# BAB 3
# PELAKSANAAN KERJA MAGANG

## 3.1 Kedudukan dan Organisasi

Posisi yang ditetapkan selama pelaksanaan program kerja magang di PT. Vanz Inovatif Teknologi adalah sebagai *Developer* di departemen *Internal/Inhouse Application and Development*. Program kerja magang ini dilaksanakan dengan bimbingan Aditia Prasetio sebagai *supervisor* dan *Team Leader* dari tim *developer Internal/Inhouse Application Development*. Koordinasi dilakukan setiap harinya melalui aplikasi *Google Meet*, media komunikasi yang digunakan adalah WhatsApp, dan email untuk menginformasikan dan menjadwalkan *meeting*, baik *meeting* internal ataupun *meeting* dengan klien.

## 3.2 Tugas yang Dilakukan

Terdapat tugas-tugas yang diberikan oleh *supervisor* untuk merancang, membangun, dan melakukan *bug fixing* pada fitur-fitur yang ada pada aplikasi manajemen data berbasis web, baik dalam proyek yang masih dalam tahap *development* maupun proyek yang sudah *live*.

Program kerja magang diawali dengan menjalankan proses *onboarding* ke dalam tim *developer Internal/Inhouse Application Development*. *Developer onboarding* ini dimulai dengan pemberian akses ke repositori Github perusahaan dan diberi undangan ke grup WhatsApp tim *developer*. Setelah itu dijelaskan kegiatan yang dilakukan selama satu hari kerja yang dimulai dengan *daily meeting* untuk rekap hasil kerja setiap individu pada hari sebelumnya dan diakhiri dengan *update progress* pada grup WhatsApp tim *developer*. Proses *onboarding* dilanjutkan dengan tutorial untuk *setup environment*. Karena proses *development* menggunakan Linux, perlu di-*install* Windows Subsystem Linux (WSL) bagi perangkat yang menggunakan *operating system* Windows. Selain itu, diperlukan juga PostgreSQL dan Redis Server sebagai sistem *database*.

Setelah selesai melakukan *setup environment*, dilakukan *clone* repositori sampel *backend* dan template *content management system (CMS)* ke subsistem Linux yang sudah di-*install*. Ketika sampel *backend* dan *template* CMS sudah di-*setup* dan dapat dijalankan, diberi tugas untuk menambahkan fitur untuk menyimpan nama dan logo sponsor pada *backend* dan *template* CMS.

Tugas selanjutnya adalah untuk merubah bentuk data *permission* yang sudah ada pada *database* menjadi bentuk data yang baru dengan membuat *function* konversi serta menjalankan *function* tersebut pada *template* CMS dan menggunakan *endpoint* yang ada di *backend* untuk memperbaharui data *permission* yang ada pada database. *Function* ini kemudian dipindahkan ke *library* privat perusahaan yang mengatur *role* dan *permission*.

Karena fitur-fitur pada *backend* diimplementasikan melalui *library* privat perusahaan, maka fitur sponsor yang sebelumnya dibuat serta fitur artikel yang ada pada proyek lain perlu dibuat *library-library* baru untuk kedua fitur tersebut. Oleh karena itu, tugas selanjutnya adalah memindahkan fitur sponsor dan fitur artikel ke *library* terpisah pada repositori Github sendiri untuk masing-masing fitur.

Setelah itu, tugas selanjutnya adalah melakukan *bug fixing* serta menambahkan beberapa limitasi dalam proses input data pada fitur shipment dalam proyek Track n Trace (TnT). Ketika tugas tersebut selesai, diberi tugas untuk melakukan riset dan implementasi API dari Meta, terutama WhatsApp Business dan Instagram Business. Untuk menggunakan kedua API tersebut, perlu dibuat Meta App dan Meta Business Account serta pengaturan yang sesuai. Keperluan implementasi WhatsApp Business API adalah untuk melakukan *broadcast* pesan dan Instagram Graph API digunakan untuk mendapatkan data media dari *post* yang ada pada akun Instagram Business yang dihubungkan ke Meta Business Account yang sebelumnya sudah dibuat.

Tugas terakhir dalam masa program kerja magang di PT. Vans Inovatif Teknologi adalah memastikan bahwa kode proyek Mayora yang di-*recover* dari server produksi perusahaan klien dapat dijalankan secara lokal pada perangkat pribadi. Hal ini dilakukan karena staf klien mengamati adanya anomali data pada tampilan proyek Mayora. Anomali data yang ditemukan berupa angka total yang tidak sesuai, data yang hilang, dan data yang muncul dengan sendirinya.

### 3.3 Uraian Pelaksanaan Magang

Selama durasi program kerja magang di PT. Vanz Inovatif Teknologi, tugas-tugas yang diberikan terbagi ke beberapa proyek, yaitu:

1. *Template* CMS
2. Proyek TracknTrace
3. Proyek Gudang Acai

4. Proyek Mayora

5. Integrasi Media Sosial WhatsApp dan Instagram ke CMS

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

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

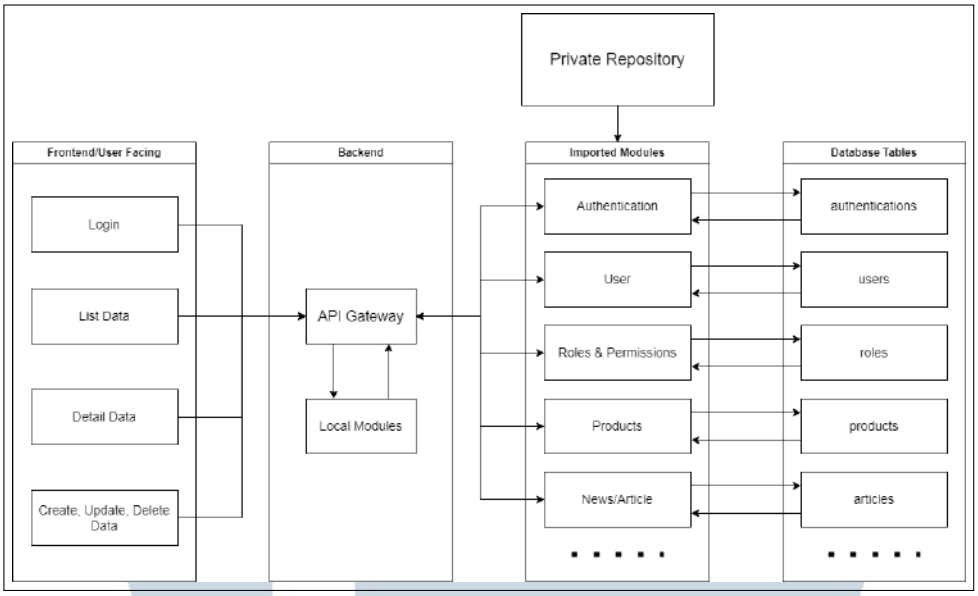
| Minggu Ke - | Pekerjaan yang dilakukan |
|---|---|
| 1 | *Setup environment* untuk melakukan pengembangan aplikasi dan mencoba menjalankan sampel *backend* dan *template* CMS secara lokal. Pembuatan fitur sponsor pada sampel *backend* dan CMS lokal. |
| 2 | Membuat *function* untuk mengecek versi bentuk data dan mengubah bentuk data *permission*. Menambahkan *function upload image* di fitur sponsor. |
| 3, 4, & 5 | Memindahkan fitur artikel dari proyek ke *library* privat dan melakukan *test* implementasi ke *backend* dan CMS lokal. |
| 6 | *Monitoring* dan *tracing* isu koneksi antara aplikasi proyek Kalventis dan *API production* klien |
| 7 | *Bug fixing* pada fitur shipment dan return serta penambahan limitasi pada input data dalam proyek TracknTrace (TnT) |
| 8 | Menambahkan DOKU Checkout sebagai *payment gateway* baru pada library privat *libs-payment* |
| 9 | *Setup* Meta Business Account, Meta Application, dan Instagram Business Account untuk menggunakan Whatsapp Business API dan Instagram Graph API. |
| 10 | Implementasi *function* untuk melakukan cek data duplikat pada proyek TracknTrace (TnT). |
| 11 | Riset dan *testing* Instagram Graph API dan integrasi ke *backend* CMS lokal. |
| 12 & 13 | Menambahkan iLock pada proyek TracknTrace untuk mencegah perubahan data. |
| 14 & 15 | Melanjutkan integrasi Instagram Graph API ke *backend* CMS lokal dan *fixing* isu generasi dokumen pada proyek Gudang Acai. |
| 16 | Melakukan *setup* dan *testing code* proyek Mayora yang di-*recover* dari server *production* untuk *tracing* dan *fixing* anomali data. |

### 3.3.1 *Content Management System (CMS) Template*

*Content Management System* (CMS) adalah aplikasi berbasis web yang berfungsi untuk mengelola dan manajemen data. Data yang dikelola melalui CMS berupa data konten yang ditampilkan dalam suatu *web page* seperti artikel, sponsor, dan media, data autentikasi seperti daftar pengguna, *role* yang dapat diberikan kepada setiap pengguna, dan *permission* yang ada pada setiap *role*, serta data operasional perusahaan seperti produk dan lokasi perusahaan. PT. Vans Inovatif Teknologi memiliki *template* CMS sebagai basis dari semua proyek yang dibuat dan dikembangkan. Sebuah aplikasi CMS terdiri dari *backend* dan *frontend*, *Backend* dari sebuah CMS terkoneksi dengan database dan terdiri dari berbagai *endpoint* yang tersedia pada setiap fitur yang di-*import* dari *library-library* privat perusahaan. *Endpoint* yang ada pada *backend* CMS dapat dipanggil melalui *frontend* CMS melalui *Http request*. Bagian *frontend* dari CMS terdiri dari beberapa *web page* yang menampilkan fitur-fitur yang diimplementasikan pada CMS tersebut. Setiap fitur memiliki halamannya sendiri dimana pada setiap halaman, pengguna dapat melihat semua data, membuka halaman detail untuk setiap data, dan melakukan proses *create, read, update, delete* (CRUD) pada data yang disimpan pada fitur tersebut.

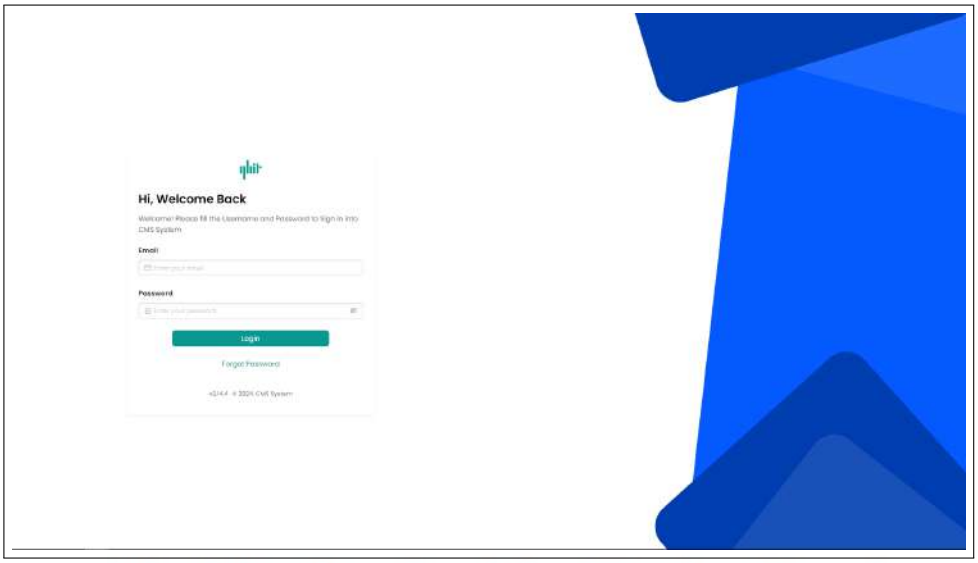
Setiap fitur memiliki tabel masing-masing yang digunakan untuk menyimpan data yang digunakan pada fitur tersebut, contohnya fitur *user* terhubung dengan tabel *user* pada database dan fitur produk terhubung dengan tabel produk pada database. Semua fitur yang sudah pernah dibuat sebelumnya disimpan di dalam *library-library* privat dan diimplementasikan pada *template* CMS melalui *import*. Dengan adanya *template* CMS, setiap proyek hanya perlu menambahkan atau mengurangi fitur yang sudah ada pada *template* CMS.

Arsitektur dari *template* CMS terdiri dari tabel-tabel yang ada pada database, repositori privat perusahaan, modul-modul/fitur-fitur yang di-*import* dari repositori privat, *backend* CMS yang berisi *API Gateway* dan modul-modul/fitur-fitur yang diimplementasikan secara lokal, dan *frontend/user facing interface* yang berisi berbagai halaman yang menampilkan data serta menjadi sarana pengguna untuk mengelola data. Kerangka arsitektur dari aplikasi web CMS dapat dilihat pada Gambar 3.1.




