*success* atau *failed*).

6. *Audit Log*
   Berfungsi sebagai pencatatan aktivitas sistem, yang dapat digunakan untuk keperluan monitoring dan keamanan.

7. *Incident*
   Tabel inti yang menyimpan ringkasan *incident*, meliputi informasi tingkat keparahan (*severity*), waktu mulai dan selesai *incident*, status penyelesaian, serta metadata terkait pembuat dan pembaruan.

8. *Incident Detail* (Subtabel insiden)
   Menyimpan informasi detail berdasarkan jenis *incident*. Subtabel ini memiliki relasi langsung ke *incident*, di antaranya:

   (a) *incident_info_alert* : mencatat data detail *incident* yang berasal dari *alert monitoring server*.

   (b) *incident_info_batch_ids* : menyimpan data detail *incident* yang berasal dari berimpact ke proses *Batch IDS*.

   (c) *incident_info_batch_process* : menyimpan data detail *incident* yang menjelaskan proses *batch IDS* sudah berjalan sejauh mana.

   (d) *incident_info_batch_regla* : menyimpan data detail *incident* yang menjelaskan proses *batch Regla* sudah berjalan sejauh mana.

   (e) *incident_routine* : mencatat data detail *Incident* yang terjadi diluar proses monitoring.

   (f) *incident_pomi* : menyimpan data detail *incident* yang bersifat berpotensi menjadi *urgent* dan memiliki impact pada finansial.

   (g) *incident_info_job_not_online* : menyimpan data detail *Incident* ketika terjadi kegagalan pada suatu job.

D Mekanisme Pengelolaan Data *Incident*

Berdasarkan Gambar 3.3, Pengelolaan data *incident* dalam sistem ini dilakukan melalui dua mekanisme utama, yaitu proses otomatis melalui integrasi dengan *ELK Stack* dan proses manual melalui *Content Management System* (CMS).

Kedua mekanisme ini saling melengkapi untuk memastikan data yang tersimpan di *database* tetap konsisten dan dapat diakses pengguna melalui aplikasi.

Proses otomatis dimulai dari penarikan data *incident* melalui *job synchronization* yang terhubung ke *ELK Stack*. Data yang berhasil ditarik kemudian disimpan ke dalam *database* utama. Karena format data dari ELK tidak selalu seragam, sistem menggunakan *regular expression* (*regex*) untuk menyesuaikan pola teks agar sesuai dengan struktur tabel yang tersedia. Dengan cara ini, informasi penting seperti tingkat keparahan *incident*, waktu, hingga deskripsi dapat diekstrak dan dipetakan dengan baik.

Namun, dalam praktiknya proses otomatis ini tidak selalu berjalan sempurna. Beberapa data tidak terbaca dengan benar akibat format yang tidak sesuai atau adanya atribut yang hilang. Kondisi ini menimbulkan permasalahan dalam akurasi pencatatan *incident* di dalam sistem.

Untuk mengatasi keterbatasan tersebut, disediakan CMS *internal* sebagai solusi *input* dan koreksi manual. Melalui CMS, admin atau pengguna terkait dapat menambahkan *incident* baru atau memperbaiki data yang salah dengan mengikuti *template* yang sudah disediakan. Perlu dicatat bahwa data yang dicatat melalui CMS hanya melakukan pembaruan ke *database* dan tidak diteruskan ke *Notification Service*. Dengan demikian, informasi dari CMS tidak akan memunculkan notifikasi pada perangkat pengguna, tetapi tetap dapat diakses melalui aplikasi ketika pengguna membuka detail *incident*.

Berbeda dengan data dari CMS, hanya *incident* yang berhasil diproses melalui alur otomatis dari *ELK Stack* yang akan didistribusikan ke *Notification Service*. Data tersebut kemudian diteruskan ke *Firebase* untuk dikirimkan sebagai *push notification* ke perangkat pengguna. Dengan desain ini, sistem dapat memastikan bahwa informasi penting tetap dikirim secara *real-time*, sementara data yang tidak terbaca atau bermasalah tetap tercatat dan dapat ditinjau ulang melalui mekanisme manual di CMS.

E Analisis Komparatif penggunaan *REGEX* dan *Porting Incident Report*

Pada sistem *Incident Report* yang berjalan sebelum pengembangan *Content Management System* (CMS), proses ekstraksi data *incident* dari *log* dilakukan menggunakan pendekatan *regular expression* (*regex*). Pendekatan ini mengandalkan kesesuaian pola teks yang telah didefinisikan sebelumnya untuk memecah pesan *log* menjadi atribut terstruktur seperti jenis *incident*, waktu

kejadian, layanan terdampak, dan *severity level*. Dalam kondisi ideal, *regex* mampu melakukan pemrosesan data secara otomatis dengan latensi rendah dan implementasi yang relatif sederhana. Namun, berbagai studi menunjukkan bahwa pendekatan *log parsing* berbasis aturan eksplisit seperti *regex* memiliki keterbatasan ketika dihadapkan pada variasi format *log* yang bersifat dinamis [15].

Dalam praktik operasional di lingkungan perbankan, format pesan *incident* tidak selalu bersifat statis. Perubahan minor pada template pesan—seperti perbedaan tanda baca, perubahan urutan kata, serta penambahan atau penghapusan frasa tertentu—sering terjadi akibat penyesuaian operasional atau *human error* dari masing-masing operator. Perubahan tersebut menyebabkan pola *regex* gagal melakukan pencocokan (*pattern mismatch*), sehingga data *incident* tidak dapat diekstraksi dan akhirnya tidak tercatat dalam sistem notifikasi. Kondisi ini sejalan dengan temuan penelitian yang menyatakan bahwa metode *parsing* tradisional sulit beradaptasi terhadap perubahan struktur *log*, sehingga berpotensi menyebabkan hilangnya informasi penting dalam sistem monitoring.

Sebagai alternatif terhadap pendekatan berbasis *regex*, dalam literatur dan praktik industri dikenal konsep *incident report porting* secara terpilih. Dalam pembahasan ini, istilah *porting incident report* digunakan untuk menggambarkan pendekatan penyesuaian laporan *incident* secara selektif dari berbagai operator ke dalam format standar, yang berfungsi sebagai alternatif terhadap proses *parsing* otomatis berbasis *regex*. Pendekatan ini menekankan proses pemetaan dan normalisasi laporan insiden dari berbagai sumber ke dalam skema data yang seragam, tanpa sepenuhnya bergantung pada kecocokan pola teks. Sebaliknya, mekanisme ini memanfaatkan lapisan validasi, serta aturan pemetaan (*mapping rules*) yang lebih adaptif terhadap variasi format pesan, sehingga dinilai lebih *robust* dalam menjaga konsistensi data terstruktur, khususnya pada lingkungan sistem terdistribusi yang kompleks dan dinamis [16].

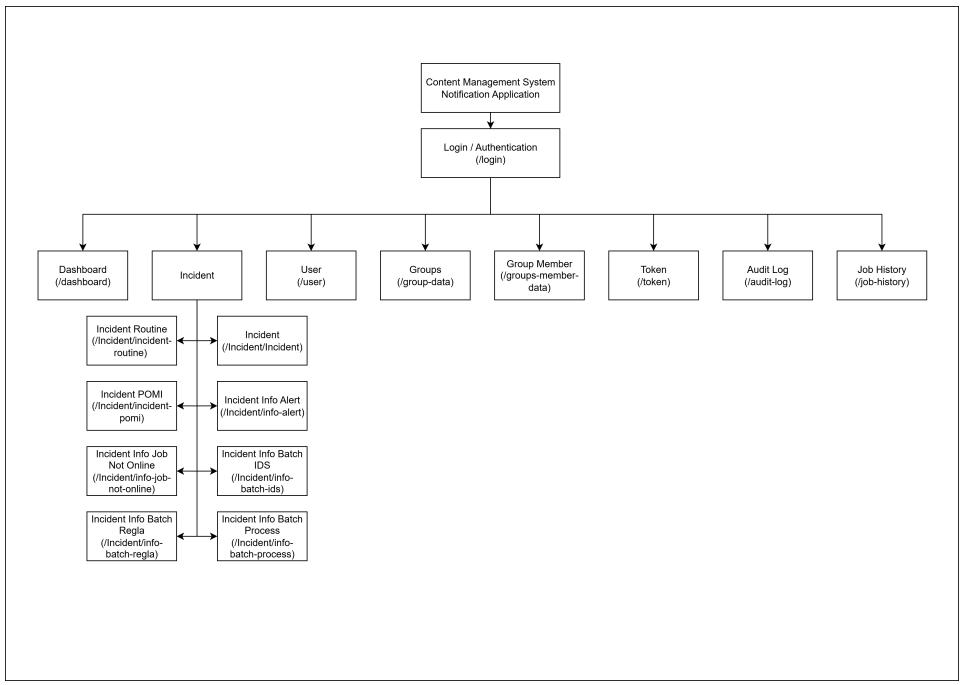
Meskipun pendekatan *porting incident report* menawarkan tingkat ketahanan (*robustness*) yang lebih tinggi terhadap perubahan format, implementasinya memerlukan kompleksitas sistem yang lebih besar, termasuk *database* lintas operator, logika normalisasi, serta mekanisme validasi yang konsisten. Dalam konteks pengembangan sistem di PT Bank Central Asia, pendekatan ini belum diimplementasikan secara penuh. Namun, analisis komparatif menunjukkan bahwa konsep *incident report porting* dapat menjadi arah pengembangan lanjutan untuk meningkatkan keandalan sistem notifikasi seiring meningkatnya kompleksitas dan *volume* data *incident*.

Berdasarkan kondisi tersebut, CMS *Incident Report* dikembangkan sebagai solusi antara (*bridging solution*) yang menjembatani keterbatasan pendekatan *regex* tanpa mengubah arsitektur sistem yang sudah ada secara signifikan. CMS berperan sebagai lapisan koreksi dan validasi manual, di mana laporan *incident* yang gagal diproses oleh *regex* tetap dapat dikelola, diperbaiki, dan disimpan secara terstruktur sebelum diteruskan ke sistem notifikasi. Pendekatan ini meningkatkan reliabilitas data *incident* sekaligus memberikan fleksibilitas operasional tanpa menambah kompleksitas integrasi lintas operator, serta sejalan dengan rekomendasi literatur terkait peningkatan ketahanan pemrosesan *log* terhadap perubahan format [15] [16].

3.3.3 *Content Management System Incident Report Application*

A *Sitemap Content Management System Incident Report Application*

Sitemap pada *Content Management System (CMS) Incident Report Application* menggambarkan struktur navigasi serta hubungan antarhalaman dalam aplikasi. Melalui *sitemap* ini, dapat dilihat bagaimana setiap modul dan submodul saling terhubung sehingga membentuk alur penggunaan aplikasi secara keseluruhan.



Gambar 3.6. Struktur Modul CMS *Incident Report Application*

Sumber : [11]

Gambar 3.6 menunjukkan *sitemap* CMS *Incident Report Application* yang

terdiri atas beberapa halaman utama, dimulai dari proses *Login / Authentication* hingga berbagai fitur fungsional seperti *Dashboard, Incident, User, Groups, Group Member, Token, Audit Log*, dan *Job History*.

1. *Login / Authentication (/login)*
   Halaman ini merupakan pintu masuk utama bagi pengguna untuk mengakses sistem CMS. Pengguna wajib melakukan proses autentikasi menggunakan *username* dan *password*. Setelah berhasil *login*, sistem akan menghasilkan *token* dan *session* agar pengguna dapat mengakses modul lain.

2. *Dashboard*
   Menampilkan ringkasan (*overview*) data *incident* yang ada dalam sistem. Informasi yang ditampilkan meliputi jumlah *incident* dalam tujuh hari terakhir, *incident* aktif, *incident* terselesaikan, dan tren harian. Halaman ini berfungsi sebagai pusat pemantauan kondisi sistem secara umum.

3. *Incident*
   Merupakan bagian utama dari CMS yang digunakan untuk pengelolaan *incident*. Modul ini memiliki beberapa cabang halaman (*submodul*) yang menampilkan detail dari berbagai jenis *incident*, yaitu:

   - ☐ *Incident*
     Daftar dan pengelolaan data *incident* utama (*create, edit*).
   - ☐ *Incident Info Alert*
     Detail *Incident* yang berasal dari *alert monitoring server*.
   - ☐ *Incident Info Batch IDS*
     Detail *Incident* yang berasal dari berimpact ke proses *Batch IDS*.
   - ☐ *Info Batch Process*
     Detail yang menjelaskan proses batch IDS sudah berjalan sejauh mana.
   - ☐ *Incident Info Batch Regla*
     Detail yang menjelaskan proses batch Regla sudah berjalan sejauh mana.
   - ☐ *Incident Info Job Not Online*
     Detail *Incident* ketika terjadi kegagalan pada suatu *job*.
   - ☐ *Incident POMI*
     Detail *incident* yang bersifat berpotensi menjadi *urgent* dan memiliki *impact* pada finansial.

- *Incident Routine*

  Detail *Incident* yang terjadi diluar proses monitoring.

4. *User*

   Halaman ini berfungsi untuk mengelola data pengguna yang memiliki akses ke aplikasi notifikasi. *Administrator* dapat menambahkan, mengubah, atau menonaktifkan akun pengguna melalui halaman ini.

5. *Groups*

   Halaman *Groups* digunakan untuk Mengelola entitas grup (kelompok pengguna) yang digunakan untuk pengaturan distribusi dan hak akses.

6. *Group Member*

   Halaman ini berfungsi untuk mengelola keanggotaan pengguna pada grup tertentu, termasuk atribut seperti status aktif atau preferensi notifikasi.

7. *Token*

   Menyimpan *device token* atau informasi autentikasi yang digunakan untuk integrasi notifikasi (misal ke *Firebase*).

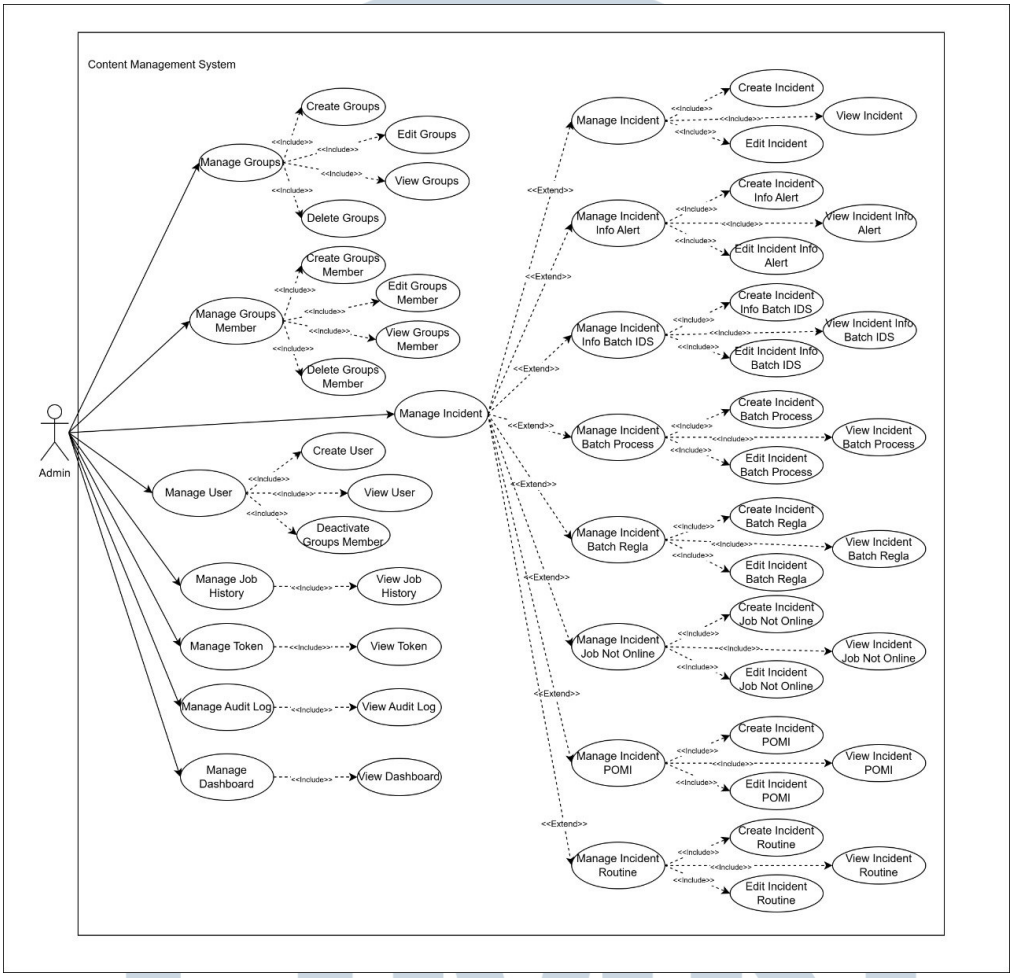
8. *Audit Log*

   Mencatat aktivitas penting pengguna dan perubahan data sebagai fasilitas *monitoring* dan penelusuran (*troubleshooting*) di CMS.

9. *Job History*

   Menyimpan riwayat eksekusi *job* sinkronisasi (mis. *Job Sync*) dan metadata terkait (waktu mulai, waktu selesai, jumlah *record*, status).

B *Use Case CMS Incident Report Application*



Gambar 3.7. *Use Case* Diagram CMS *Incident Report Application*

Use case diagram pada Gambar 3.7 menggambarkan hubungan antara pengguna dan seluruh fungsi yang terdapat dalam sistem *Content Management System (CMS) Incident Report*. pengguna utama dalam sistem ini adalah Admin, yang memiliki kendali penuh terhadap seluruh modul dan fitur yang tersedia. Diagram ini berfungsi untuk memberikan gambaran umum mengenai aktivitas yang dapat dilakukan oleh *admin* serta keterkaitan antarproses di dalam sistem.

Pada sistem ini, *admin* dapat melakukan berbagai operasi pengelolaan data seperti menambah, mengubah, menghapus, dan menampilkan data untuk beberapa entitas utama, yaitu *User*, *Groups*, *Group Member*, *Incident*, *Audit Log*, *Job History*, *Token*, dan *Dashboard*. Setiap modul memiliki serangkaian *use case* tersendiri yang menggambarkan alur interaksi antara *admin* dan sistem.

Admin dapat mengelola data pengguna melalui fitur *Manage User*, yang mencakup pembuatan akun baru, pengeditan data pengguna, penonaktifan akun, serta melihat daftar pengguna yang terdaftar. Pada fitur *Manage Groups* dan *Manage Group Member*, *admin* dapat membuat grup baru, menambahkan atau menghapus anggota grup, serta memperbarui data grup sesuai kebutuhan.

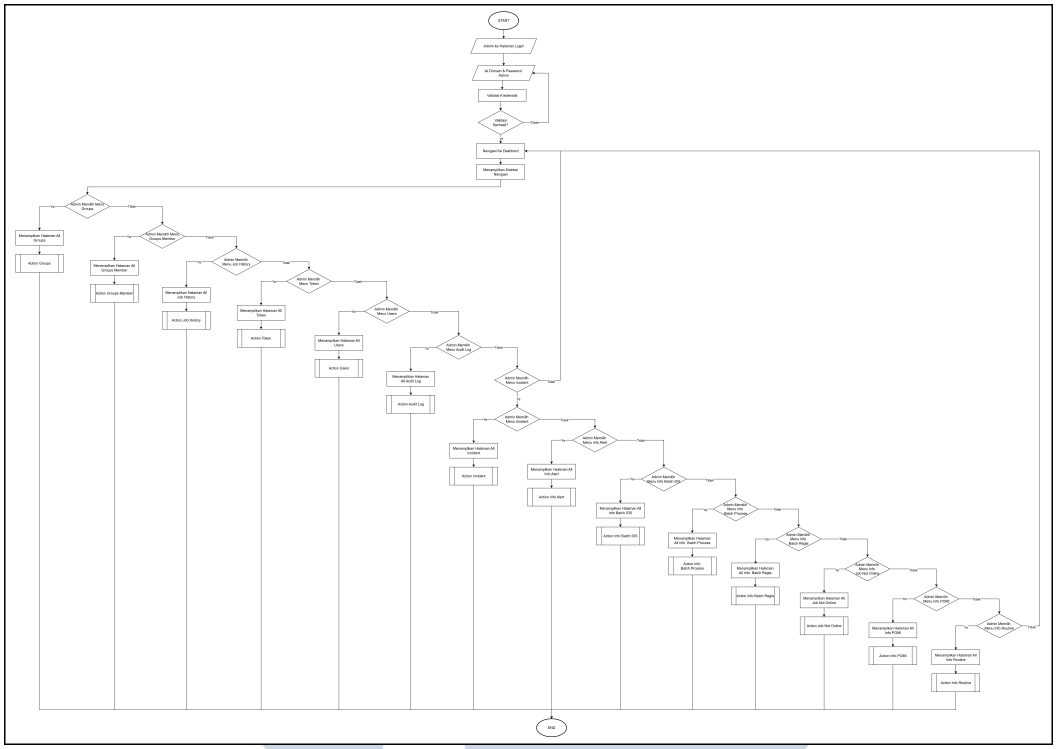
Modul *Manage Incident* menjadi salah satu komponen utama dalam sistem ini karena berfungsi untuk mengelola data insiden yang terjadi pada berbagai layanan. Di dalamnya terdapat beberapa subproses tambahan seperti *Info Alert*, *Batch Process*, *Batch Regla*, *Job Not Online*, *POMI*, dan *Routine*. Setiap subproses memiliki alur pembuatan, pengeditan, dan peninjauan data insiden yang saling berkaitan, yang direpresentasikan dalam diagram melalui hubungan *extend* dan *include*.

Selain itu, *admin* juga dapat memantau aktivitas didalam sistem CMS melalui fitur *Manage Audit Log*, yang merekam setiap tindakan yang dilakukan dalam aplikasi, seperti pembuatan, pembaruan, atau penghapusan data. Fitur *Manage Job History* dan *Manage Token* digunakan untuk melihat riwayat *Job History* sistem serta memantau status *token authorization*. Sementara itu, fitur *Manage Dashboard* berfungsi untuk menampilkan ringkasan data utama, seperti jumlah insiden aktif terkini yang terjadi di sistem.

Secara keseluruhan, *use case diagram* ini menunjukkan bahwa seluruh proses dalam *CMS Incident Report* terpusat pada peran *admin* sebagai pengelola utama sistem. Melalui struktur modular, sistem ini dirancang agar setiap aktivitas pengelolaan data dapat dilakukan dengan efisien, terintegrasi, dan mudah dipantau dalam satu *platform* terpusat.

C *Flowchart Utama CMS Incident Report Application*

Flowchart *CMS Incident Report Application* menggambarkan alur kerja sistem dari proses *login* pengguna hingga interaksi dengan berbagai modul. Proses dimulai dengan autentikasi *login*, kemudian pengguna diarahkan ke *Dashboard*. Dari sini, pengguna dapat memilih modul sesuai kebutuhan, seperti *Incident Management, User, Groups, Group Member, Audit Log,* atau *Job History*. Masing-masing modul memiliki alur proses tersendiri, baik itu untuk membuat data baru, memperbarui data, menghapus, maupun melakukan pencarian.

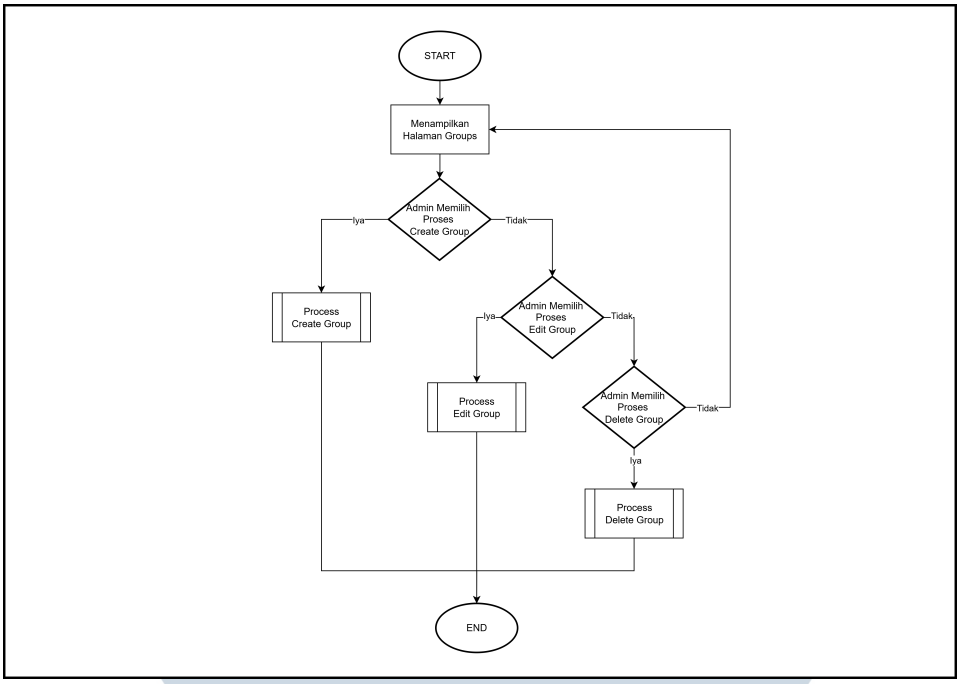


Gambar 3.8. *Flowchart* Utama CMS *Incident Report Application*

Sumber : [11]

Pada Gambar 3.15, menampilkan gambaran dari alur kerja CMS *Incident Report Application*. Untuk memberikan pemahaman lebih detail, berikut ditampilkan *flowchart* pada setiap fitur.

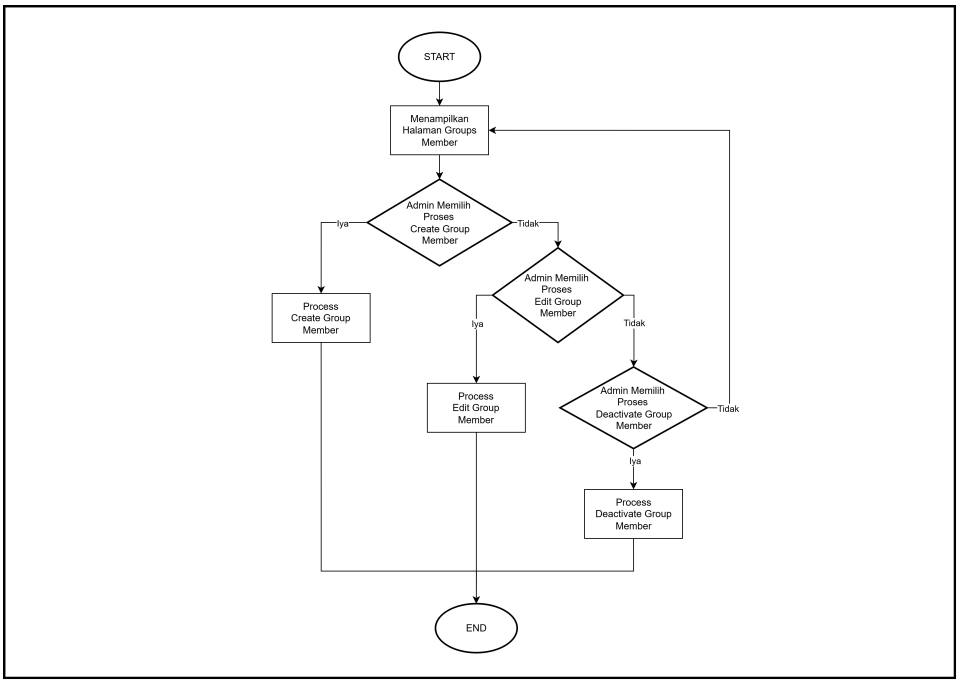
1. Flowchart Page Groups



Gambar 3.9. *Flowchart* Fitur Data *groups*

Sumber : [11]

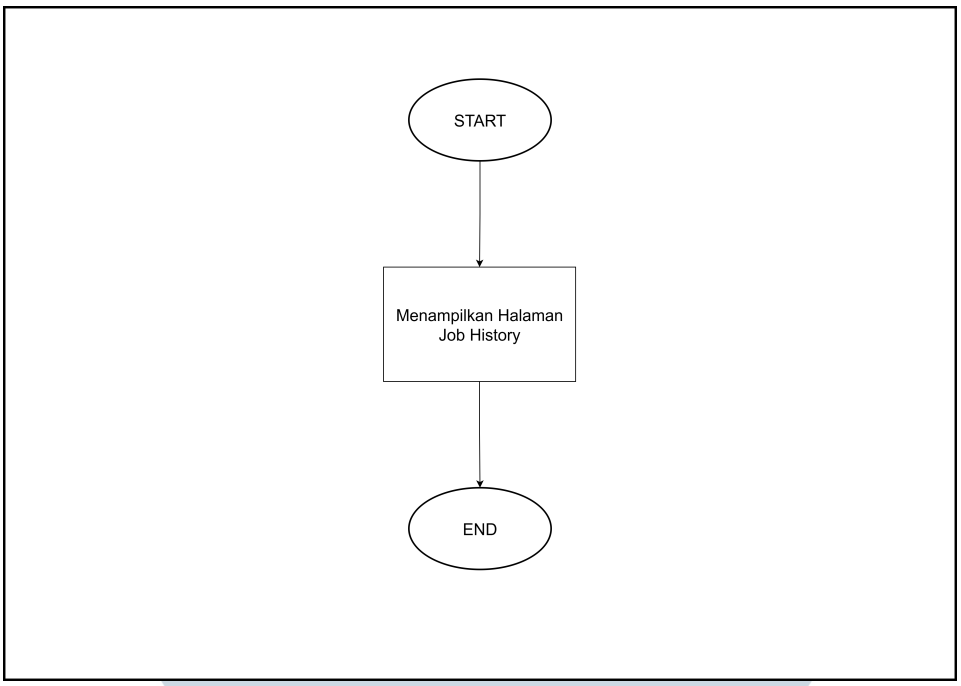
2. Flowchart Page Groups Member



Gambar 3.10. *Flowchart* Fitur Data *groups member*

Sumber : [11]

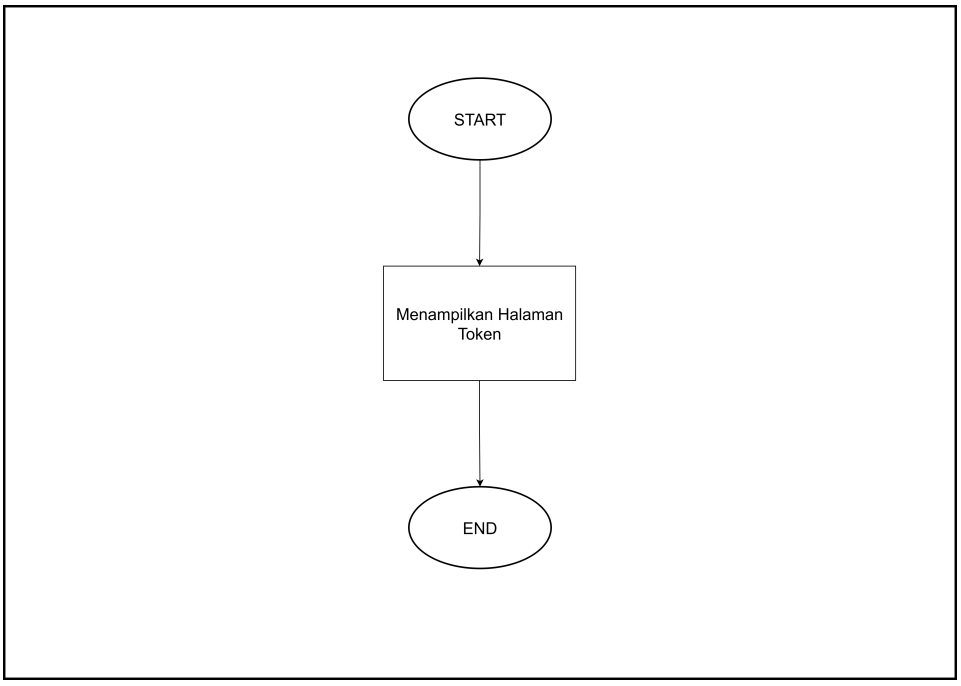
3. Flowchart Page Job History



Gambar 3.11. *Flowchart* Fitur Data *Job History*

Sumber : [11]

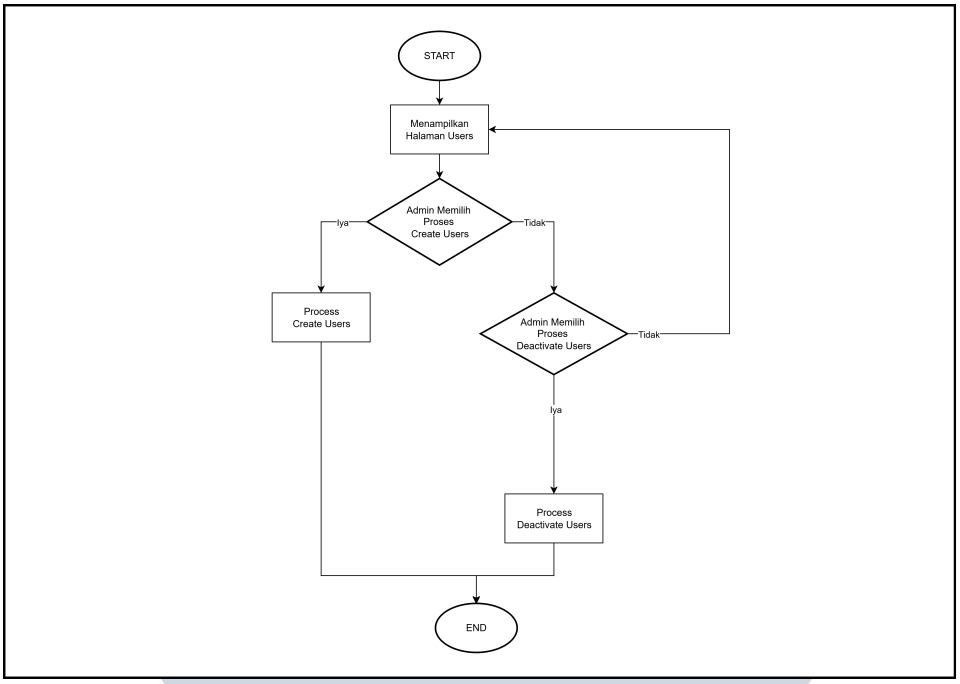
4. Flowchart Page Token



Gambar 3.12. *Flowchart* Fitur Data *Token*

Sumber : [11]

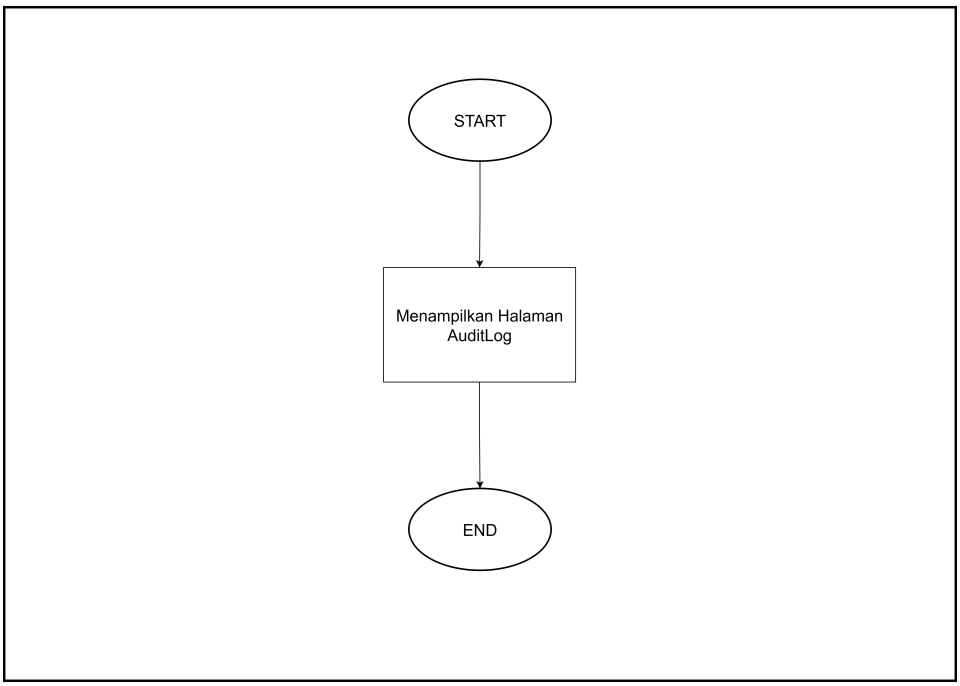
5. Flowchart Page Users



Gambar 3.13. *Flowchart* Fitur Data *Users*

Sumber : [11]

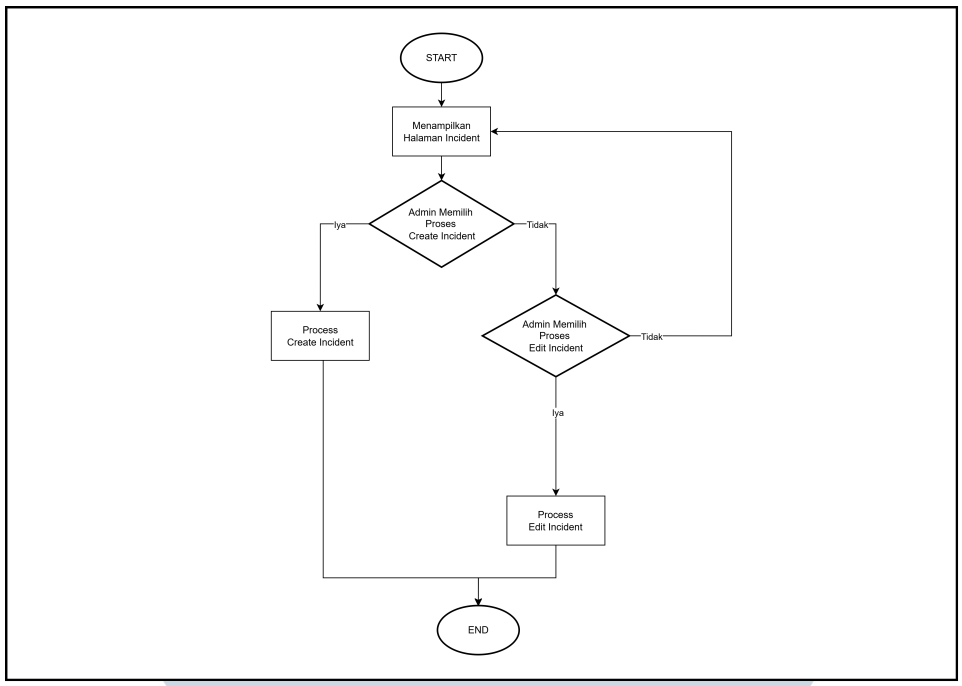
6. Flowchart Page Audit Log



Gambar 3.14. *Flowchart* Fitur Data *Audit Log*

Sumber : [11]

7. Flowchart Page Incident & Detail Incident Lainnya



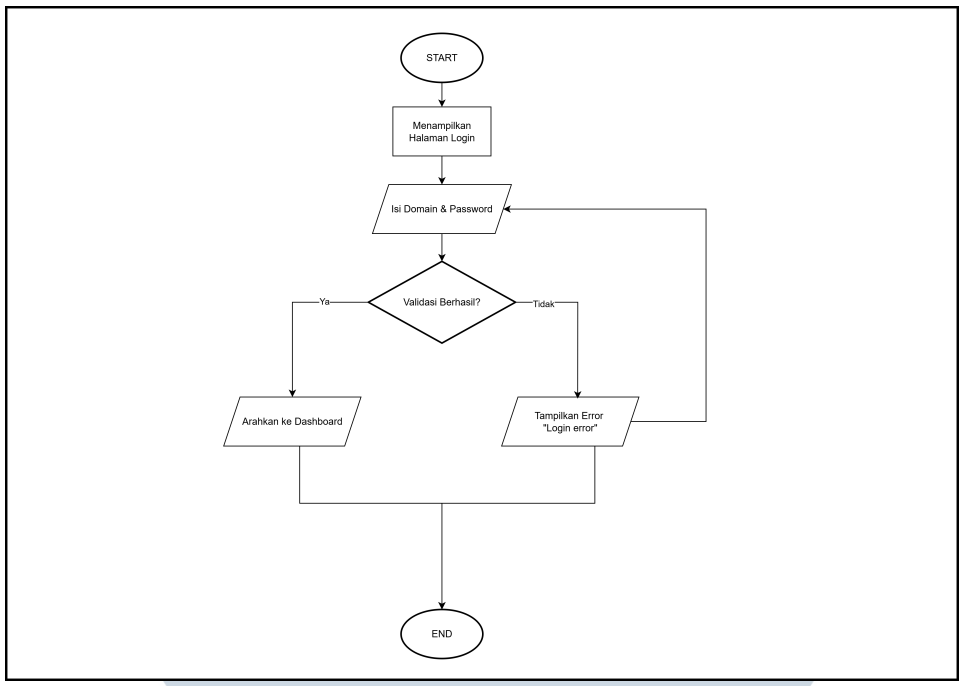
Gambar 3.15. *Flowchart* Fitur Data *Incident* & Detail Data *Incident* Lainnya

Sumber : [11]

D *Flowchart Fitur CMS Incident Report Application*

Setiap halaman dalam sistem memiliki fungsinya masing-masing, seperti melakukan penambahan, pengubahan, atau penghapusan data sesuai kebutuhan. Sementara itu, halaman publik hanya memiliki satu alur utama, yaitu proses login. Seluruh aktivitas spesifik yang digambarkan pada flowchart utama dapat dilihat pada daftar berikut.