Gambar 3.1. Arsitektur aplikasi web CMS

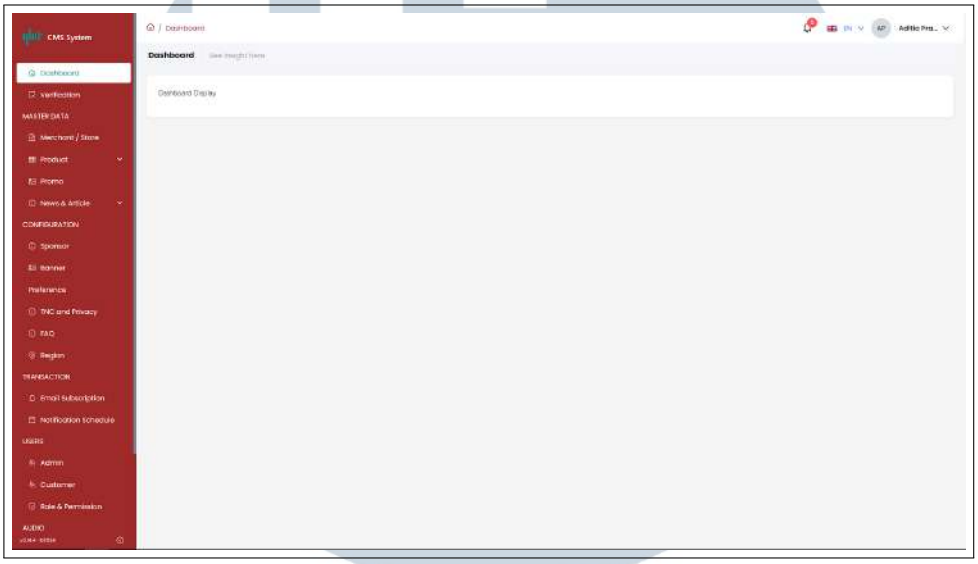
Berikut adalah beberapa contoh tampilan dari *template CMS*. Gambar 3.2 adalah tampilan dari halaman *login* dari aplikasi CMS. Pada halaman tersebut, pengguna dapat memasukkan *email* dan *password*/kata sandi untuk mengakses fitur-fitur pengelolaan data yang diimplementasikan pada aplikasi CMS.




Gambar 3.2. Tampilan halaman login aplikasi web CMS

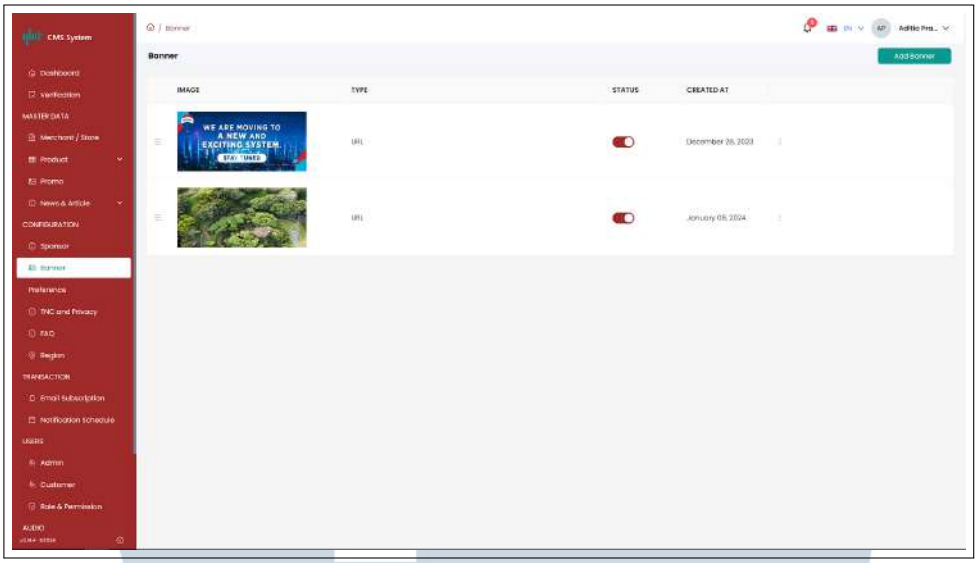
Gambar 3.3 adalah tampilan dasbor dari aplikasi CMS. Halaman dasbor ini adalah halaman pertama yang ditampilkan kepada pengguna setelah pegguna

melakukan *login*. Pada *template* CMS, halaman ini tidak menampilkan data apapun, tetapi dapat diatur untuk menampilkan data tertentu sesuai dengan kebutuhan dan permintaan dari klien. Pada setiap halaman, terdapat *sidebar* yang mengarahkan pengguna ke berbagai halaman fitur/modul yang diimplementasikan di dalam aplikasi CMS.



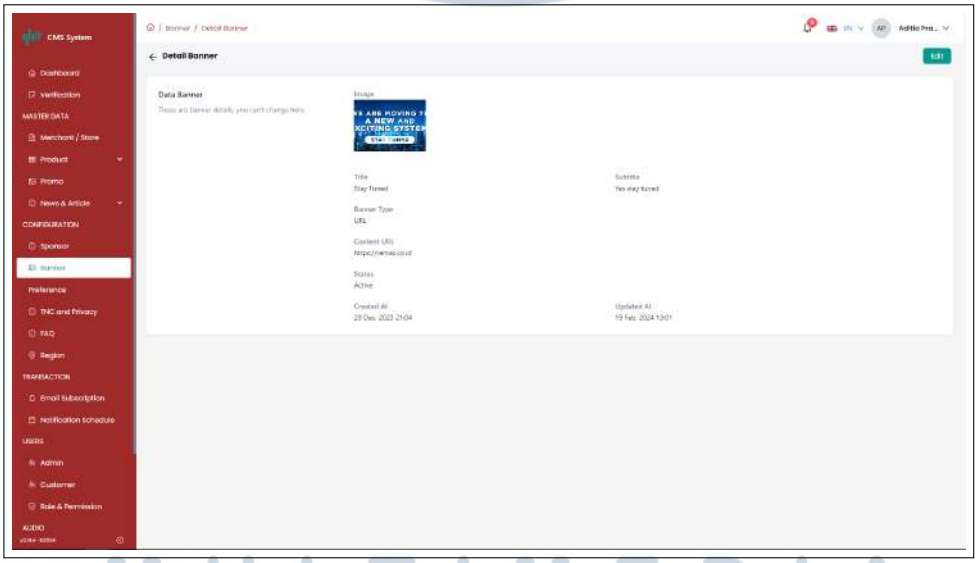

Gambar 3.3. Tampilan halaman dasbor aplikasi web CMS

Pada Gambar 3.4, ditampilkan salah satu fitur/modul yang diimplementasikan pada *template* CMS, yaitu banner. Fitur ini adalah salah satu fitur yang di-*import* dari repositori privat perusahaan. Pada halaman daftar banner yang ditampilkan di Gambar 3.4, ditampilkan daftar data yang tersimpan di dalam tabel database banner.




Gambar 3.4. Tampilan halaman daftar banner aplikasi web CMS

Data yang ditampilkan pada halaman daftar banner hanya sebagian dari setiap data yang tersimpan. Informasi dari data yang lebih lengkap dapat dilihat pada halaman detail dari setiap banner seperti pada gambar 3.5.