1. *Flowchart Login*



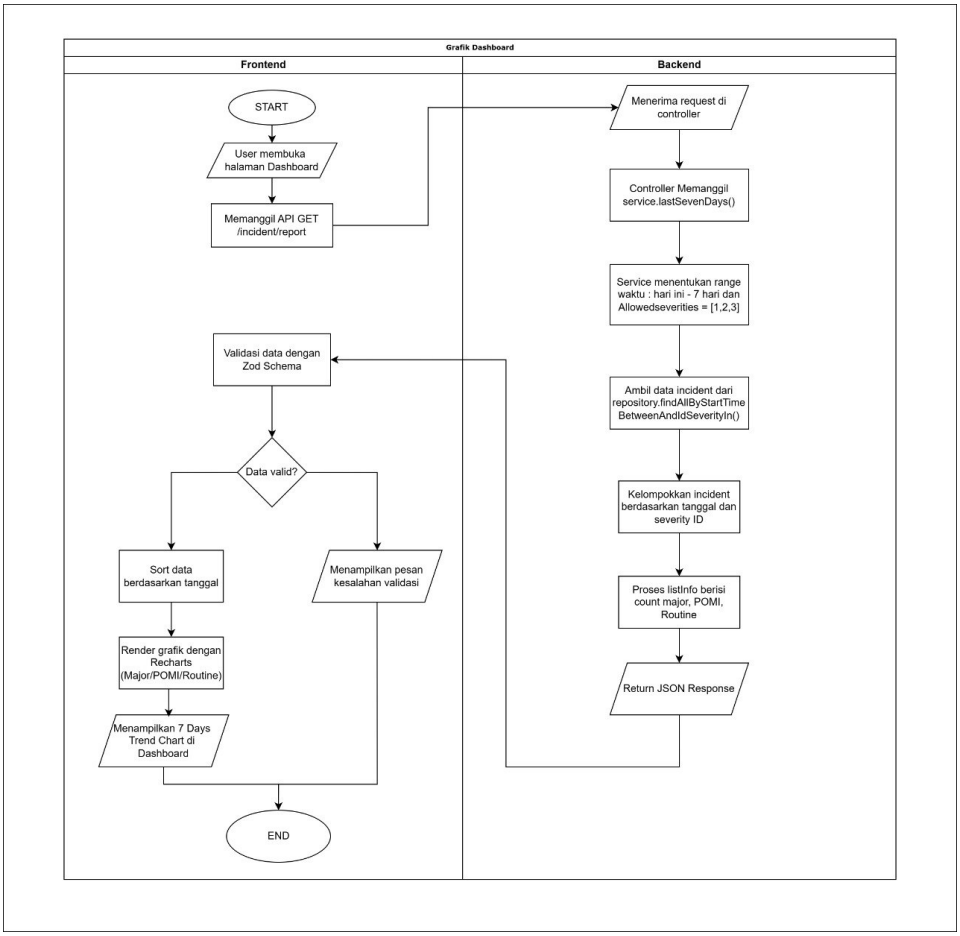
Gambar 3.16. *Flowchart Login*

Sumber : [11]

Flowchart pada Gambar 3.16 menunjukkan proses login pada *Content Management System Incident Report Application*. Proses dimulai dengan menampilkan halaman login kepada pengguna. Pengguna kemudian mengisi kolom domain dan password untuk melakukan autentikasi. Sistem akan melakukan proses validasi terhadap data yang dimasukkan.

Apabila validasi berhasil, pengguna akan diarahkan ke halaman Dashboard untuk mengakses fitur-fitur CMS. Namun, jika validasi gagal, sistem akan menampilkan pesan kesalahan *"Login error"* dan mengembalikan pengguna ke halaman login untuk mencoba kembali. *Flowchart* ini menggambarkan alur sederhana namun penting dalam memastikan keamanan akses aplikasi.

2. *Flowchart Graph 7 Days Incident Trend*



Gambar 3.17. *Flowchart* Proses Grafik *Dashboard* pada *CMS Incident Report*

Sumber : [11]

Gambar 3.17 menunjukkan *flowchart* proses penampilan grafik *7 Days Trend Chart* pada halaman *Dashboard* dalam *CMS Incident Report Application*. Flowchart ini menggambarkan alur komunikasi antara sisi *frontend* dan *backend* dalam menampilkan jumlah insiden berdasarkan kategori *Major, POMI*, dan *Routine* selama tujuh hari terakhir.

Proses dimulai ketika pengguna membuka halaman *dashboard*. *Frontend* kemudian mengirim permintaan (*request*) ke *endpoint API GET /incident/report* untuk mengambil data insiden dari *incident-service*. Permintaan ini diterima oleh *controller* pada sisi *backend*, yang kemudian memanggil *method service.lastSevenDays()*. Method tersebut menentukan rentang waktu tujuh hari terakhir serta daftar tingkat keparahan (*allowedSeverities*) yang akan dihitung.

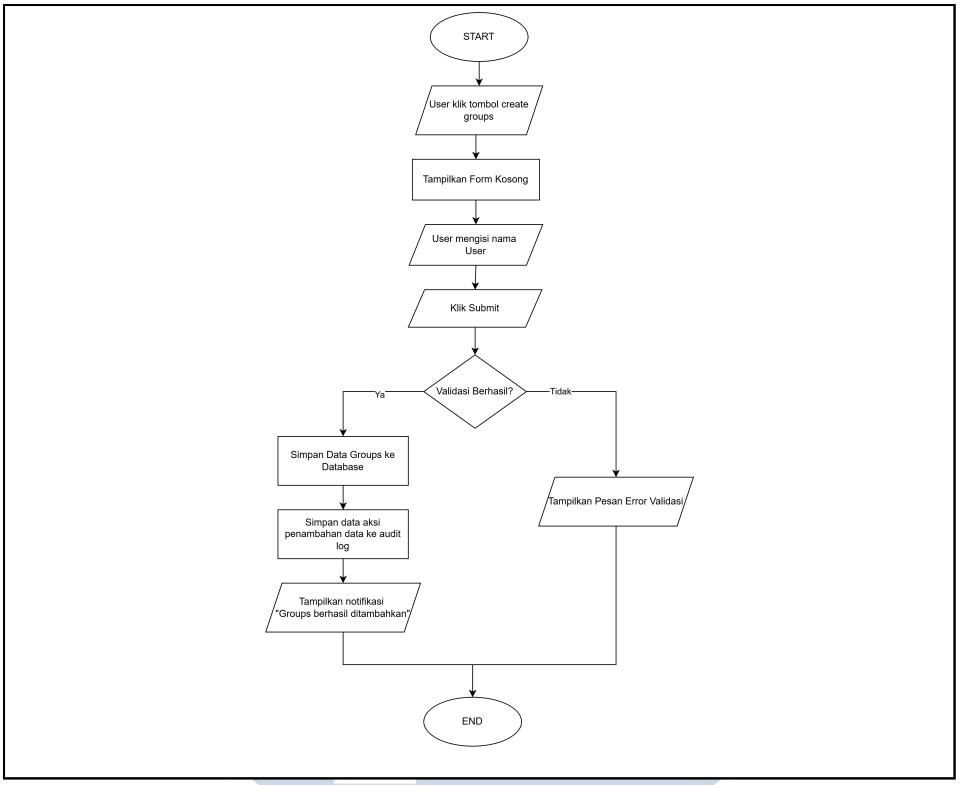
Selanjutnya, sistem mengambil data insiden

dari *database* melalui pemanggilan method *repository.findAllByStartTimeBetweenAndIdSeverityIn()*. Data yang berhasil diperoleh dikelompokkan berdasarkan tanggal kejadian dan tingkat keparahan (*severity ID*). Hasil pengelompokan kemudian diproses untuk menghasilkan *listInfo* yang berisi jumlah insiden harian untuk masing-masing kategori (*Major, POMI, Routine*). Data akhir dikembalikan ke *frontend* dalam format *JSON Response*.

Di sisi frontend, data yang diterima akan divalidasi menggunakan *Zod Schema* untuk memastikan struktur data sesuai dengan skema yang diharapkan. Jika validasi gagal, sistem akan menampilkan pesan kesalahan validasi kepada pengguna. Namun, jika data dinyatakan *valid*, sistem akan melakukan *sorting* berdasarkan tanggal, kemudian merender grafik batang menggunakan *Recharts*. Grafik ini menampilkan jumlah insiden per hari untuk masing-masing kategori selama tujuh hari terakhir.

Proses ini diakhiri dengan tampilan *7 Days Trend Chart* pada *dashboard* CMS, yang memberikan gambaran visual mengenai tren insiden dalam satu minggu terakhir.

3. *Flowchart Create Groups*

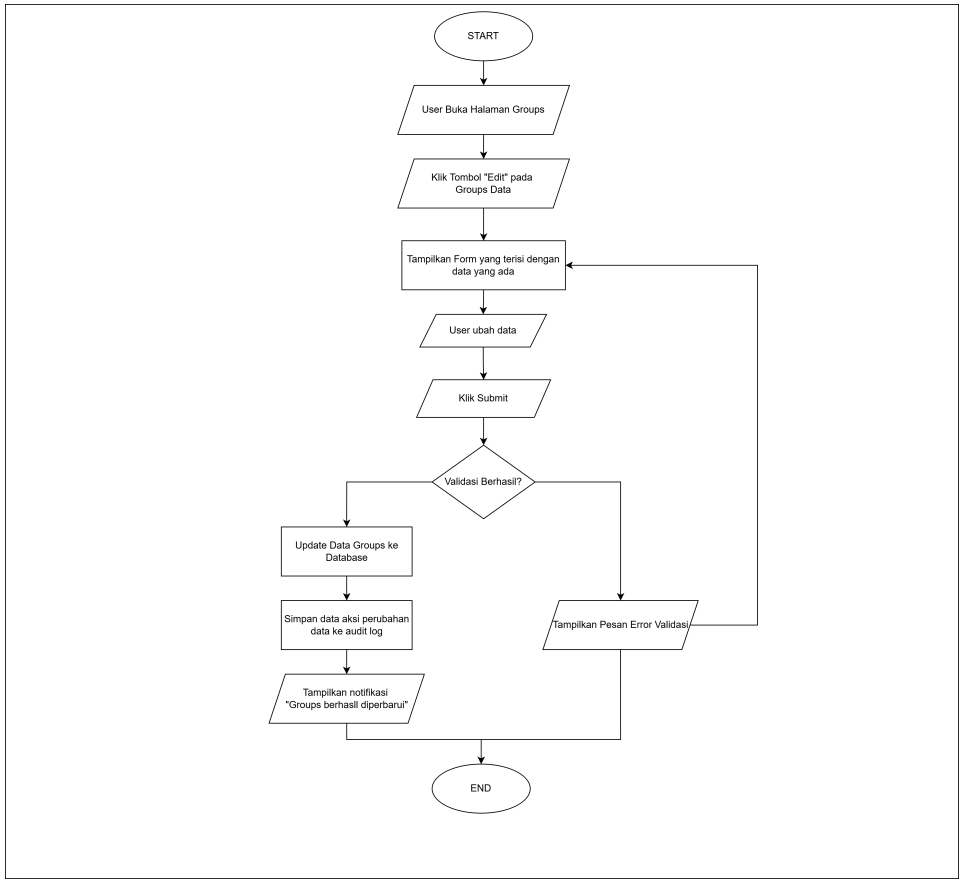


Gambar 3.18. *Flowchart Create Groups*

Sumber : [11]

Pada Gambar 3.18 menggambarkan proses ketika *user* membuat *group* baru pada sistem CMS. Proses dimulai saat pengguna menekan tombol "*Create Groups*", kemudian sistem menampilkan form kosong untuk diisi. Pengguna memasukkan nama *group* dan menekan tombol *submit*. Setelah itu, sistem melakukan validasi terhadap data yang dimasukkan. Jika validasi berhasil, *data group* akan disimpan ke dalam *database*, sistem mencatat aksi penambahan data ke *audit log*, dan menampilkan notifikasi bahwa group berhasil ditambahkan. Namun, jika validasi gagal, sistem akan menampilkan pesan *error* validasi sehingga pengguna dapat memperbaiki input yang salah sebelum mencoba kembali.

4. *Flowchart Edit Groups*

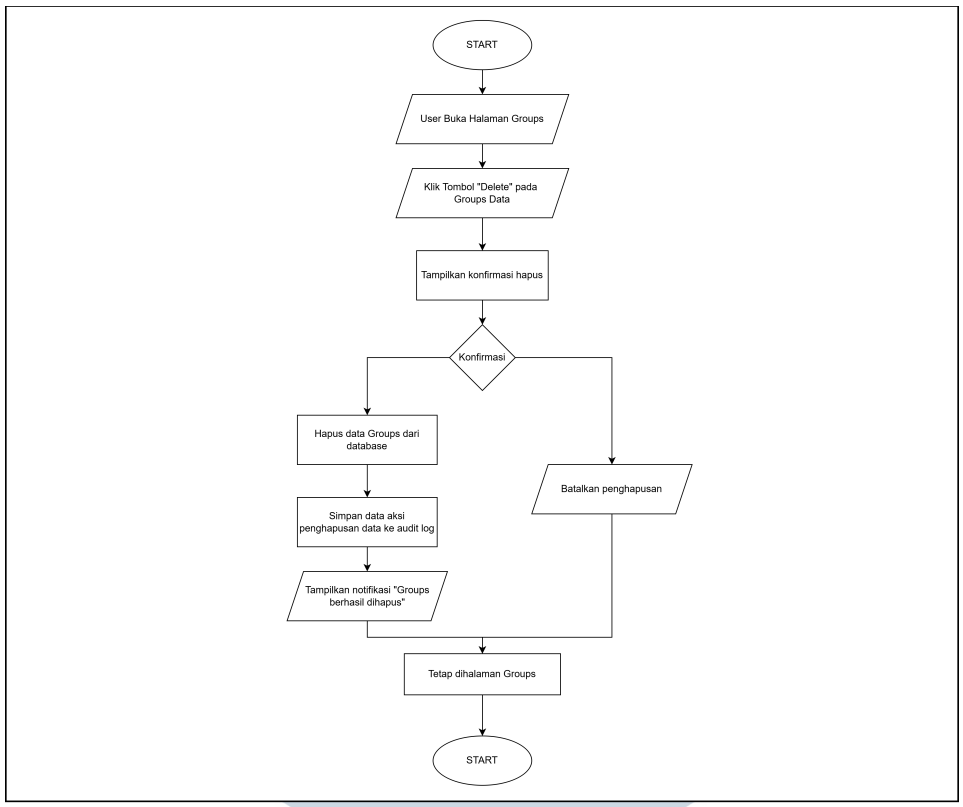


Gambar 3.19. *Flowchart Edit Groups*

Sumber : [11]

Pada Gambar 3.19 menggambarkan proses pengeditan *data group* pada sistem CMS. Proses dimulai ketika pengguna membuka halaman *groups*, lalu menekan tombol "*Edit*" pada data *group* yang ingin diubah. Sistem kemudian menampilkan form yang sudah terisi dengan data sebelumnya, dan pengguna dapat melakukan perubahan sesuai kebutuhan. Setelah itu, pengguna menekan tombol *submit* untuk menyimpan perubahan. Sistem akan melakukan validasi terhadap data yang dimasukkan. Jika validasi berhasil, *data group* akan diperbarui di *database*, aksi perubahan dicatat ke dalam *audit log*, dan sistem menampilkan notifikasi bahwa *group* berhasil diperbarui. Namun, jika validasi gagal, sistem akan menampilkan pesan *error* validasi agar pengguna dapat memperbaiki data yang salah sebelum menyimpan kembali.

5. *Flowchart Delete Groups*



Gambar 3.20. *Flowchart Delete Groups*

Sumber : [11]

Pada Gambar 3.20 menggambarkan proses penghapusan *data group* pada sistem CMS. Proses dimulai ketika pengguna membuka halaman *groups*, kemudian menekan tombol "*Delete*" pada data group yang ingin dihapus. Setelah itu, sistem menampilkan dialog konfirmasi untuk memastikan tindakan penghapusan. Jika pengguna mengonfirmasi, sistem akan menghapus *data group* dari *database*, mencatat aksi penghapusan ke dalam *audit log*, dan menampilkan notifikasi bahwa *group* berhasil dihapus. Namun, jika pengguna membatalkan konfirmasi, maka proses penghapusan tidak dilakukan dan pengguna tetap berada di halaman *groups*.

3.3.4 Implementasi Sistem

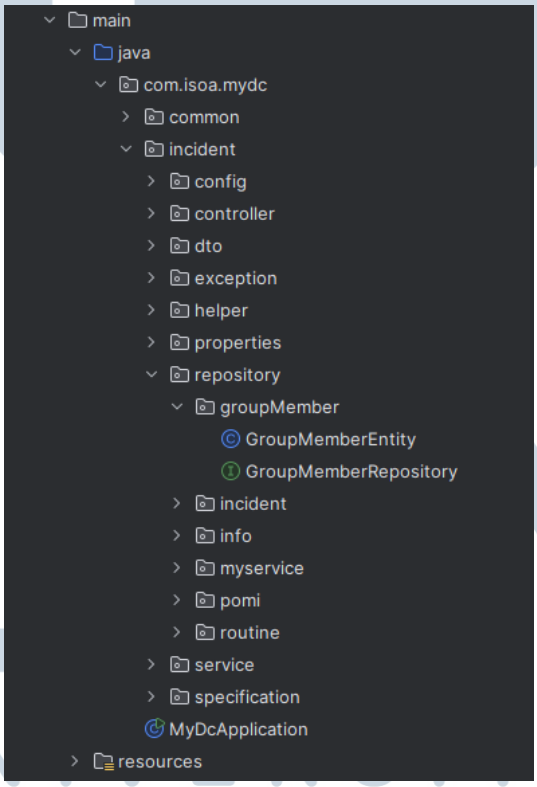
Setelah dilakukan perancangan alur sistem melalui flowchart pada setiap fitur utama, tahap berikutnya adalah implementasi kode pada sisi *backend* dan *frontend*. Bagian ini menjelaskan bagaimana rancangan sistem diubah menjadi aplikasi yang dapat berjalan secara fungsional. Implementasi dibagi menjadi dua

bagian utama, yaitu backend menggunakan Spring Boot dan frontend menggunakan Next.js yang saling terhubung melalui API Gateway.