Gambar 3.5. Tampilan halaman login aplikasi web CMS

Setiap fitur yang dapat diakses melalui *sidebar* pada *template* CMS diimplementasikan secara modul lokal atau *import* dari repositori privat perusahaan. Untuk fitur yang di-*import*, fitur-fitur tersebut di-*import* melalui *package libs-[nama fitur]*. Contohnya untuk fitur sponsor, di-*import* melalui *package lins-sponsor* dan

fitur *role* di-*import* melalui *package libs-role*.

### A. *libs-sponsor*

Tugas pertama dalam program kerja magang adalah membuat fitur baru di *backend* dan *frontend* CMS lokal dan memindahkan fitur tersebut ke repositori *library* privat. Fitur yang dibuat adalah fitur sponsor yang menyimpan dan mengelola data yang berisi *link website* sponsor, nama sponsor, dan logo sponsor. Pengelolaan data tersebut diimplementasikan dengan proses CRUD, dilengkapi dengan *find one, find all, upload image,* dan *edit/update image* yang dapat diakses melalui *endpoint* yang dibuat pada *backend* CMS.

Untuk menyimpan data sponsor, diperlukan tabel ”sponsors” pada database dengan desain pada gambar 3.6.

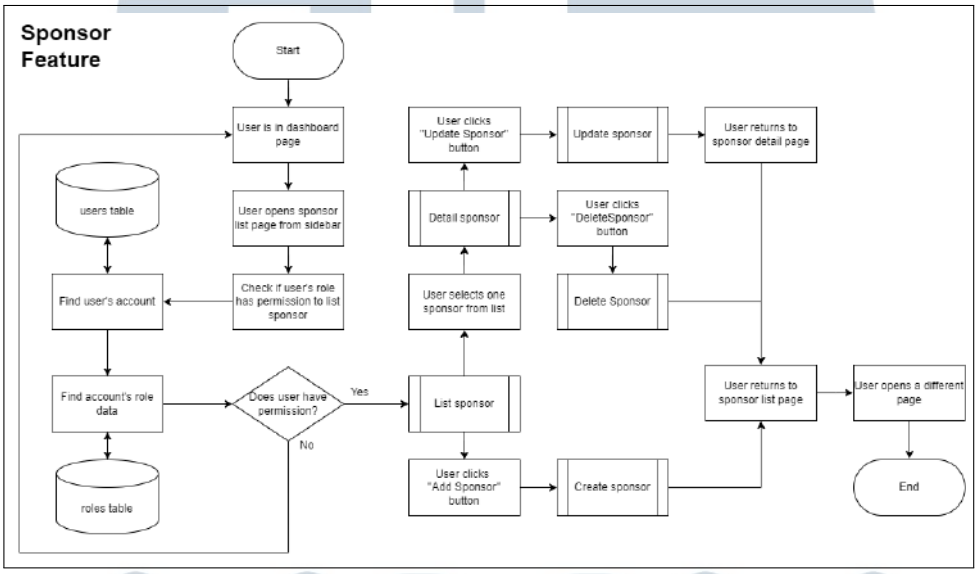
Tabel sponsors

| Kolom | Tipe Data |
|---|---|
| sponsorId | string |
| sponsorName | string |
| sponsorUrl | string |
| isPublished | boolean |
| createdAt | date |
| updatedAt | date |

Gambar 3.6. Desain tabel sponsors

Setiap fitur dalam CMS berdasar pada proses *create, read, update, and delete* (CRUD) untuk mengelola semua data yang disimpan pada fitur tersebut. Aktivitas yang dapat dilakukan pengguna dipaparkan dalam gambar 3.7. Halaman daftar sponsor dapat diakses melalui *sidebar* yang ada pada tampilan CMS. Pengguna hanya dapat melihat daftar sponsor jika akun yang digunakan pengguna memiliki *role* dengan *permission* untuk mengakses data sponsor. *Frontend* CMS akan meminta *backend* untuk melakukan cek terhadap akun yang sedang dipakai oleh pengguna ke tabel users pada database dan melihat daftar *permission* yang

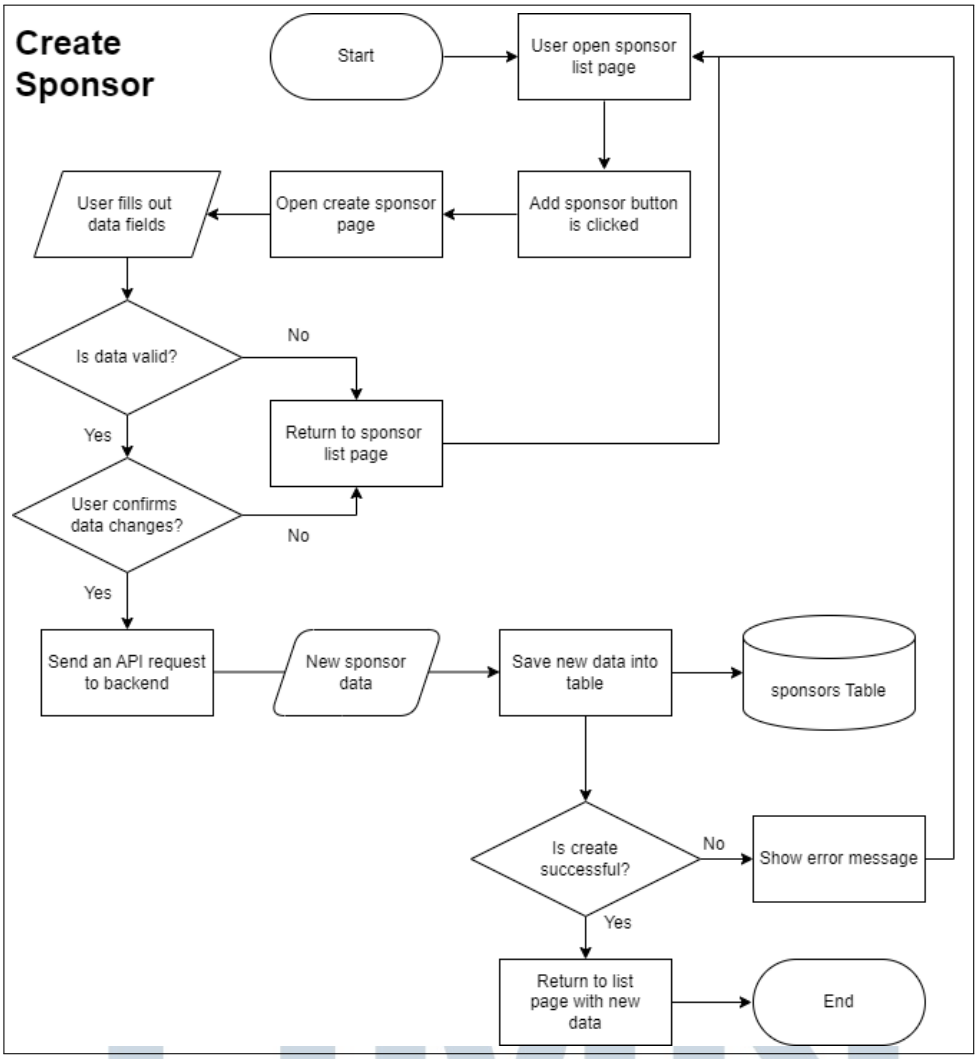
diberikan kepada *role* akun pengguna. Apabila *role* dari akun user memiliki *permission* yang sesuai, maka halaman daftar sponsor akan ditampilkan. Ketika halaman daftar sponsor dibuka, CMS akan melakukan proses ”List Sponsor” untuk menampilkan semua data sponsor. Pengguna dapat menambahkan data sponsor baru atau memilih data sponsor tertentu untuk membuka halaman detail sponsor dan melihat informasi lebih lengkap dari data sponsor tersebut. Pada halaman detail sponsor, pengguna dapat melakukan pembaruan atau penghapusan terhadap data sponsor yang ditampilkan pada halaman tersebut.