A Implementasi *Backend (Spring Boot)*

Backend dari sistem *CMS Incident Report Application* dikembangkan menggunakan *framework Spring Boot* dengan menerapkan *layered architecture*. Pendekatan ini bertujuan untuk membuat struktur kode menjadi lebih terorganisir, mudah dipelihara, dan memisahkan tanggung jawab pada setiap lapisan secara jelas.

Struktur folder proyek mengikuti konvensi standar *Spring Boot* yang membagi setiap komponen berdasarkan fungsinya. Gambar 3.21 menunjukkan struktur folder dari modul incident pada sistem CMS.



Gambar 3.21. Struktur Folder Springboot Pada *CMS Incident Report Application*
Sumber : [11]

Struktur tersebut terdiri atas beberapa folder utama yang memiliki fungsi masing-masing :

1. *Controller*

   Berfungsi untuk menangani permintaan (*request*) dari sisi *frontend* dan

mengembalikan *response* yang sesuai. *class* di folder ini menjadi pintu masuk utama bagi pengguna untuk berinteraksi dengan sistem.

2. *Service*
   Menampung logika bisnis utama aplikasi, termasuk proses validasi, pengolahan data, serta pemanggilan

3. *Entity*
   Berisi kelas-kelas yang merepresentasikan struktur tabel di dalam *database*. Setiap *entity class* didefinisikan dengan anotasi seperti @*Entity* dan @*Table*, yang menandakan bahwa kelas tersebut akan dipetakan langsung ke tabel tertentu di *database*.

4. *Repository*
   Berisi *interface* yang berfungsi untuk melakukan interaksi langsung dengan database menggunakan *Spring Data JPA*.

5. DTO (*Data Transfer Object*)
   Digunakan sebagai wadah pertukaran data antara lapisan *controller* dan *service*. DTO memastikan data yang dikirim maupun diterima tetap terstruktur dan aman tanpa berinteraksi langsung dengan entitas *database*.

6. *Exception*
   Berisi *class* yang menangani berbagai jenis kesalahan (error handling) secara terpusat. Dengan adanya folder ini, aplikasi dapat menampilkan pesan kesalahan yang lebih informatif dan mudah dipahami pengguna.

7. *Specification*
   Digunakan untuk mendefinisikan kriteria pencarian atau filter yang kompleks di database menggunakan JPA Specification. Folder ini membantu membuat query dinamis tanpa menulis SQL secara manual.

Dengan pembagian seperti ini, pengembangan sistem menjadi lebih modular, terstruktur, dan mudah diperluas di masa mendatang. Setiap lapisan memiliki peran spesifik yang mendukung implementasi CMS Incident Report Application secara keseluruhan.

1. Controller
   Salah satu komponen utama dari sistem ini adalah *controller*, yang berfungsi sebagai penghubung antara pengguna dan sistem *backend*.

Controller menggunakan anotasi *@RestController* dan *@RequestMapping* untuk mendefinisikan *endpoint* API yang akan diakses oleh *frontend*.

Pada *CMS Incident Report Application*, contoh implementasi *controller* terdapat pada *CmsEditController*, yang menangani permintaan untuk memperbarui data group melalui *endpoint PUT /cms/groups/update*. *Controller* ini menerima objek *request* dari pengguna, kemudian meneruskan data tersebut ke lapisan *service* untuk diproses lebih lanjut.

```
@RestController
@RequestMapping("/cms")
public class CmsEditController {

    private final CmsEditService cmsEditService;

    @Autowired
    public CmsEditController(CmsEditService cmsEditService,
    AuditLogService auditLogService) {
        this.cmsEditService = cmsEditService;
        this.auditLogService = auditLogService;
    }

    @PutMapping("/groups/update")
    public ResponseEntity<ResponseSchema<Groups>>
    updateGroupName(
            @RequestBody GroupEditRequest request
    ) {
        Groups updatedGroup = cmsEditService.updateGroupName(
            request.getId(),
            request.getNewname(),
            request.getdomain()
        );

        return ResponseEntity.ok(
            ResponseSchema.<Groups>builder()
                .errorSchema(ErrorSchema.success())
                .outputSchema(updatedGroup)
                .build()
        );
    }
}
```

Kode 3.1: Contoh Implementasi *Controller* pada *CMS Incident Report*

Pada Kode 3.1, menunjukkan bahwa *controller* hanya berperan sebagai

pengatur alur komunikasi. Logika utama tidak ditempatkan di sini, melainkan dikelola oleh *service* agar kode tetap bersih dan terpisah dengan baik.

2. Service
   Lapisan *service* merupakan inti dari logika bisnis sistem. Pada lapisan ini, semua proses seperti validasi, manipulasi data, hingga pemanggilan *repository* dilakukan secara terpisah dari *controller*. Salah satu contoh implementasi *service* dapat dilihat pada *CmsGetServiceImpl*, yang berfungsi untuk menampilkan data *incident* berdasarkan parameter pencarian tertentu seperti waktu, *severity*, dan status penyelesaian.

```
@Service
@Slf4j
public class CmsGetServiceImpl implements CmsGetService {
    private final IncidentRepository repository;

    public CmsGetServiceImpl(IncidentRepository repository) {
        this.repository = repository;
    }

    @Override
    public Page<IncidentEntity> getAllIncidents(
            LocalDate startTimeFromParam, LocalDate
    startTimeToParam,
            LocalDate endTimeFromParam, LocalDate
    endTimeToParam,
            String search,
            Integer idSeverity,
            Pageable pageable,
            Boolean solved
    ) {
        LocalDateTime startTimeFrom = (startTimeFromParam !=
    null)
            ? startTimeFromParam.atStartOfDay() : null;
        LocalDateTime startTimeTo = (startTimeToParam != null
    )
            ? startTimeToParam.atTime(LocalTime.MAX) : null;
        LocalDateTime endTimeFrom = (endTimeFromParam != null
    )
            ? endTimeFromParam.atStartOfDay() : null;
        LocalDateTime endTimeTo = (endTimeToParam != null)
            ? endTimeToParam.atTime(LocalTime.MAX) : null;

        Specification<IncidentEntity> spec =
```

```
    IncidentSpecifications.withDynamicQuery(
                startTimeFrom, startTimeTo, endTimeFrom,
    endTimeTo, search, idSeverity, solved
        );

        return repository.findAll(spec, pageable);
    }
}
```

Kode 3.2: Contoh Implementasi Service pada *CMS Incident Report Application*

Kode 3.2 menunjukkan bagaimana lapisan *service* memanfaatkan *Spring Data JPA Specification* untuk membangun *query* pencarian secara dinamis berdasarkan parameter yang diterima. Dengan pendekatan ini, sistem dapat menampilkan data sesuai filter pengguna tanpa perlu menulis *query SQL* secara manual.

Setiap operasi, seperti pengambilan, pembaruan, maupun penghapusan data, dikelola melalui metode yang terpisah. Hal ini membuat proses *debugging* lebih mudah serta mengurangi risiko terjadinya kesalahan selama pengembangan.

3. *Entity*

   Lapisan *entity* berfungsi untuk merepresentasikan struktur tabel yang ada di dalam *database*. *Entity* menjadi fondasi utama bagi operasi *CRUD (Create, Read, Update, Delete)* yang dijalankan oleh sistem melalui *repository*.

   Salah satu contoh implementasi *entity* dalam *CMS Incident Report Application* adalah *GroupMemberEntity*, yang merepresentasikan data anggota *group* di dalam sistem. Kelas ini digunakan untuk menyimpan informasi seperti identitas pengguna, status aktif, serta waktu pembuatan dan pembaruan data.

```
@Data
@Entity
@Table(name = "GROUP_MEMBER")
public class GroupMemberEntity {

@Id
private String idGroupMember;
private String domain;
private String idGroup;
private boolean isActive;
```

```
11 private LocalDateTime updatedOn;
12 private String updatedBy;
13 private LocalDateTime createdOn;
14 private String createdBy;
15 private boolean isMuted;
16
17
18 }
```

Kode 3.3: Contoh Implementasi Entity pada *CMS Incident Report Application*

Kode 3.3 menunjukkan bahwa kelas *GroupMemberEntity* dipetakan ke tabel *GROUP_MEMBER* di dalam *database*. Setiap atribut di dalam kelas ini merepresentasikan kolom pada tabel yang bersesuaian. Lapisan *entity* juga membantu memastikan konsistensi antara struktur *database* dan model data yang digunakan dalam aplikasi.

4. *Repository*

   Lapisan *repository* bertanggung jawab terhadap proses komunikasi langsung dengan *database*. Pada *CMS Incident Report Application, repository* menggunakan *Spring Data JPA*, yang menyediakan berbagai operasi dasar seperti penyimpanan, pembaruan, pencarian, dan penghapusan data tanpa perlu menulis *query SQL* secara manual.

   Salah satu contoh implementasi *repository* terdapat pada *RoutineRepository*, yang digunakan untuk mengelola data *routine* dalam sistem. Repository ini tidak hanya menggunakan *JpaRepository*, tetapi juga mengimplementasikan *JpaSpecificationExecutor* agar dapat menjalankan *query* dinamis berdasarkan spesifikasi tertentu.

```
1 @Repository
2 public interface RoutineRepository extends JpaRepository<
      RoutineEntity, String>,
3 JpaSpecificationExecutor<RoutineEntity> {
4
5 List<RoutineEntity> findAllByNumberOrderByTimeDesc(String
      number);
6
7
8 }
```

Kode 3.4: Contoh Implementasi Repository pada *CMS Incident Report Application*

Kode 3.4 menunjukkan bahwa *RoutineRepository* memanfaatkan konvensi penamaan metode (*method naming convention*) dari *Spring Data*

JPA. Metode *findFirstByNumberOrderByTimeDesc*() digunakan untuk mengambil data *routine* terbaru berdasarkan nomor tertentu, sedangkan *findAllByNumberOrderByTimeDesc*() digunakan untuk menampilkan seluruh data dengan urutan waktu dari yang terbaru hingga terlama.

Pendekatan ini membuat kode menjadi lebih ringkas dan mudah dipahami, karena *Spring Boot* secara otomatis menghasilkan *query SQL* berdasarkan nama metode tanpa perlu menulis *query* secara eksplisit.

5. DTO *(Data Transfer Object)*

   Untuk memisahkan struktur data yang diterima dari sisi *frontend* dengan data yang dikembalikan sebagai *response*, *CMS Incident Report Application* menggunakan beberapa kelas DTO (*Data Transfer Object*). DTO berfungsi sebagai objek perantara yang menyederhanakan dan mengamankan proses pertukaran data antara *controller* dan *service*, serta menghindari pemetaan langsung ke entitas *database*.

   Salah satu DTO yang digunakan untuk menangani permintaan dari pengguna adalah *UserDeleteRequest*. Kelas ini berperan dalam proses penghapusan atau perubahan status pengguna, dengan memuat informasi seperti daftar domain pengguna yang akan dihapus, kondisi perubahan, serta domain pengguna yang diproses. DTO ini juga menggunakan anotasi *@JsonNaming(PropertyNamingStrategies.SnakeCaseStrategy.class)* agar penamaan atribut pada format JSON tetap konsisten dengan standar penulisan *snake_case*. Contoh implementasi kelas UserDeleteRequest dapat dilihat pada Kode 3.5 berikut:

```
@Builder
@Data
@NoArgsConstructor
@AllArgsConstructor
@JsonNaming(PropertyNamingStrategies.SnakeCaseStrategy.class)
public class UserDeleteRequest {
private List<String> domains;
private Boolean condition;
private String domain;
}
```

Kode 3.5: Contoh implementasi Request DTO pada CMS Incident Report Application

Untuk contoh respons, *CMS Incident Report Application* menggunakan kelas *UserDeleteResponse*. Kelas ini digunakan untuk mengembalikan hasil dari

proses penghapusan atau pembaruan status pengguna yang telah dilakukan oleh sistem. Data yang dikembalikan berupa daftar entitas pengguna yang berhasil diubah statusnya menjadi aktif atau tidak aktif. Contoh implementasi *UserDeleteResponse* dapat dilihat pada Kode 3.6 berikut:

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class UserDeleteResponse {
private List<UsersEntity> ActiveInactiveUsers;
}
```

Kode 3.6: Contoh implementasi Response DTO pada CMS Incident Report Application

Dengan penggunaan DTO, *CMS Incident Report Application* dapat menjaga agar struktur data yang dikirim dan diterima tetap konsisten, mudah dibaca, dan aman dari pemetaan langsung ke entitas *database*. Selain itu, pendekatan ini juga mempermudah proses pengujian dan pemeliharaan kode, karena setiap alur data memiliki objek perantara yang terdefinisi dengan jelas.

6. *Exception*

   Untuk memastikan sistem dapat menangani berbagai kondisi kesalahan secara konsisten dan terstruktur, *CMS Incident Report Application* menerapkan mekanisme *exception handling* menggunakan anotasi *@ControllerAdvice*. Pendekatan ini memungkinkan semua kesalahan yang terjadi di dalam aplikasi, khususnya pada lapisan *service* atau *controller*, dapat ditangani secara terpusat dan dikembalikan dalam format respons yang seragam.

   Salah satu contoh implementasi *exception handler* terdapat pada kelas *IncidentExceptionHandler*. Kelas ini digunakan untuk menangani kesalahan ketika data *incident* yang diminta tidak ditemukan di dalam sistem. Dengan memanfaatkan anotasi *@ExceptionHandler*, sistem dapat mengidentifikasi jenis pengecualian tertentu dan memberikan respons yang sesuai tanpa menyebabkan aplikasi berhenti secara tiba-tiba. Contoh implementasi dapat dilihat pada Kode 3.7 berikut:

```
@ControllerAdvice
@Slf4j
public class IncidentExceptionHandler {

```

```
@Autowired
private MessageProperties properties;

@ExceptionHandler(IncidentNotFoundException.class)
public ResponseEntity<ResponseSchema<String>>
    incidentNotFoundException(IncidentNotFoundException ex) {
    log.error("Terjadi kesalahan | IncidentNotFoundException
    : {}", ex.getMessage());
    return ResponseEntity.status(HttpStatus.NOT_FOUND)
        .body(ResponseSchema.<String>builder()
            .errorSchema(ErrorSchema.builder()
                .errorCode(properties.getProps().get("
    incident.not-found.code"))
                .errorMessage(properties.getProps().get("
    incident.not-found.msg"))
                .build()
            )
            .outputSchema(ex.getMessage())
            .build()
        );
}


}
```

Kode 3.7: Contoh implementasi Exception Handler pada *CMS Incident Report Application*

Kode 3.7 menunjukkan bahwa *IncidentExceptionHandler* bertugas menangkap pengecualian *IncidentNotFoundException* dan mengubahnya menjadi respons *HTTP* dengan status *404 Not Found*. Pesan kesalahan yang dikembalikan diambil dari file konfigurasi *MessageProperties*, sehingga memudahkan pengelolaan dan standarisasi pesan error di seluruh sistem.

Dengan adanya mekanisme ini, *CMS Incident Report Application* dapat memberikan tanggapan kesalahan yang lebih informatif dan mudah dipahami oleh pengguna maupun *developer*.

7. *Specification*

   Lapisan *specification* digunakan untuk membangun *query* dinamis pada *database* menggunakan *JPA Specification*. Pendekatan ini memungkinkan sistem melakukan pencarian atau penyaringan data dengan berbagai kombinasi parameter tanpa perlu menulis *query SQL* secara manual. Dengan

spesifikasi dinamis, fitur pencarian di *CMS Incident Report Application* menjadi lebih fleksibel, efisien, dan mudah dikembangkan sesuai kebutuhan.

Salah satu implementasi *specification* terdapat pada kelas *AuditLogSpecifications*, yang digunakan untuk melakukan pencarian data log aktivitas berdasarkan waktu maupun kata kunci tertentu. Kelas ini membentuk kriteria pencarian menggunakan *CriteriaBuilder* dari JPA, kemudian menggabungkannya secara dinamis sesuai parameter yang diterima. Contoh implementasi dapat dilihat pada Kode 3.8 berikut:

```
1  public class AuditLogSpecifications {
2
3  public static Specification<AuditLogEntity> withDynamicQuery(
4          LocalDateTime TimeFrom,
5          LocalDateTime TimeTo,
6          String search) {
7      return (root, query, criteriaBuilder) -> {
8
9          List<Predicate> predicates = new ArrayList<>();
10
11         if (TimeFrom != null) {
12             predicates.add(criteriaBuilder.
    greaterThanOrEqualTo(root.get("time"), TimeFrom));
13         }
14         if (TimeTo != null) {
15             predicates.add(criteriaBuilder.lessThanOrEqualTo(
    root.get("time"), TimeTo));
16         }
17
18         if (StringUtils.hasText(search)) {
19             String searchPattern = "%" + search.toLowerCase()
     + "%";
20
21             Predicate searchPredicate = criteriaBuilder.or(
22                     criteriaBuilder.like(criteriaBuilder.
    lower(root.get("domain").as(String.class)), searchPattern)
    ,
23                     criteriaBuilder.like(criteriaBuilder.
    lower(root.get("action")), searchPattern)
24             );
25             predicates.add(searchPredicate);
26         }
27         return criteriaBuilder.and(predicates.toArray(new
```