Gambar 3.7. Master flowchart fitur sponsor

Gambar 3.8 adalah *flowchart* cara kerja penambahan data baru pada fitur sponsor. Ketika pengguna menekan tombol ”Add New Sponsor” pada *frontend* CMS, mereka akan diarahkan ke halaman pembuatan sponsor baru. Di halaman tersebut, terdapat beberapa kolom input yang diperlukan untuk membuat sponsor baru. Setelah pengguna melakukan pengisian, mereka harus melakukan konfirmasi terhadap pembuatan sponsor baru dengan data yang mereka sudah isi. Jika pembuatan sudah dikonfirmasi, *frontend* CMS akan mengirimkan *API request* dengan metode POST ke *backend* dengan URL ”/sponsors/create” dan *body* yang berisi data baru dari pengguna. *Backend* akan melakukan penyimpanan data sponsor baru ke tabel ”sponsors” pada database, penyimpanan ini hanya akan berhasil jika data yang dikirimkan dalam *body* API request sesuai dengan desain tabel. Jika penyimpanan berhasil, maka pengguna akan diarahkan ke halaman daftar sponsor untuk menunjukkan bahwa data sponsor yang mereka isi berhasil tersimpan

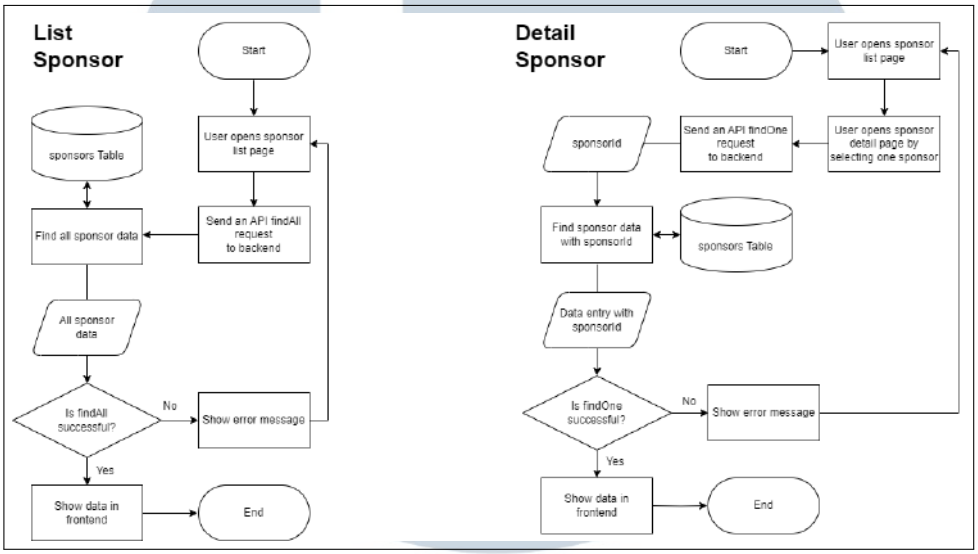
ke dalam database.




Gambar 3.8. Flowchart create new sponsor

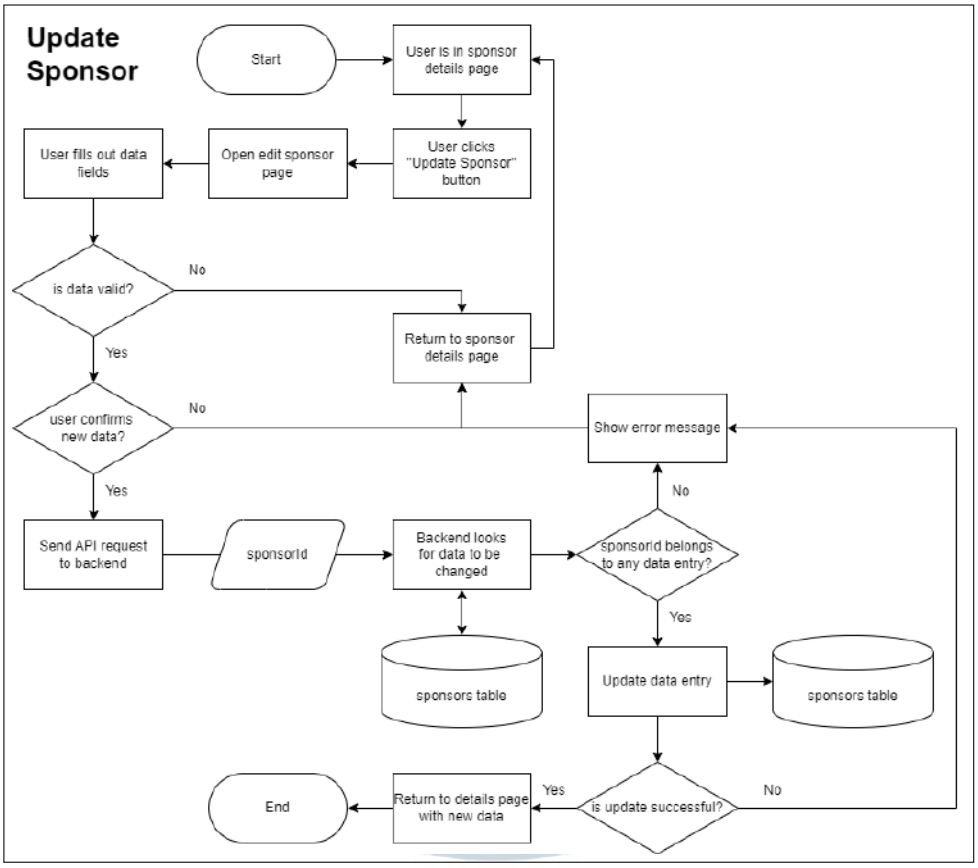
Untuk menampilkan semua data sponsor pada halaman daftar sponsor dan data lengkap setiap sponsor pada halaman detail dari masing-masing sponsor, diperlukan proses pengambilan data dari database seperti yang dipaparkan pada gambar 3.9. Ketika pengguna membuka halaman daftar sponsor, *frontend* CMS akan mengirimkan *API request* dengan metode GET ke *backend* dengan URL ”/sponsors”. *Backend* akan mengambil semua data yang tersimpan di dalam tabel ”sponsors” pada database dan mengirimkan semua data yang didapat ke *frontend* untuk ditampilkan di halaman daftar sponsor. Pengguna dapat memilih salah satu sponsor dari daftar sponsor yang ditampilkan untuk melihat informasi lebih detail dari data sponsor tersebut. Ketika pengguna memilih salah satu sponsor, *frontend*