```
        Predicate[0]));
28      };
29  }
30
31
32  }
```

Kode 3.8: Contoh implementasi *Specification* pada *CMS Incident Report Application*

Kode 3.8 menunjukkan bahwa metode *withDynamicQuery*() akan membangun *predicate* secara otomatis berdasarkan parameter waktu dan teks pencarian yang diberikan. Jika pengguna memasukkan rentang waktu tertentu, sistem akan menyaring data berdasarkan kolom time. Sementara jika parameter search diisi, sistem akan mencocokkan kata kunci tersebut dengan kolom domain atau action.

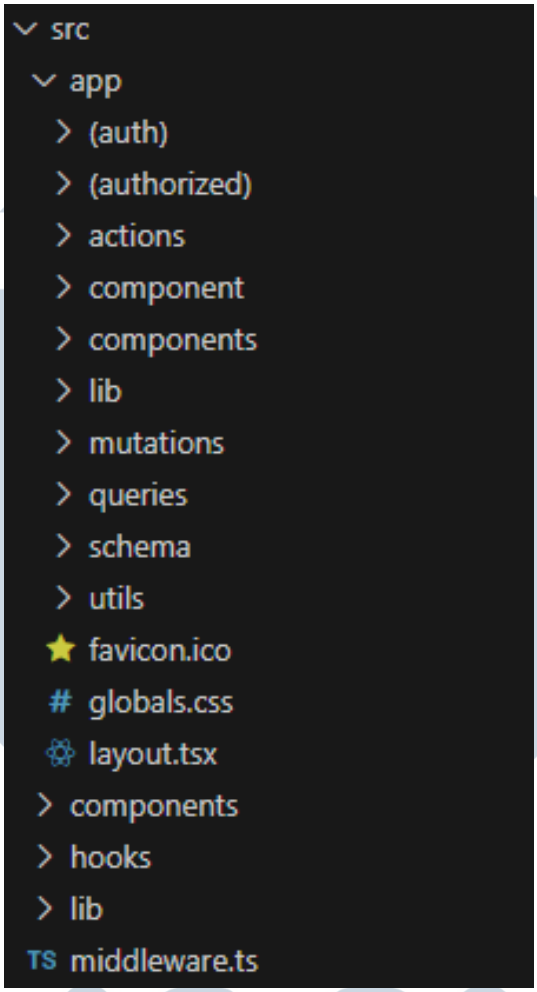
Dengan penerapan *JPA Specification* seperti ini, *CMS Incident Report Application* dapat melakukan pencarian data yang kompleks secara efisien dan tetap mempertahankan struktur kode yang bersih serta mudah dipelihara.

Seluruh komponen pada sisi backend mulai dari *Controller, Service, Repository*, hingga *Specification* saling berinteraksi untuk memastikan proses pengolahan data berjalan optimal. Setelah implementasi pada sisi *backend* selesai, tahap berikutnya adalah pengembangan *frontend* menggunakan *Next.js*, yang berperan sebagai antarmuka utama bagi pengguna dalam mengakses fitur *CMS Incident Report Application*.

B Implementasi *Frontend (Next.js)*

Pada sisi *frontend*, *CMS Incident Report Application* dikembangkan menggunakan *framework Next.js* yang mendukung *server-side rendering* dan pengelolaan rute berbasis file (*file-based routing*). *Next.js* dipilih karena kemampuannya dalam menggabungkan fleksibilitas *React* dengan performa tinggi, serta kemudahan integrasi terhadap API dari *backend Spring Boot*.

Struktur utama proyek frontend dapat dilihat pada Gambar 3.22.



Gambar 3.22. Struktur Folder Next.js Pada *CMS Incident Report Application*

Sumber : [11]

Gambar tersebut menunjukkan bahwa direktori *src/app* menjadi pusat utama pengelolaan halaman dan logika aplikasi. Setiap folder memiliki tanggung jawab yang berbeda, di antaranya:

1. *Auth*
   berisi halaman *login* dan autentikasi pengguna
2. *authorized*
   berisi halaman yang hanya dapat diakses setelah pengguna berhasil *login*
3. *actions*
   berisi fungsi *server actions* yang digunakan untuk memproses data secara langsung di *server*

4. *queries* dan *mutations*
   berfungsi sebagai pengelola proses pengambilan dan pengiriman data menggunakan *TanStack React Query*

5. *schema*
   menyimpan definisi validasi data menggunakan library *Zod*

6. *utils* dan *lib*
   berisi fungsi bantu dan konfigurasi umum yang digunakan di seluruh aplikasi.

Pembagian struktur seperti ini menjadikan proyek lebih modular dan mudah dikelola, karena setiap bagian memiliki tanggung jawab yang jelas sesuai dengan prinsip *separation of concerns*.

1. *Actions*
   Folder *actions* berisi kumpulan fungsi yang dijalankan di sisi server (*server actions*), yang berfungsi sebagai penghubung antara antarmuka pengguna dengan *API backend*. Setiap fungsi yang didefinisikan di dalam folder ini bertanggung jawab untuk mengirim permintaan ke *backend Spring Boot*, menerima respons, serta menangani hasil atau pesan kesalahan yang terjadi selama proses komunikasi.

   Salah satu contoh implementasi fungsi pada folder *actions* adalah *createGroups*, yang digunakan untuk menambahkan data *group baru* ke dalam sistem. Fungsi ini memanggil *endpoint /cms/create/groups/create* dari backend dengan mengirimkan data berupa *newname* dan *domain*. Contoh implementasi kode dapat dilihat pada Kode 3.9 berikut:

```
export const createGroups = async (
request: CreateGroupInput
): Promise<ResponseType<GroupCreateResponseType>> => {
try {
const response = await notifbeapi
.post("/cms/create/groups/create", {
newname: request.newname,
domain: request.domain,
})
.then((res) => {
const createGroupsResponse = GeneralResponseSchema(
GroupCreateResponseSchema
).safeParse(res.data);
```

```
14      return { data: createGroupsResponse.data?.output_schema
        };
15    });
16  return response;
17
18
19  } catch {
20  return {
21  error: Group with name ${request.newname} already exists,
22  };
23  }
24  };
```

Kode 3.9: Contoh implementasi *server action* pada *CMS Incident Report Application*

Berdasarkan Kode 3.9, fungsi *createGroups* menggunakan *API client* untuk mengirim data ke *endpoint backend*. Setelah permintaan dikirim, respons yang diterima akan divalidasi menggunakan *schema parser (GeneralResponseSchema* dan *GroupCreateResponseSchema*) untuk memastikan struktur data yang diterima sesuai standar sistem. Apabila proses pembuatan *group* berhasil, data hasil respons dikembalikan melalui properti *output_schema*. Namun, jika terjadi kesalahan misalnya nama group sudah terdaftar sebelumnya maka fungsi akan mengembalikan pesan kesalahan yang sesuai.

Pendekatan ini membantu menjaga agar proses komunikasi antara *frontend* dan *backend* tetap aman, terstruktur, dan mudah untuk di-*debug*.

2. *Schema*

   Folder *schema* digunakan untuk mendefinisikan struktur data dan melakukan validasi *input* sebelum dikirim ke *backend*, agar data yang diterima sesuai dengan format yang diharapkan sistem dan meminimalkan kesalahan saat komunikasi dengan API. Dalam *CMS Incident Report Application*, setiap *form input* memiliki *schema* yang didefinisikan menggunakan *library Zod*, yang memungkinkan pembuatan aturan validasi secara deklaratif serta menghasilkan *TypeScript type inference* secara otomatis untuk menjaga konsistensi dan keamanan data di seluruh aplikasi.

   Salah satu contoh implementasi *schema* terdapat pada *auditLoginSchema*, yang digunakan untuk memvalidasi data *login* pengguna sebelum dikirim ke *backend*. *Schema* ini memastikan bahwa setiap pengguna harus mengirimkan

data *domain* dan ip yang valid. Contoh implementasinya dapat dilihat pada Kode 3.10 berikut:

```
import { z } from "zod";

export const auditLoginSchema = z.object({
domain: z.string().min(1, "domain is required").max(100, "
    udomain is too long"),
ip: z.string().min(1, "ip is required").max(100, "ip is too
    long"),
});

export type auditLoginInput = z.infer<typeof auditLoginSchema
    >;
```

Kode 3.10: Contoh implementasi *schema* validasi data *login*

Berdasarkan Kode 3.10, *auditLoginSchema* memastikan setiap *field* memiliki panjang minimal satu karakter dan maksimal seratus karakter. Selain itu, tipe *auditLoginInput* dihasilkan secara otomatis dari *schema* tersebut menggunakan *z.infer*, sehingga tipe data antara *frontend* dan *backend* selalu konsisten.

Dengan pendekatan seperti ini, setiap proses *input* yang dilakukan pengguna akan terlebih dahulu divalidasi di sisi *frontend* sebelum dikirim ke *server*. Hal ini membantu mengurangi potensi *error*, meningkatkan keandalan sistem, dan menjaga integritas data di seluruh alur aplikasi.

3. *Queries*
   Folder *queries* digunakan untuk mengelola proses pengambilan data (*fetching*) dari *backend* menggunakan *library TanStack React Query*. Pendekatan ini memanfaatkan konsep *caching* dan *state synchronization*, sehingga data yang telah dimuat dapat digunakan kembali tanpa harus melakukan permintaan ulang ke *server*. Dengan demikian, performa aplikasi menjadi lebih efisien dan responsif.

   Salah satu contoh implementasi pada folder *queries* adalah *hook useAuditLogData*, yang berfungsi untuk mengambil data aktivitas log pengguna berdasarkan parameter filter tertentu. Implementasi kode dapat dilihat pada Kode 3.11 berikut:

```
"use client";

```

```
import { useQuery } from "@tanstack/react-query";

export function useAuditLogData(filters: GetAuditLogFilters)
    {
return useQuery<AuditLogResponse>({
queryKey: ["auditlog", filters],
queryFn: () => getAuditLogdata(filters),
staleTime: 1000 * 60 * 2,
});
}
```

Kode 3.11: Contoh implementasi *query data audit log*

Berdasarkan Kode 3.11, fungsi *useAuditLogData* memanfaatkan *hook useQuery* dari *TanStack React Query* untuk mengambil data dari *backend*. Parameter *queryKey* digunakan sebagai identitas unik bagi *cache* agar data dapat dikelola dan disegarkan secara otomatis oleh *React Query*. Sementara itu, *queryFn* berisi fungsi yang akan dijalankan untuk melakukan permintaan data ke *endpoint backend*.

Dengan penerapan ini, proses pengambilan data menjadi lebih efisien, mengurangi beban *server*, serta memberikan pengalaman pengguna yang lebih cepat dan responsif.

4. *Mutations*

   Folder *mutations* digunakan untuk menangani proses perubahan data (*write operations*) di sisi *backend*, seperti menambah, memperbarui, maupun menghapus data. Berbeda dengan *queries* yang berfokus pada pengambilan data, *mutations* berfungsi untuk melakukan operasi yang memodifikasi data dan kemudian memperbarui *state* aplikasi agar tetap konsisten dengan data di *server*.

   Salah satu contoh implementasi *mutation* pada *CMS Incident Report Application* adalah *hook useRemoveGroup*, yang digunakan untuk menghapus data *group* dari sistem. Contoh implementasi kode dapat dilihat pada Kode 3.12 berikut:

```
"use client";

import { useMutation } from "@tanstack/react-query";

export const useRemoveGroup = () => {
return useMutation({
```

```
7 mutationFn: (request: GroupDeleteRequestType) => deleteGroup(
    request),
8 });
9 };
```

Kode 3.12: Contoh implementasi *mutation delete data group*

Berdasarkan Kode 3.12, *hook useRemoveGroup* menggunakan *useMutation* untuk menjalankan fungsi *deleteGroup*, yang berfungsi mengirimkan permintaan penghapusan data *group* ke *backend*. Fungsi ini menerima parameter *request* bertipe *GroupDeleteRequestType*, yang berisi informasi group yang akan dihapus dari sistem. Ketika proses *mutation* berhasil dijalankan, *React Query* akan secara otomatis memperbarui *cache* terkait agar tampilan data di antarmuka tetap sinkron dengan data terbaru di server.

Melalui penerapan *queries* dan *mutations* ini, *CMS Incident Report Application* dapat menjaga konsistensi data secara *realtime*, meningkatkan efisiensi komunikasi antara *frontend* dan *backend*, serta menyederhanakan pengelolaan status data di sisi *frontend*.

3.3.5 Hasil Implementasi Sistem

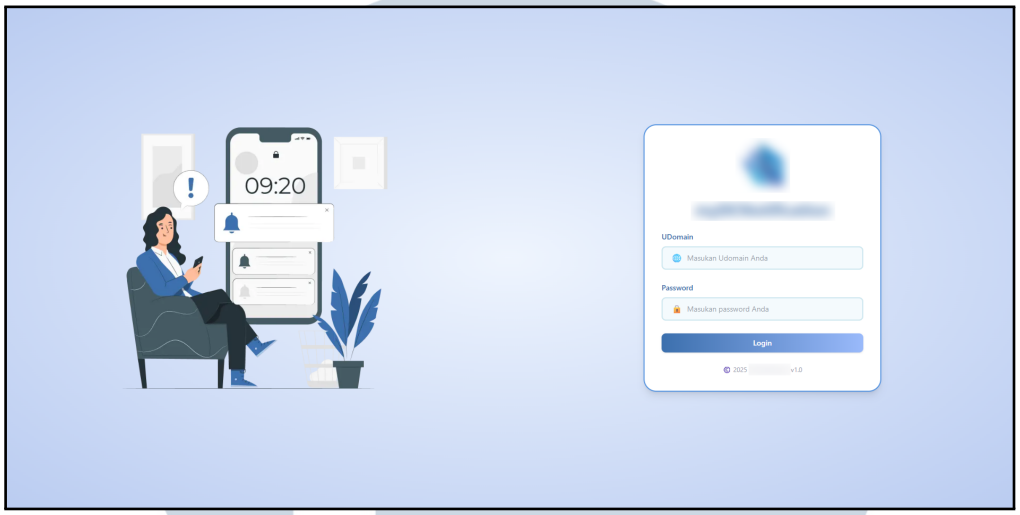
Setelah proses implementasi sistem *CMS Incident Report Application* selesai dilakukan, tahap selanjutnya adalah melakukan pengujian untuk memastikan bahwa seluruh komponen aplikasi dapat berjalan dengan baik dan saling terhubung. Sistem ini dikembangkan menggunakan *framework Next.js* pada sisi *frontend* dan *Spring Boot* pada sisi *backend*, yang berperan dalam menangani proses autentikasi, manajemen data, serta komunikasi antar layanan.

Setiap halaman dalam sistem memiliki fungsi yang saling berkaitan, mulai dari proses *login* pengguna hingga pengelolaan data notifikasi. Bagian berikut menjelaskan hasil implementasi antarmuka sistem berdasarkan fungsi utama yang terdapat dalam aplikasi.

A Halaman *Login*

Halaman login merupakan komponen awal yang digunakan pengguna untuk mengakses sistem CMS *Incident Report*. Pada halaman ini, pengguna diminta memasukkan *Domain* dan *Password* sebagai kredensial autentikasi. Proses *login* ini bertujuan memastikan bahwa hanya pengguna yang memiliki hak akses valid yang

dapat masuk ke dalam sistem. Tampilan antarmuka halaman *login* ditunjukkan pada Gambar 3.23.



Gambar 3.23. Halaman *Login*

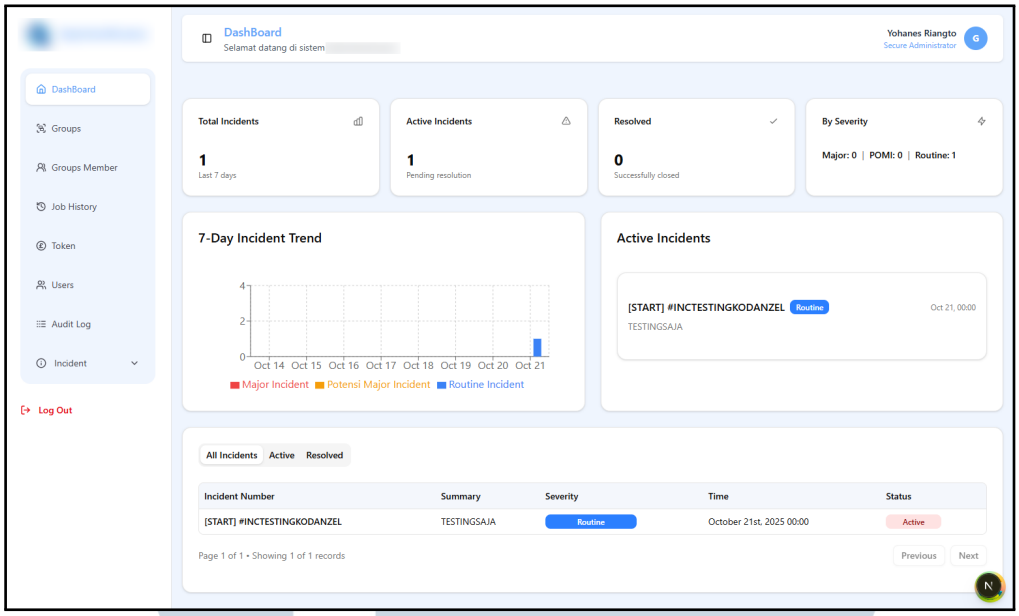
Setelah pengguna mengisi kedua kolom tersebut dan menekan tombol *Login*, sistem akan memproses data melalui fungsi autentikasi yang terhubung dengan *backend*. Validasi data dilakukan menggunakan *schema* validasi *Zod* untuk memastikan bahwa format *input* sesuai dengan ketentuan. Selanjutnya, data dikirim ke *server* menggunakan mekanisme *mutation React Query*, yang menangani proses komunikasi asinkron antara *frontend* dan *backend*.

Jika autentikasi berhasil, pengguna akan diarahkan menuju *dashboard*. Sebaliknya, jika data yang dimasukkan tidak *valid*, sistem akan menampilkan pesan kesalahan agar pengguna dapat memperbaiki *input* sebelum mencoba kembali.

B Halaman *Dashboard*

Setelah pengguna berhasil melewati proses *login*, sistem akan mengarahkan ke halaman *Dashboard*, yang berfungsi sebagai tampilan utama sekaligus pusat informasi dari sistem *CMS Incident Report*. Halaman ini menampilkan ringkasan (*overview*) kondisi terkini dari aktivitas notifikasi dan *insident* yang terjadi dalam sistem.

Tampilan halaman *dashboard* dapat dilihat pada Gambar 3.24. Di bagian kiri terdapat *sidebar* navigasi yang berisi daftar menu seperti *Dashboard, Groups, Job History, Token, Users, Audit Log,* dan *Incident*. Menu tersebut memudahkan pengguna berpindah antarhalaman sesuai kebutuhan pengelolaan data.



Gambar 3.24. Halaman *Dashboard*

Pada bagian utama, pengguna dapat melihat beberapa informasi penting, seperti:

- *Total Incidents*: menampilkan jumlah total insiden yang tercatat dalam tujuh hari terakhir.
- *Active Incidents*: menampilkan jumlah insiden yang masih dalam proses penyelesaian (*pending resolution*) dalam tujuh hari terakhir.
- *Resolved*: menunjukkan jumlah insiden yang telah berhasil diselesaikan dalam tujuh hari terakhir.
- *By Severity*: memberikan klasifikasi insiden, seperti *Major, POMI,* dan *Routine* dalam tujuh hari terakhir.

Selain itu, terdapat juga grafik *7 Days Incident Trend* yang memperlihatkan tren insiden dalam tujuh hari terakhir berdasarkan kategori. Grafik ini membantu *administrator* memantau aktivitas dan potensi gangguan secara visual.

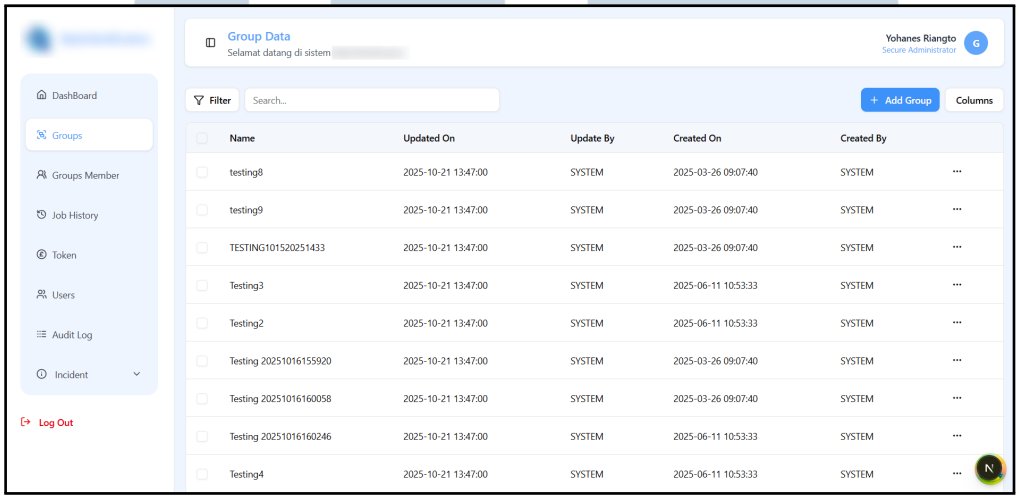
Pada bagian bawah *dashboard*, sistem menampilkan tabel daftar insiden terbaru yang dapat difilter berdasarkan statusnya, *All Incidents, Active,* atau *Resolved*. Setiap baris tabel menampilkan detail seperti nomor insiden, ringkasan (*summary*), *severity*, waktu kejadian, serta status terkini.

Secara keseluruhan, halaman *dashboard* berperan penting sebagai pusat pemantauan dan pengendalian sistem, memungkinkan pengguna untuk memperoleh

informasi kondisi sistem secara *realtime* serta mengambil tindakan cepat apabila terjadi insiden.

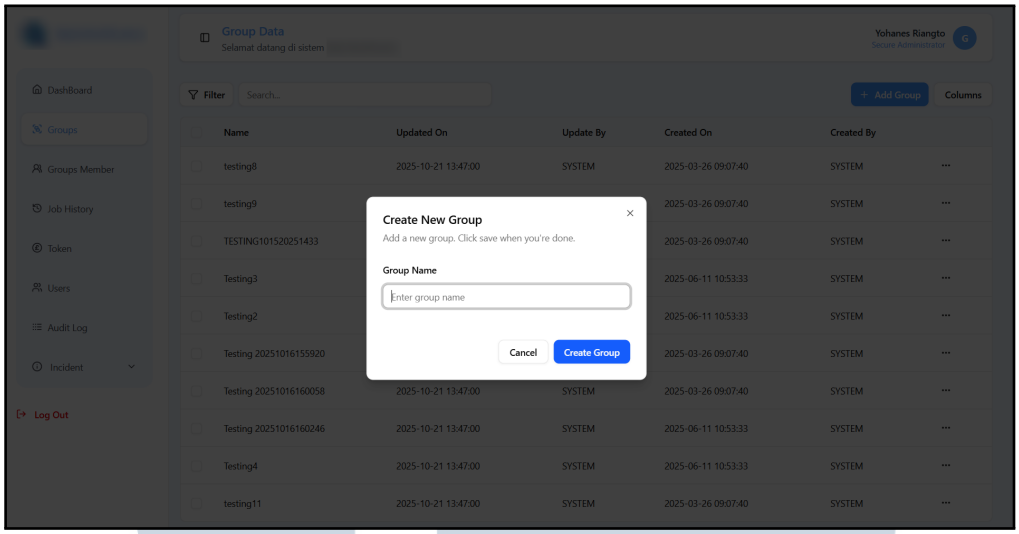
C Halaman *Groups*

Halaman *Groups* berfungsi untuk mengelola data grup pada sistem Notifiation Application. Melalui halaman ini, *administrator* dapat melihat daftar grup yang telah terdaftar, menambahkan grup baru, mengubah data grup yang sudah ada, serta menghapus grup apabila tidak lagi digunakan.



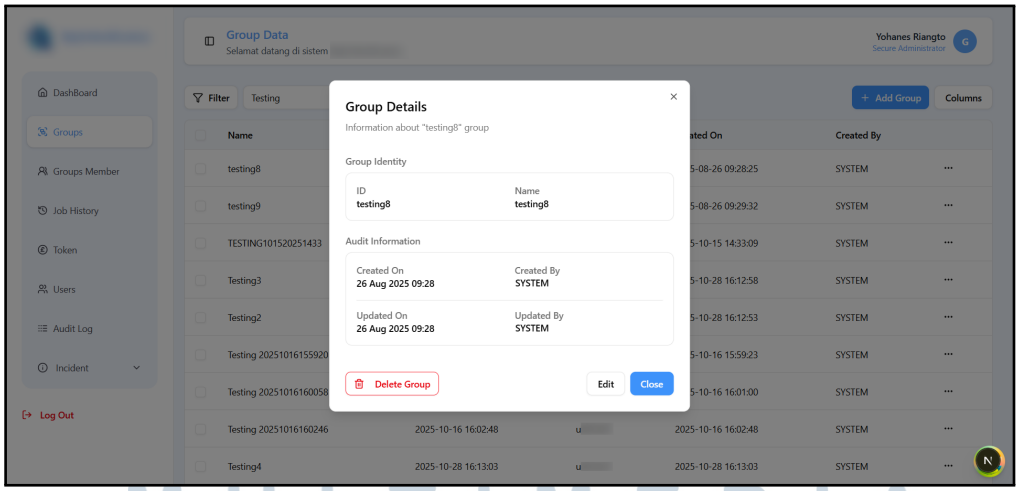
Gambar 3.25. Tampilan Halaman *Groups*

Gambar 3.25 menunjukkan tampilan utama halaman *Groups*, di mana pengguna dapat melihat seluruh data grup yang tersimpan dalam bentuk tabel. Tabel ini menampilkan beberapa kolom seperti *Name, Updated On, Updated By, Created On*, dan *Created By*. Selain itu, terdapat fitur *Filter* dan *Search* untuk mempermudah pencarian data grup tertentu. Tombol *Add Group* di sisi kanan atas digunakan untuk menambahkan grup baru, sedangkan menu tiga titik di sisi kanan setiap baris menyediakan opsi *Edit* dan *Delete*.



Gambar 3.26. Tampilan Dialog *Create Groups*

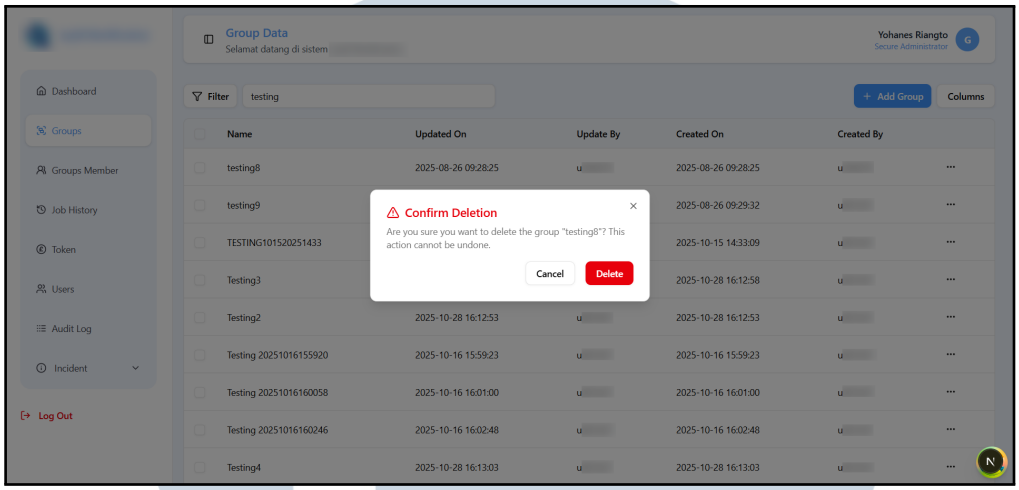
Fitur *Add Group* digunakan untuk menambahkan grup baru ke dalam sistem. Ketika tombol *Add Group* ditekan, sistem akan menampilkan *popup form* berjudul *Create New Group* seperti yang ditunjukkan pada Gambar 3.26. Pengguna dapat mengisi kolom *Group Name* untuk menambahkan nama grup baru, kemudian menekan tombol *Create Group* untuk menyimpan data. Tombol *Cancel* digunakan untuk membatalkan proses penambahan grup.



Gambar 3.27. Tampilan Dialog *Detail Groups*

Fitur *View Detail* berfungsi untuk menampilkan informasi lengkap dari sebuah grup. Setelah pengguna memilih opsi *View Detail* pada menu *actions*, sistem akan menampilkan *popup dialog* berjudul *Group Details* seperti terlihat pada Gambar 3.27. Form ini menampilkan informasi *Group Identity* (berisi ID dan *Name*) serta *Audit Information* yang terdiri dari *Created On, Created By, Updated*

On, dan *Updated By*. Selain itu, tersedia tombol *Edit* untuk langsung melakukan perubahan, serta tombol *Close* untuk menutup tampilan detail grup.

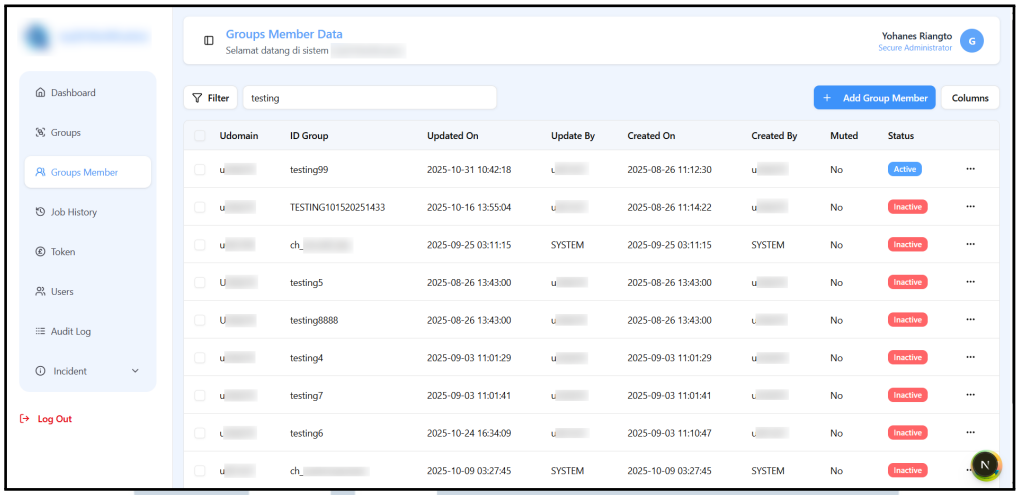


Gambar 3.28. Tampilan Dialog *Delete Groups*

Fitur *Delete Group* berfungsi untuk menghapus data grup yang tidak lagi digunakan. Setelah memilih opsi *Delete*, sistem akan menampilkan *confirmation dialog* seperti yang diperlihatkan pada Gambar 3.28. *Dialog* tersebut berisi pesan konfirmasi untuk memastikan tindakan pengguna, karena proses penghapusan tidak dapat dibatalkan. Jika pengguna menekan tombol *Delete*, maka data grup akan dihapus dari sistem. Tombol *Cancel* digunakan untuk membatalkan proses tersebut.

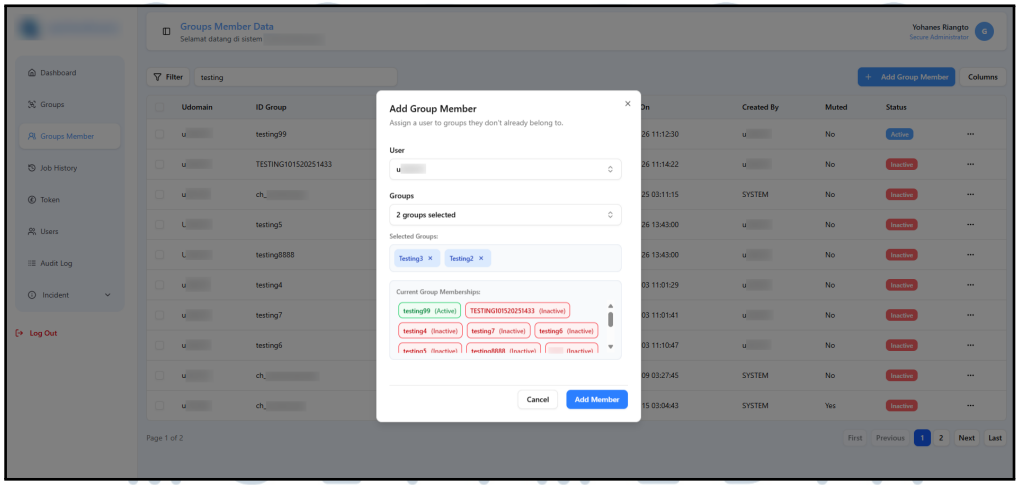
D Halaman *Group Member*

Halaman *Group Member* berfungsi untuk mengelola daftar anggota yang termasuk ke dalam suatu grup pada aplikasi *Incident Report Application*. Melalui halaman ini, *administrator* dapat melihat daftar anggota grup yang telah terdaftar, menambahkan anggota baru, memperbarui data anggota yang sudah ada, serta menghapus anggota apabila diperlukan.



Gambar 3.29. Tampilan Halaman *Group Member*

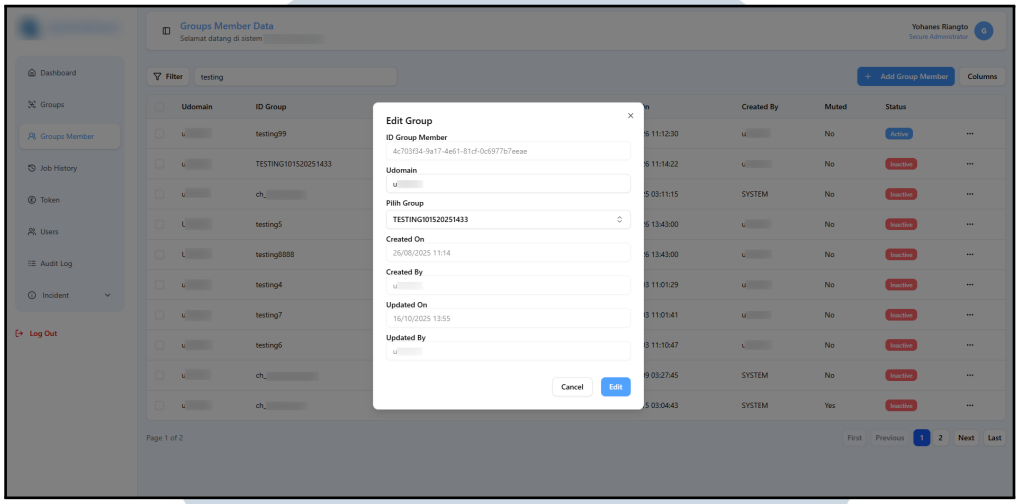
Gambar 3.29 menunjukkan tampilan utama halaman *Group Member*, di mana pengguna dapat melihat daftar seluruh anggota grup yang tersimpan dalam bentuk tabel. Tabel ini menampilkan beberapa kolom seperti *Domain*, *Group ID*, *Created On*, *Updated On*, serta status *Active* dan *Muted*. Selain itu, terdapat fitur *Filter* dan *Search* untuk mempermudah pencarian anggota tertentu. Tombol *Add Group Member* di sisi kanan atas digunakan untuk menambahkan anggota baru ke dalam grup, sementara menu tiga titik di sisi kanan setiap baris menyediakan opsi seperti *Edit*, *View Detail*, dan *Delete*.



Gambar 3.30. Tampilan Dialog *Create Group Member*

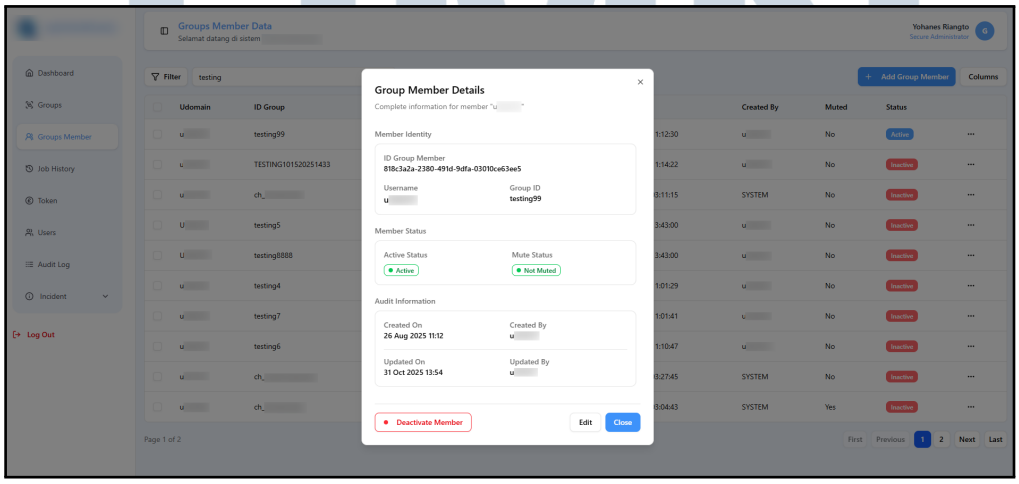
Fitur *Add Group Member* digunakan untuk menambahkan pengguna baru ke dalam satu atau lebih grup. Ketika tombol ini ditekan, sistem akan menampilkan *popup dialog* berjudul *Add Group Member* seperti yang terlihat pada Gambar 3.30. Form ini menampilkan kolom pilihan *User* dan *Groups*, di mana *administrator*