akan mengirimkan *API request* dengan metode GET ke *backend* dengan URL ”sponsors/[sponsorId]”. [sponsorId] pada URL akan digantikan dengan sponsorId dari sponsor yang dipilih oleh pengguna. Setelah *backend* menerima *API request* dengan URL tersebut, *backend* akan mencari satu data yang memiliki sponsorId yang sama dengan sponsorId yang terdapat pada URL dan mengirimkan data tersebut ke *frontend* untuk ditampilkan pada halaman detail sponsor.



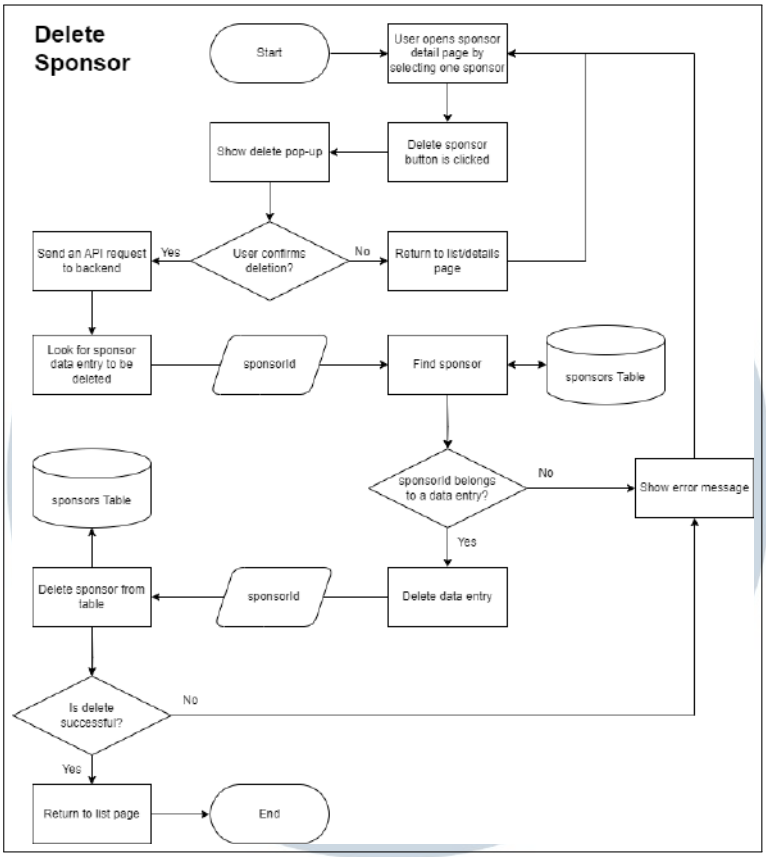

Gambar 3.9. Flowchart list sponsor dan detail sponsor

Pada halaman detail sponsor, pengguna dapat melakukan modifikasi terhadap data yang ditampilkan dengan menekan tombol ”Edit Sponsor”. Ketika tombol tersebut ditekan pengguna, mereka akan diarahkan ke halaman edit sponsor. Halaman edit sponsor memiliki kolom-kolom input yang sama dengan halaman pembuatan sponsor baru. Pengguna dapat mengisi data yang mereka ingin ubah dari data sponsor yang mereka pilih dan melakukan konfirmasi terhadap perubahan data. Jika perubahan data sudah dikonfirmasi, *frontend* CMS akan mengirimkan *API request* dengan metode PATCH ke *backend* dengan URL ”/sponsors/[sponsorId]/edit” dan *body* yang berisi data baru dengan perubahan dari pengguna. Sebelum menyimpan data yang sudah diubah, *backend* akan melakukan pencarian menggunakan sponsorId terlebih dahulu seperti pada pencarian detail data sponsor. Jika data sponsor yang ingin diubah berhasil ditemukan, *backend* akan menyimpan perubahan data pada baris data yang ditemukan menggunakan sponsorId. Setelah perubahan data berhasil, pengguna akan diarahkan ke halaman detail sponsor untuk memperlihatkan bahwa perubahan daa yang mereka lakukan berhasil tersimpan.