dapat memilih satu pengguna dan menambahkannya ke beberapa grup sekaligus. Setelah semua data terisi, pengguna dapat menekan tombol *Save* untuk menyimpan perubahan, atau *Cancel* untuk membatalkan proses.



Gambar 3.31. Tampilan Dialog *Edit Group Member*

Fitur *Edit Group Member* digunakan untuk memperbarui data anggota grup yang sudah ada. Seperti terlihat pada Gambar 3.31, halaman ini menampilkan *popup dialog* dengan form berisi informasi anggota yang dapat disesuaikan. *Administrator* dapat melakukan perubahan pada *Domain* atau mengganti grup yang diikuti oleh anggota tersebut. Setelah dilakukan pembaruan, pengguna dapat menekan tombol *Update* untuk menyimpan perubahan, atau *Cancel* untuk membatalkan.



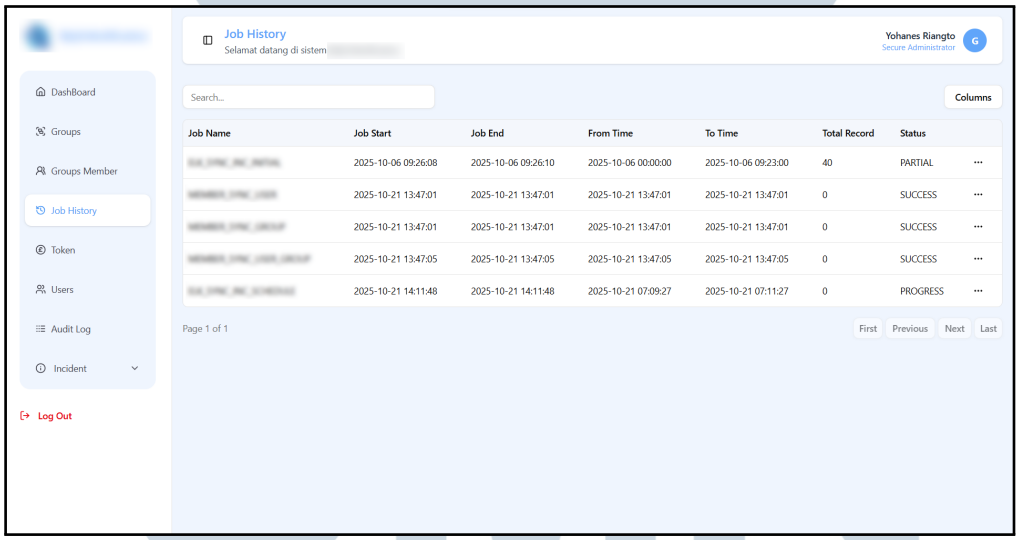
Gambar 3.32. Tampilan Dialog *View Detail Group Member*

Fitur *View Detail Group Member* berfungsi untuk menampilkan informasi lengkap mengenai anggota grup tertentu. Setelah pengguna memilih opsi *View*

Detail pada menu *actions*, sistem akan menampilkan *popup dialog* seperti yang diperlihatkan pada Gambar 3.32. Form ini menampilkan data lengkap anggota seperti *Domain*, *Group ID*, status aktif, serta informasi audit berupa *Created On*, *Created By*, *Updated On*, dan *Updated By*. Tombol *Close* digunakan untuk menutup tampilan detail, sedangkan tombol *Edit* memungkinkan pengguna langsung melakukan perubahan data dari halaman detail tersebut.

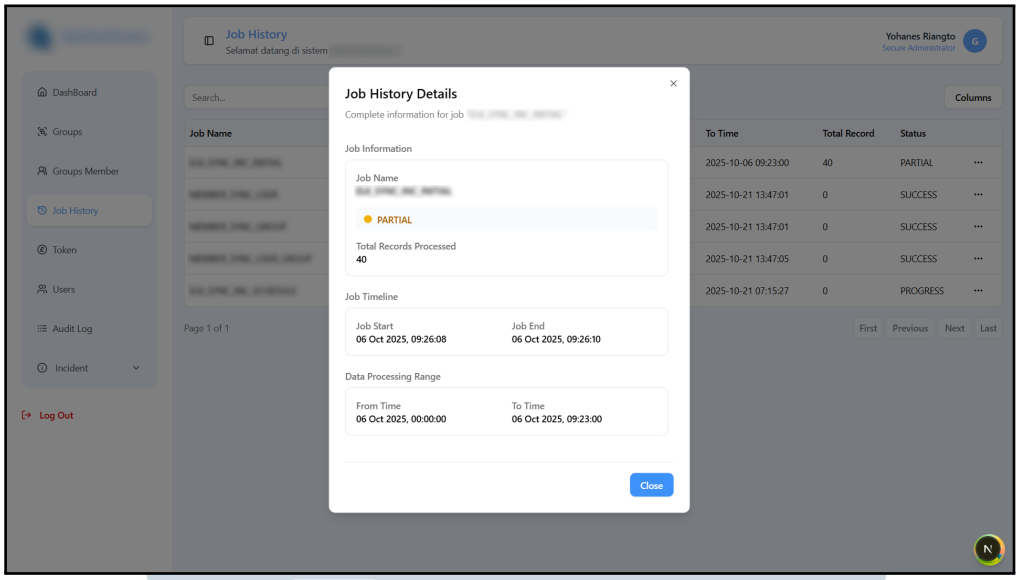
E Halaman *Job History*

Halaman *Job History* berfungsi untuk menampilkan riwayat eksekusi proses atau pekerjaan (*jobs*) yang telah dijalankan dalam sistem *Incident Report Application*. Melalui halaman ini, *administrator* dapat memantau setiap aktivitas pemrosesan data, termasuk waktu mulai, waktu selesai, serta status akhir dari setiap *job*.



Gambar 3.33. Tampilan Halaman *Job History*

Gambar 3.33 menunjukkan tampilan utama halaman *Job History*, di mana sistem menampilkan daftar seluruh riwayat pekerjaan dalam bentuk tabel. Tabel ini memiliki beberapa kolom penting seperti *Job Name*, *Job Start*, *Job End*, *From Time*, *To Time*, *Total Record*, dan *Status*. Kolom *Status* menunjukkan hasil akhir dari proses yang telah dijalankan, seperti *Success*, *Failed*, atau *In Progress*. Selain itu, tersedia fitur *Filter* dan *Search* untuk memudahkan pengguna mencari riwayat pekerjaan berdasarkan nama atau rentang waktu tertentu.

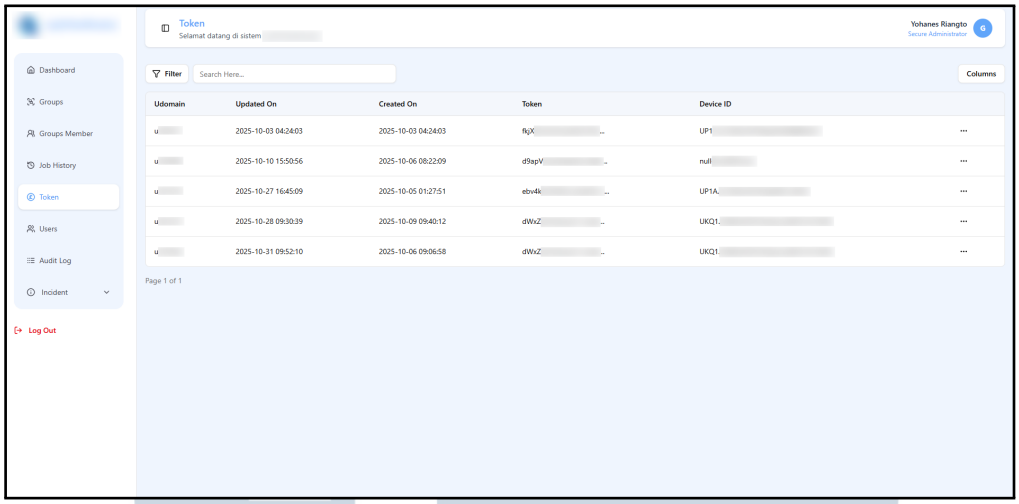


Gambar 3.34. Tampilan Dialog *View Detail Job History*

Fitur *View Detail* berfungsi untuk menampilkan informasi lebih rinci mengenai proses pekerjaan tertentu. Seperti terlihat pada Gambar 3.34, ketika pengguna memilih opsi *View Detail* dari menu *actions*, sistem akan menampilkan *popup dialog* berjudul *Job Details*. *Dialog* ini berisi informasi detail seperti *Job ID*, *Job Name*, waktu mulai dan selesai, rentang waktu data yang diproses (*From Time* dan *To Time*), jumlah total data yang diproses, serta pesan status atau keterangan hasil eksekusi. Tombol *Close* digunakan untuk menutup tampilan detail dan kembali ke halaman utama *Job History*.

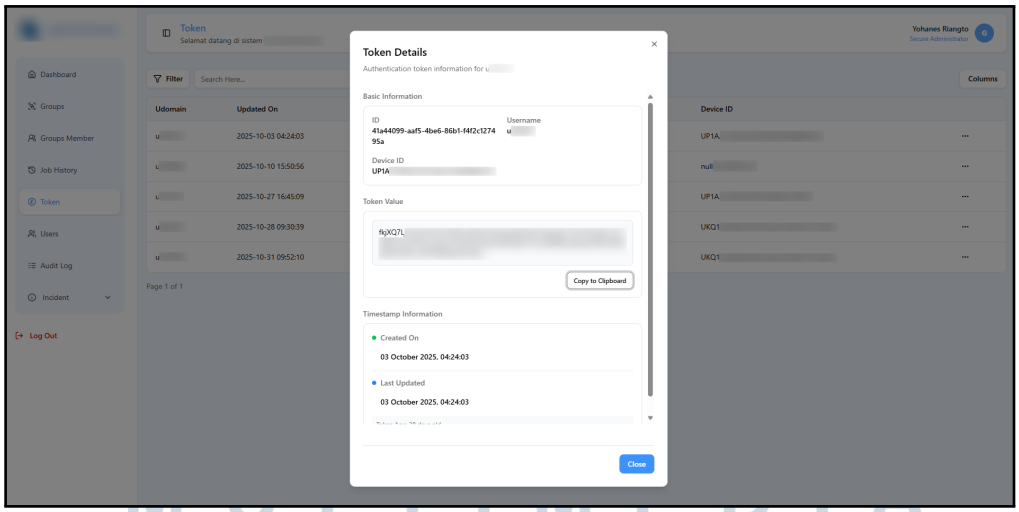
F Halaman *Token*

Halaman *Token* berfungsi untuk menampilkan serta mengelola daftar *token* yang digunakan oleh perangkat (*device*) dalam sistem *Incident Report Application*. Melalui halaman ini, *administrator* dapat memantau setiap *token* yang terdaftar, termasuk informasi *domain*, waktu pembaruan, dan identitas perangkat yang menggunakan *token* tersebut.



Gambar 3.35. Tampilan Halaman *Token*

Gambar 3.35 menunjukkan tampilan utama halaman *Token*, di mana sistem menampilkan seluruh data token dalam bentuk tabel. Tabel ini memiliki beberapa kolom utama seperti *Domain*, *Updated On*, *Created On*, *Token*, dan *Device ID*. Kolom *Domain* menunjukkan domain dari pemilik token, sedangkan *Device ID* menampilkan identitas perangkat yang terkait. Selain itu, tersedia fitur *Filter* dan *Search* untuk memudahkan pencarian data token tertentu berdasarkan domain atau perangkat.



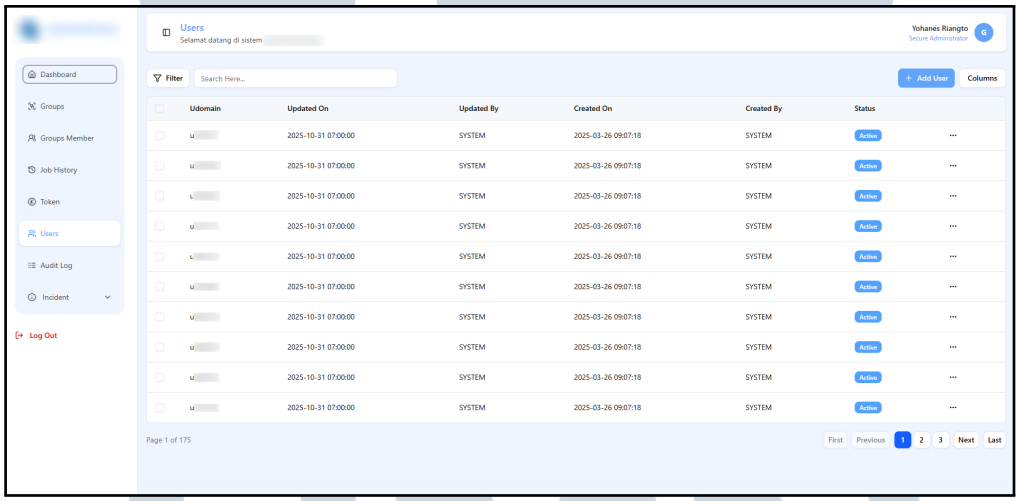
Gambar 3.36. Tampilan Dialog *View Detail Token*

Fitur *View Detail* berfungsi untuk menampilkan informasi lebih rinci mengenai token tertentu. Seperti terlihat pada Gambar 3.36, ketika pengguna memilih opsi *View Detail* dari menu *actions*, sistem akan menampilkan jendela dialog berjudul *Token Details*. Dialog ini berisi informasi lengkap mengenai

Domain, *Token*, *Device ID*, waktu pembuatan, serta waktu terakhir pembaruan token tersebut. Tombol *Close* digunakan untuk menutup tampilan detail dan kembali ke halaman utama *Token*.

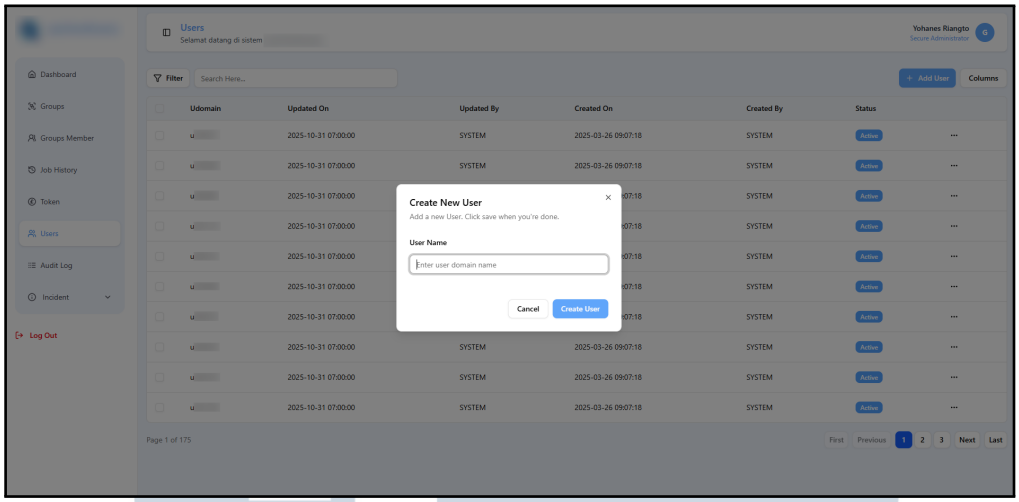
G Halaman *User*

Halaman *User* berfungsi untuk mengelola data pengguna yang memiliki akses terhadap sistem *Incident Report Application*. Melalui halaman ini, *administrator* dapat melihat daftar pengguna yang terdaftar, menambahkan pengguna baru, menonaktifkan akun tertentu, serta melihat detail informasi dari setiap pengguna.



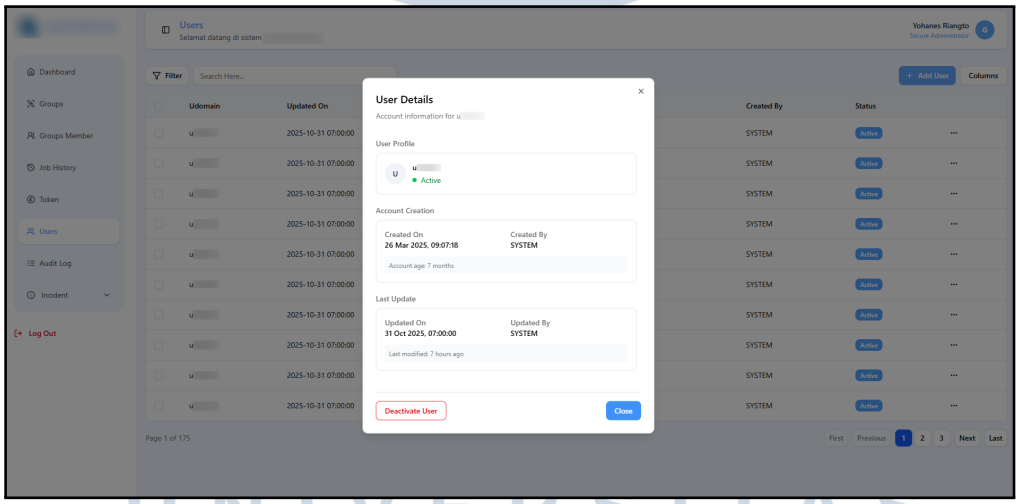
Gambar 3.37. Tampilan Halaman *User*

Gambar 3.37 menunjukkan tampilan utama halaman *User*, di mana sistem menampilkan daftar seluruh pengguna dalam bentuk tabel. Tabel ini terdiri atas beberapa kolom seperti *Domain*, *Updated On*, *Updated By*, *Created On*, *Created By*, dan *Is Active*. Kolom *Is Active* menunjukkan status aktif atau tidaknya akun pengguna. Selain itu, tersedia fitur *Search* dan *Filter* untuk mempermudah pencarian data pengguna berdasarkan domain, status aktif, *created on*, dan *updated on*.



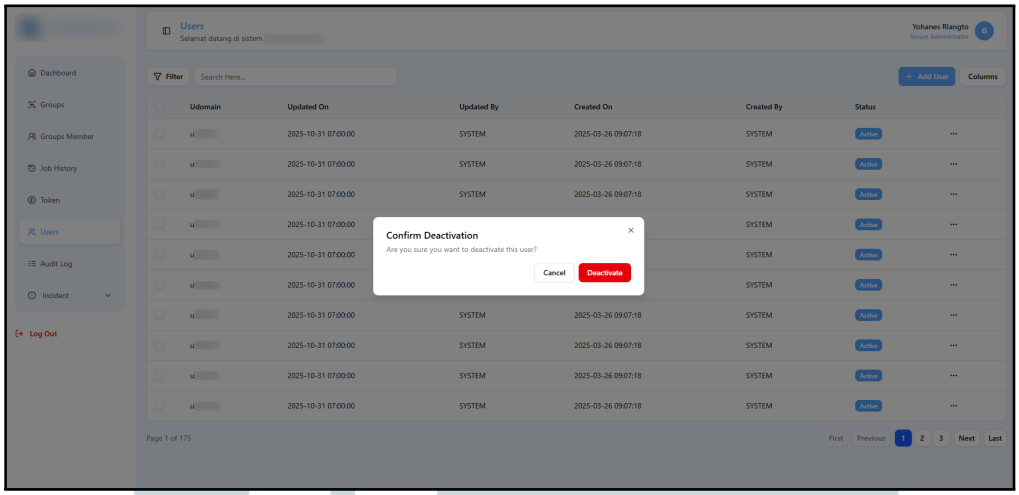
Gambar 3.38. Tampilan Dialog *Add User*

Fitur *Add User* digunakan untuk menambahkan pengguna baru ke dalam sistem. Seperti ditunjukkan pada Gambar 3.38, ketika tombol *Add User* ditekan, sistem akan menampilkan jendela dialog untuk mengisi data pengguna baru, seperti *Domain*. Setelah data diisi dan disimpan, pengguna baru akan muncul dalam daftar tabel utama.



Gambar 3.39. Tampilan Dialog *View Detail User*

Fitur *View Detail* berfungsi untuk melihat informasi lengkap mengenai pengguna tertentu. Pada Gambar 3.39 ditampilkan dialog yang berisi detail data pengguna, termasuk informasi waktu pembuatan, pembaruan terakhir, status akun, serta domain yang digunakan. Dialog ini juga menampilkan identitas siapa yang melakukan pembaruan terakhir terhadap akun tersebut dan siapa yang membuat data *user* tersebut.

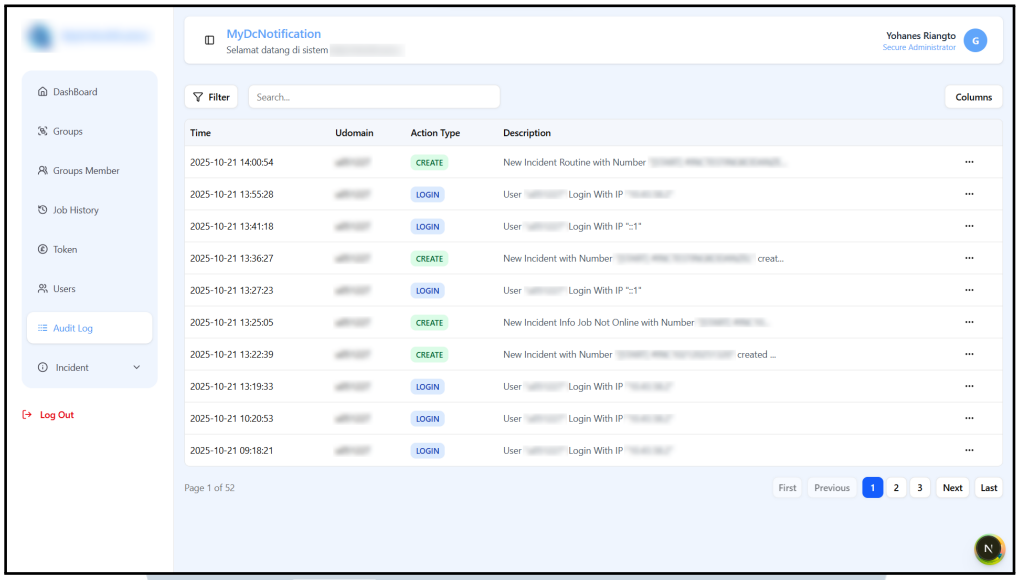


Gambar 3.40. Tampilan *Dialog Deactive User*

Fitur *Deactive User* digunakan untuk menonaktifkan akun pengguna tanpa harus menghapus data secara permanen. Seperti terlihat pada Gambar 3.40, ketika opsi *Deactive* dipilih, sistem akan menampilkan dialog konfirmasi untuk memastikan tindakan tersebut. Setelah dikonfirmasi, status pengguna akan berubah menjadi tidak aktif (*Is Active = False*) dan pengguna tidak dapat lagi mengakses sistem sampai diaktifkan kembali oleh *administrator*.

H Halaman *Audit Log*

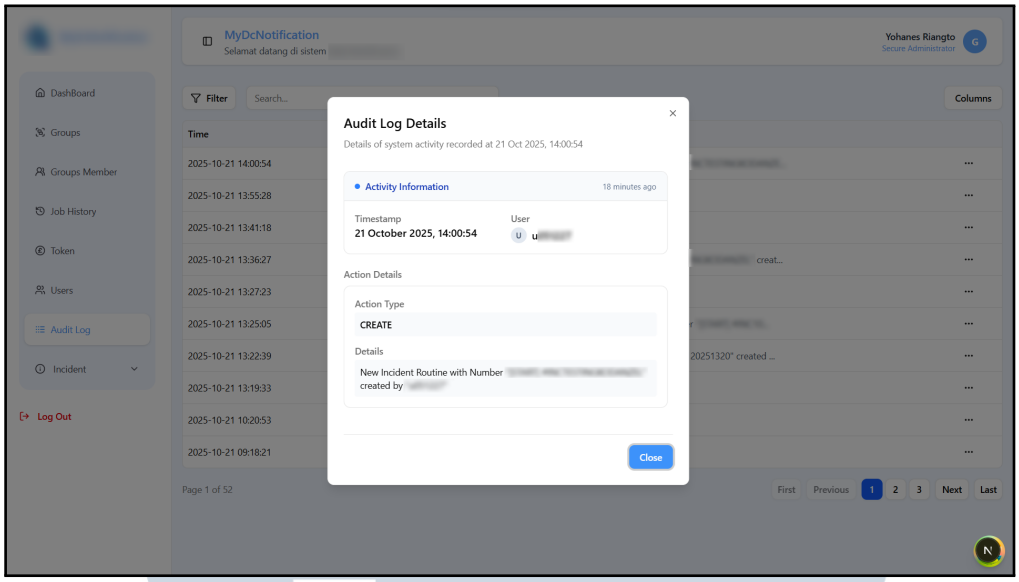
Halaman *Audit Log* berfungsi untuk menampilkan catatan aktivitas (*log*) yang terjadi di dalam sistem CMS *Incident Report Application*. Melalui halaman ini, *administrator* dapat memantau setiap tindakan yang dilakukan oleh pengguna pada sistem *CMS Incident Report*, seperti proses pembuatan data, pembaruan, penghapusan, maupun aktivitas masuk ke sistem.



Gambar 3.41. Tampilan Halaman *Audit Log*

Gambar 3.41 menunjukkan tampilan utama halaman *Audit Log*, di mana seluruh aktivitas sistem ditampilkan dalam bentuk tabel. Tabel ini memiliki beberapa kolom utama seperti *Time*, *Domain*, *Action Type*, dan *Description*. Kolom *Action Type* menampilkan jenis aksi yang dilakukan oleh pengguna, seperti *Create*, *Login*, *Delete*, atau *Update*. Sementara kolom *Description* berisi informasi mengenai objek atau data yang terpengaruh oleh aksi tersebut.

Selain itu, halaman ini dilengkapi dengan fitur *Search* dan *Filter* yang memungkinkan *administrator* melakukan pencarian aktivitas berdasarkan waktu, domain, atau jenis aksi tertentu. Fitur ini sangat membantu dalam proses audit dan pelacakan riwayat perubahan data di dalam sistem.

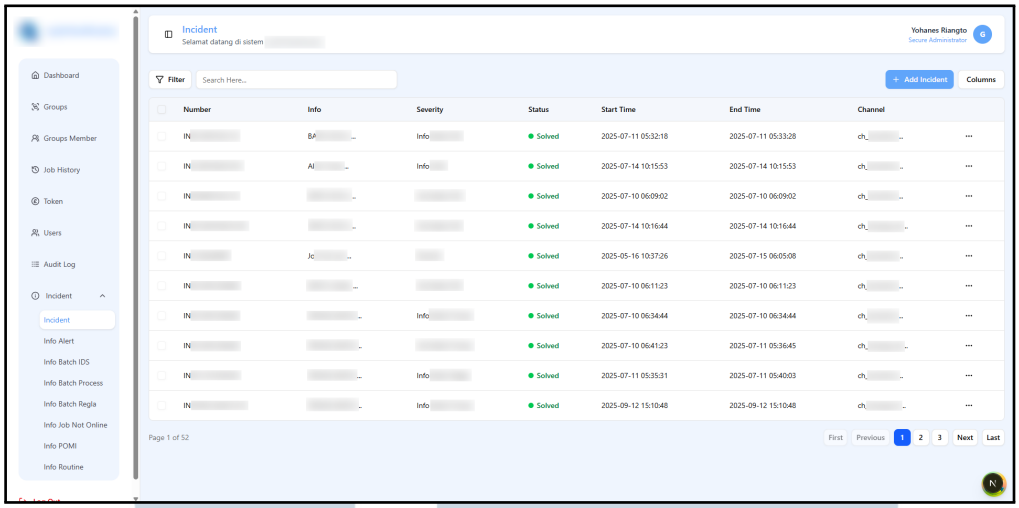


Gambar 3.42. Tampilan Dialog *View Detail Audit Log*

Fitur *View Detail* digunakan untuk menampilkan informasi lebih rinci dari suatu entri *audit log*. Seperti terlihat pada Gambar 3.42, ketika pengguna memilih opsi *View Detail* dari menu *actions*, sistem akan menampilkan *popup dialog* yang berisi detail aktivitas, termasuk waktu kejadian, nama domain, tipe aksi, deskripsi lengkap, serta identitas pengguna yang melakukan aksi tersebut. Dialog ini membantu *administrator* untuk memahami konteks dari setiap aktivitas yang tercatat dan memastikan keamanan serta integritas sistem.

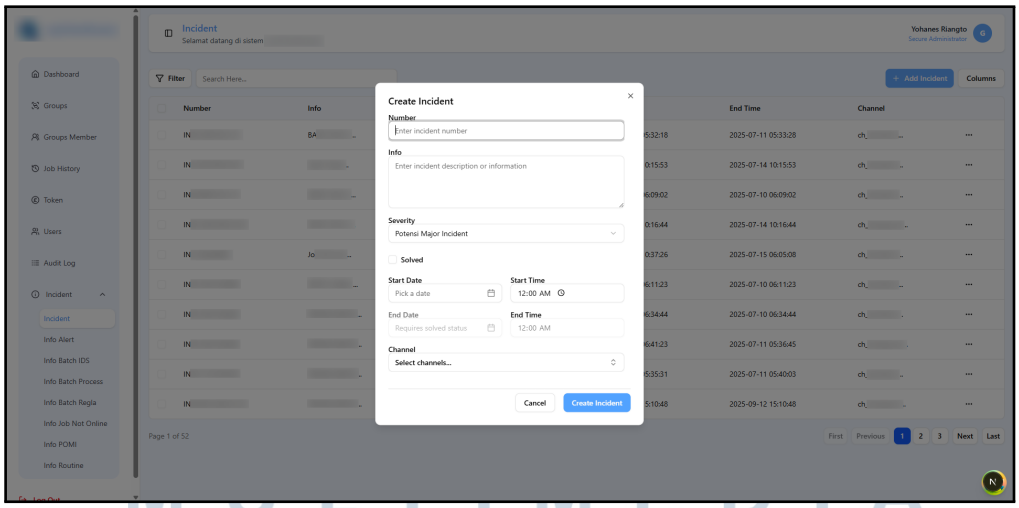
I Halaman *Incident*

Halaman *Incident* berfungsi untuk mengelola data *incident* pada sistem *Incident Report Application*. Melalui halaman ini, *administrator* dapat melihat data *incident* yang telah tercatat, menambahkan laporan *incident* baru, memperbarui data *incident* yang sudah ada, serta meninjau detail informasi *incident* secara lengkap. Fitur ini menjadi salah satu komponen utama dalam CMS, karena memungkinkan *administrator* untuk memperbaiki atau menambahkan data *incident* yang gagal diproses oleh sistem akibat kesalahan format pesan dari hasil *regex parsing*.