Gambar 3.10. Flowchart update sponsor

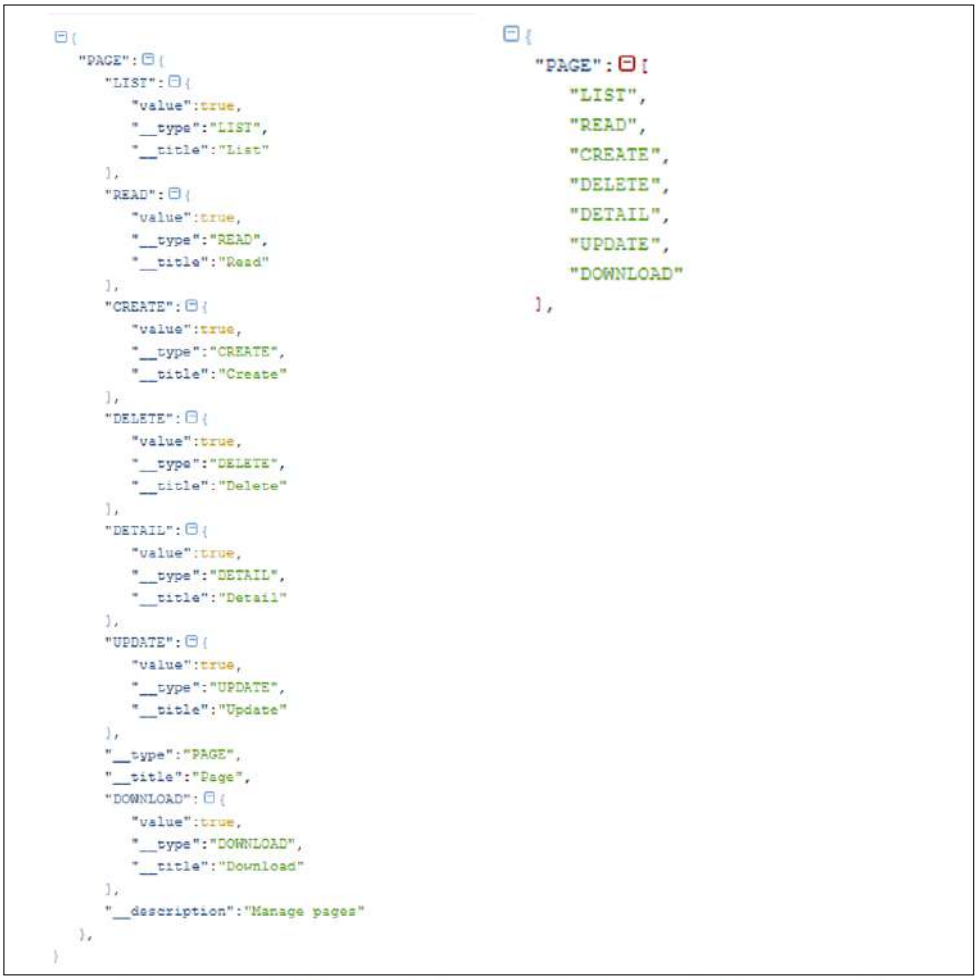
Selain melakukan perubahan data, pengguna juga dapat menghapus data sponsor melalui halaman detail maupun halaman daftar sponsor. Pengguna dapat menghapus sponsor dengan menekan tombol ”Delete Sponsor”. Ketika pengguna ingin menghapus sponsor, *frontend* akan menampilkan *pop-up* untuk meminta konfirmasi user sebelum mengirim *request* ke *backend*. Jika user memberikan konfirmasi, *frontend* akan mengirimkan *API request* dengan metode DELETE ke *backend* dengan URL ”sponsors/[sponsorId]”. Seperti pada pencarian detail data dan update data, *backend* akan mencari baris data yang memiliki sponsorId yang sama dengan sponsorId pada URL dan menghapus baris data tersebut dari tabel sponsors pada database. Setelah penghapusan berhasil, pengguna akan diarahakan ke halaman daftar sponsor untuk menampilkan bahwa sponsor yang mereka pilih berhasil dihapus dari database.




Gambar 3.11. Flowchart delete sponsor

## B. *libs-role*

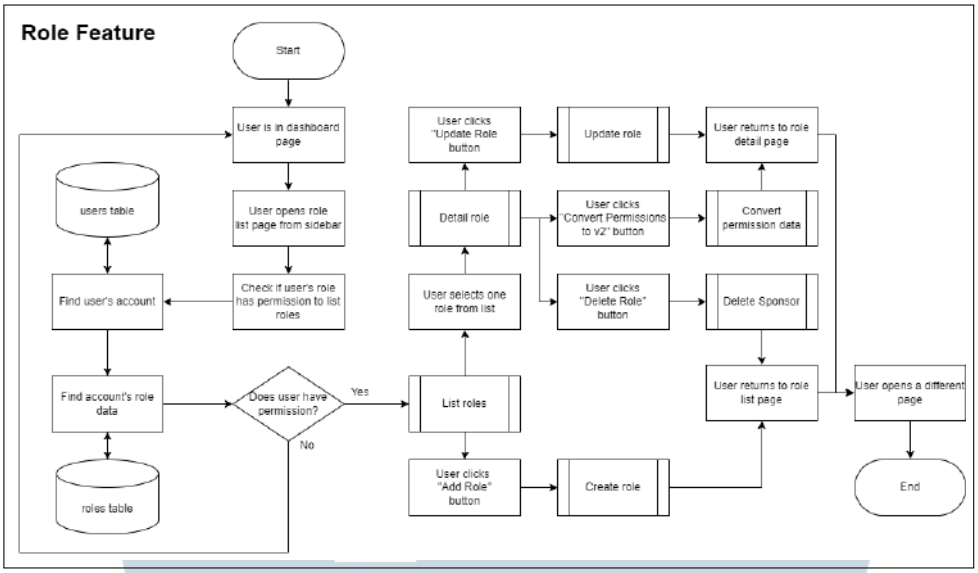
*Library libs-role* adalah *library* privat yang digunakan untuk mengatur *role* pengguna di CMS, semua *permission* untuk setiap fitur yang diimplementasikan pada aplikasi CMS, dan *permission* yang diberikan kepada setiap *role*. Perusahaan ingin merubah bentuk penyimpanan data *permission* agar lebih mudah dibaca dan diakses ketika melakukan pengecekan *permission*. Perbedaan antara kedua bentuk *permission* dapat dilihat pada gambar 3.12 untuk fitur page.




Gambar 3.12. Perbedaan bentuk penyimpanan data *permission* antara versi lama (kiri) dan versi baru (kanan) pada fitur page

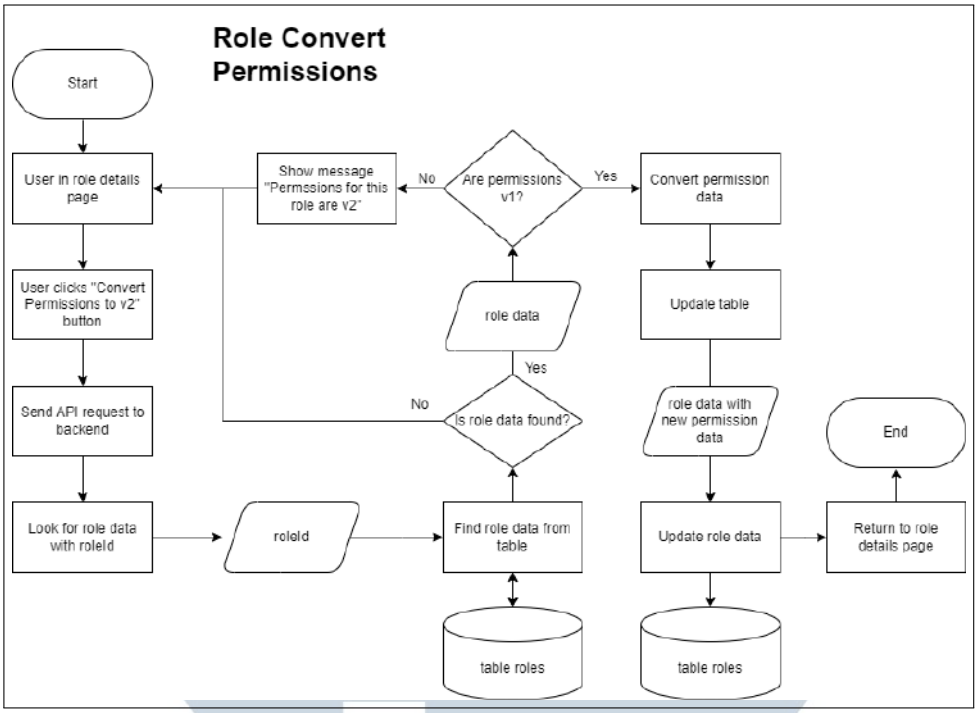
Untuk menghindari melakukan konversi secara manual pada database untuk setiap proyek yang menggunakan bentuk penyimpanan data *permission* yang lama, perlu dibuat sebuah *function* untuk melakukan konversi bentuk data secara otomatis. Konversi dilakukan ketika *endpoint* konversi di fitur *role* dipanggil melalui *http request*. Untuk mencegah terjadinya *error* jika *function* menerima data *permission* yang sudah dikonversi, diimplementasikan cek versi bentuk data *permission*.