Gambar 3.43. Tampilan Halaman *Incident*

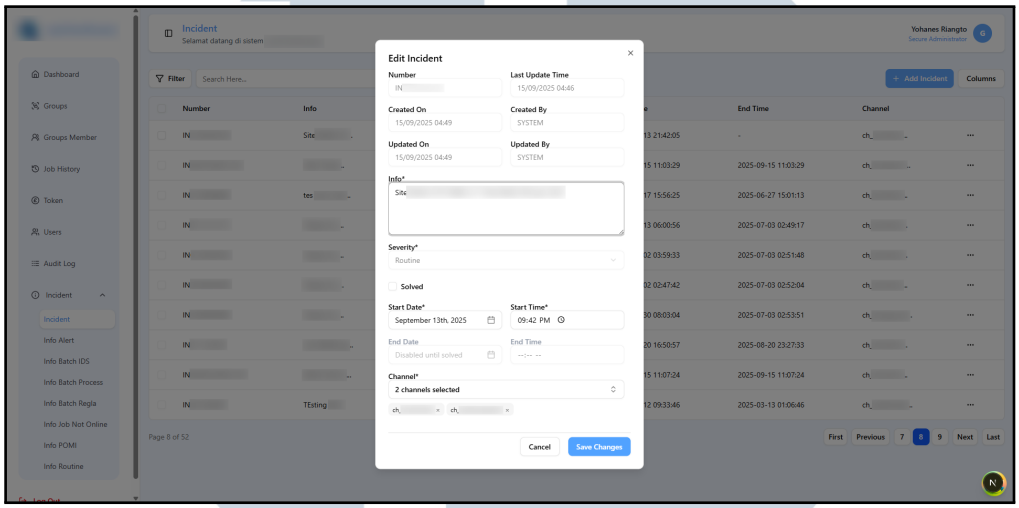
Gambar 3.43 menunjukkan tampilan utama halaman *Incident*, di mana pengguna dapat melihat seluruh data *incident* yang tersimpan dalam bentuk tabel. Tabel ini menampilkan beberapa kolom seperti *Number*, *Info*, *Severity, Status, Start Time, End Time,* dan *Channel*. Selain itu, terdapat fitur *Filter* dan *Search* untuk mempermudah pencarian data *incident* tertentu. Tombol *Add Incident* di sisi kanan atas digunakan untuk menambahkan laporan *incident* baru, sedangkan menu tiga titik di sisi kanan setiap baris menyediakan opsi untuk *Edit* dan *View Detail*.



Gambar 3.44. Tampilan *Dialog Create Incident*

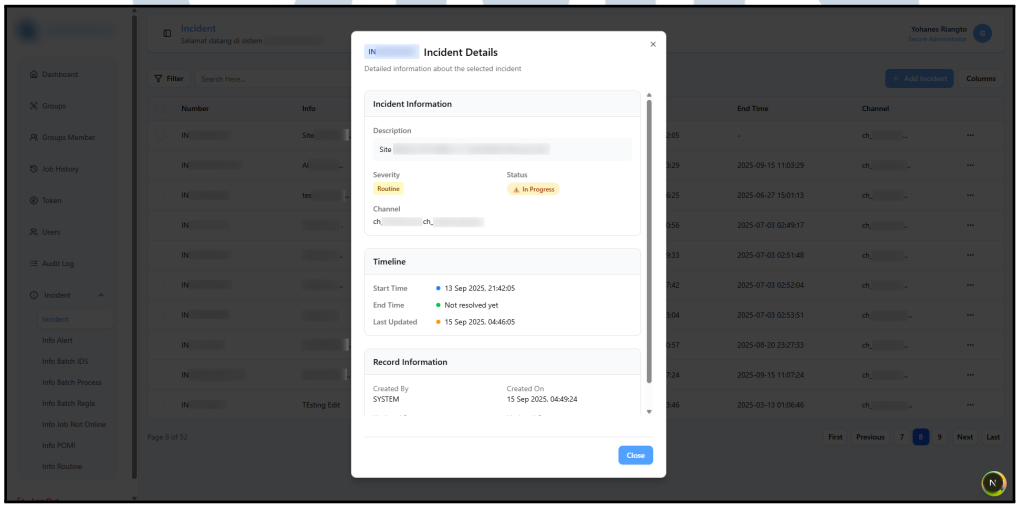
Fitur *Add Incident* digunakan untuk menambahkan laporan *incident* baru ke dalam sistem. Ketika tombol *Add Incident* ditekan, sistem akan menampilkan *popup form Create Incident* seperti yang ditunjukkan pada Gambar 3.44. Pengguna dapat mengisi kolom seperti *Number, Info, Severity, Start Date, Start Time, End*

Date, End Time, serta *Channel*. Terdapat pula opsi checkbox "*Solved*" untuk menandai bahwa *incident* telah diselesaikan. Setelah semua data diisi, pengguna dapat menekan tombol *Create Incident* untuk menyimpan data ke dalam sistem, atau tombol *Cancel* untuk membatalkan proses penambahan.



Gambar 3.45. Tampilan *Dialog Edit Incident*

Fitur *Edit Incident* berfungsi untuk memperbarui informasi *incident* yang telah tersimpan. Ketika pengguna memilih opsi *Edit*, sistem akan menampilkan *popup form Edit Incident* seperti terlihat pada Gambar 3.45. *Form* ini menampilkan informasi waktu pembuatan (*Created On*), waktu pembaruan (*Updated On*), serta identitas pengguna pembuat dan pengubah data. *Administrator* dapat memperbarui informasi seperti *Info, Severity,* Waktu Kejadian, dan *Channel*. Setelah perubahan dilakukan, pengguna dapat menekan tombol *Save Changes* untuk menyimpan pembaruan data atau *Cancel* untuk membatalkan.



Gambar 3.46. Tampilan *Dialog Detail Incident*

Fitur *View Detail* berfungsi untuk menampilkan informasi lengkap dari sebuah *incident*. Setelah pengguna memilih opsi *View Detail* pada menu *actions*, sistem akan menampilkan *popup dialog Incident Details* seperti terlihat pada Gambar 3.46. *Form* ini menampilkan informasi *Description, Severity, Status*, dan *Channel*. Selain itu, terdapat bagian *Timeline* yang berisi waktu mulai *(Start Time)*, waktu selesai *(End Time)*, serta waktu pembaruan terakhir *(Last Updated)*. Bagian bawah *form* menampilkan *Record Information* yang mencatat data pembuat *(Created By)* dan waktu pembuatan *(Created On)*.

J Pengujian Sistem

Pengujian sistem dilakukan untuk memastikan bahwa setiap fitur pada *Content Management System (CMS) Incident Report* telah berfungsi sesuai dengan kebutuhan dan spesifikasi yang ditetapkan. Metode pengujian yang digunakan adalah *black box testing*, yaitu metode yang berfokus pada pengujian fungsionalitas sistem tanpa memperhatikan struktur internal kode program. Pendekatan ini dilakukan dari sudut pandang pengguna untuk memastikan bahwa keluaran (*output*) sesuai dengan masukan (*input*) yang diberikan.

Proses pengujian dilakukan dengan membuat sejumlah skenario berdasarkan fitur utama sistem, seperti autentikasi pengguna, pengelolaan data incident, manajemen grup dan anggota, serta pencatatan audit log. Setiap skenario mencakup langkah uji, data uji, hasil yang diharapkan, serta status hasil pengujian.

Tabel 3.3. Contoh Skenario Pengujian Sistem CMS Incident Report

No	Nama Skenario	Deskripsi Pengujian	Input	Hasil yang Diharapkan	Status
1	*Login User*	Menguji autentikasi pengguna dengan kredensial valid.	Username dan Password valid	Pengguna berhasil masuk ke sistem.	Passed
2	*Create Group*	Menguji penambahan data grup baru ke dalam sistem.	Nama grup baru	Data grup tersimpan di database dan tampil di tabel.	Passed
3	*Edit Groups*	Menguji pembaruan data *Groups* yang sudah ada.	Update kolom *Name*	Data Nama *Groups* berhasil diperbarui di database.	Passed
4	*Delete Users*	Menguji penghapusan data *User* secara *soft delete*.	ID *User*	Status *is_active Users* berubah menjadi tidak aktif.	Passed
5	View Audit Log	Memeriksa pencatatan aktivitas sistem setelah aksi CRUD dilakukan.	Aksi CRUD	Aktivitas tercatat pada tabel audit log.	Passed

Setelah seluruh skenario diuji, hasil pengujian diserahkan kepada tim *Quality Assurance (QA)* untuk dilakukan verifikasi lebih lanjut. Tim QA melakukan validasi terhadap fungsi-fungsi utama sistem untuk memastikan bahwa aplikasi telah memenuhi kriteria kelayakan dan siap untuk diterapkan pada lingkungan *production*.

Berikut adalah laporan hasil pengujian dari tim QA yang digunakan dalam proyek ini:

Test Case Scenario						
ID	Action	Expected Result	Actual Result	Status	Environment	Method
9.4.1	Klik 3 titik pada salah satu data Info Job Not Online di sisi kanan tabel	Dapat memunculkan 2 opsi	TS 9 - Verifikasi Incident - Info Job Not Online (Negative).xlsx	Passed	Development	Unit Testing
9.4.2	Pilih opsi Edit	Dapat memunculkan modal Edit Job Not Online Information				
9.4.3	Edit form Edit Job Not Online Information - Info: (diisi bebas menggunakan symbol) - Status: (diisi bebas menggunakan symbol) - PIC: (diisi bebas menggunakan symbol) - Impact: (diisi bebas menggunakan symbol) - Sent by: (diisi bebas menggunakan symbol)	Form dapat diedit				
9.4.4	Klik tombol Save Changes	Dapat menutup modal Edit Job Not Online Information				
9.4.5	Verifikasi data	Menampilkan data sesuai dengan hasil edit di tabel				
9.4.6	Login aplikasi Notification	Dapat masuk ke halaman Home				
9.4.7	Verifikasi data di aplikasi	Menampilkan data sesuai dengan hasil edit di tabel				

Gambar 3.47. Contoh *Scenario Case* Untuk Pengujian Sistem oleh Tim QA

Test Result			
		Document Version	2.2
		Confirmation Date	5-Nov-25
Application	CMS Notification Application		
PIC IT	Yohanes Riangto & Willy Ng		
Feature Inisiator	Galuh Dhian Wibowo		
Time to Market	Nov-25		
App Description	Merupakan aplikasi yang digunakan untuk mengelola aplikasi Notification Application		
Objective	Dapat memantau serta melakukan update incident yang terdapat di aplikasi Notification Application		

Test Summary	Total Test Scenario	Total Test Case	Total Findings	PIC QA	PIC UAT	Status
Fitur Sudah Dites dan Siap Naik ke Production	28	113	53	Danzel Sasputra Tama	-	Passed

DC Support @ 2025

Gambar 3.48. *Test Result* Pengujian Sistem oleh Tim QA

Berdasarkan laporan hasil pengujian di atas, terdapat beberapa temuan (*findings*) selama proses uji coba. Namun, seluruh temuan tersebut telah ditangani dan diperbaiki, dan kemudian dilakukan *retesting* hingga seluruh skenario uji dinyatakan berhasil (*Passed*) tanpa menyisakan kesalahan fungsional maupun ketidaksesuaian terhadap kebutuhan sistem. Dengan demikian, CMS *Incident Report Application* dinyatakan siap untuk diimplementasikan pada lingkungan *production* dan telah memenuhi standar kelayakan dari sisi fungsionalitas.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

1. Selama proses pengujian pada lingkungan lokal, muncul kendala akibat adanya pembatasan proxy perusahaan yang mengakibatkan *error PKIX Certification* ketika sistem mencoba mengakses API dari APIGateway.
2. Sistem backend yang dikembangkan berada di dalam lingkungan intranet perusahaan, sehingga hanya dapat diakses melalui jaringan internal kantor. Kondisi ini menjadi hambatan ketika melakukan proses pengembangan atau pengujian dari luar kantor, karena perangkat laptop yang digunakan tidak memiliki akses ke jaringan tersebut.

3. Tantangan lain muncul pada saat pembuatan chart "*7 Days Incident Trend*" di sisi *frontend*. Hal ini disebabkan oleh data yang dikirim dari backend berupa *JSON* hasil agregasi data mingguan, sehingga memerlukan pengolahan tambahan di sisi *frontend* untuk menampilkan visualisasi tren dengan benar.

3.4.2 Solusi

1. Untuk mengatasi kendala sertifikasi pada saat pengujian lokal, diterapkan penggunaan *UnsafeRestTemplate*, yang memungkinkan bypass terhadap validasi sertifikat. Dengan cara ini, proses pengujian terhadap *API APIGateway* dapat berjalan dengan lancar tanpa terhalang masalah sertifikat.

2. Dalam menghadapi keterbatasan akses jaringan internal, peserta magang memanfaatkan layanan *cloud storage* untuk menyimpan referensi dan dokumen penting. Pendekatan ini memungkinkan proses penyusunan dan dokumentasi tetap dapat dilakukan menggunakan perangkat pribadi di luar lingkungan kantor.

3. Untuk mengatasi kesulitan dalam menampilkan chart "*7 Days Incident Trend*", dilakukan pencarian dan penerapan *library* visualisasi data yang sesuai. Solusi yang dipilih adalah penggunaan *Recharts*, yang mendukung pemetaan data dinamis dari format *JSON* ke dalam bentuk grafik interaktif, sehingga proses visualisasi tren insiden harian dapat dilakukan secara efisien dan informatif.