Pada gambar 3.13, dapat dilihat aktivitas yang dapat dilakukan pengguna pada fitur *role*. Fitur ini juga memiliki proses CRUD untuk setiap *role* yang dibuat pada aplikasi CMS.




Gambar 3.13. Master flowchart fitur role

Gambar 3.14 menunjukkan proses pengguna dalam melakukan konversi bentuk data *permission* dalam sebuah *role*. Untuk melakukan konversi, pengguna harus membuka halaman detail *role* dan menekan tombol ”Convert Permissions to v2”. *Frontend* CMS akan mengirimkan *API request* ke *backend* dengan URL ”role/[roleId]/convert” dan tipe *request* POST. *Backend* akan melakukan pencarian data *role* yang dituju ke tabel roles dan mengecek apakah bentuk *permission* pada *role* tersebut termasuk bentuk yang lama atau yang baru. Jika bentuk *permission* sudah yang terbaru, CMS akan menampilkan pesan *error* ”Permissions for this role are already v2”. Jika belum, *backend* akan melakukan konversi menggunakan potongan kode pada gambar 3.15 dan menyimpan data *role* dengan bentuk data *permission* yang baru ke tabel roles.




Gambar 3.14. Flowchart konversi *role permissions*

```
Object.keys(item.permissions).forEach((objectName: string) => {
 const objectData = item.permissions[objectName] as {
  [key: string]: { value: boolean; __type: string; __title: string }
 };
 const newKeys = Object.keys(objectData)
  .filter((key) => objectData[key].value == true)
  .map((key) => key.toUpperCase());
 groupedResult[objectName] = newKeys;
 newPermissions = groupedResult;
})
```

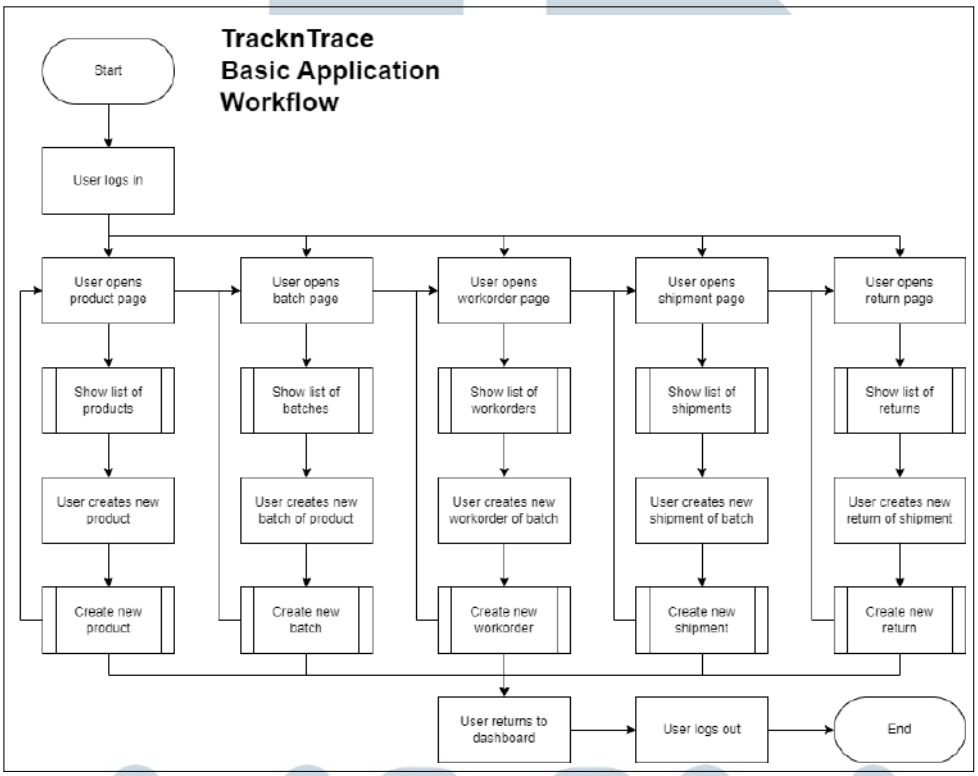
Gambar 3.15. Potongan kode fungsi konversi

### 3.3.2 Proyek TracknTrace

Proyek TracknTrace adalah proyek kerja sama PT. Vanz Inovatif Teknologi dengan klien untuk memantau produksi dan distribusi produk. TracknTrace adalah aplikasi CMS berbasis web yang menyimpan data produk, *batch*, *workorder*, *shipment*, dan *return* sebagai sistem pendataan dari berbagai proses dalam produksi dan distribusi produk obat-obatan.

*Flowchart* operasional dasar dari pengunaan aplikasi TracknTrace dapat dilihat pada gambar 3.16. Setelah melewati proses autentikasi berupa login ke akun

yang terdaftar pada CMS, pengguna dapat membuka berbagai halaman pendataan melalui *sidebar*. Halaman-halaman yang dapat diakses pengguna adalah halaman daftar produk, halaman daftar *batch*, halaman daftar *workorder*, halaman daftar *shipment*, dan halaman daftar *return*.




Gambar 3.16. *Flowchart* operasional dasar dari aplikasi TracknTrace

Tahap-tahap operasional dasar pengunaan aplikasi CMS TracknTrace untuk melakukaan pendataan adalah sebagai berikut:

1. Login ke akun yang terdaftar pada CMS TracknTrace
2. Buka halaman daftar produk untuk menambahkan produk baru yang siap diproduksi
3. Buka halaman daftar *batch* untuk membuat *batch* baru
4. Buka halaman detail dari *batch* yang baru dibuat dan tambahkan produk yang akan diproduksi dalam batch tersebut, produk yang ditambahkan dapat berupa produk yang baru dibuat ataupun produk yang sudah dibuat sebelumnya
5. Pada halaman detail *batch*, buat *workorder* untuk *batch tersebut*

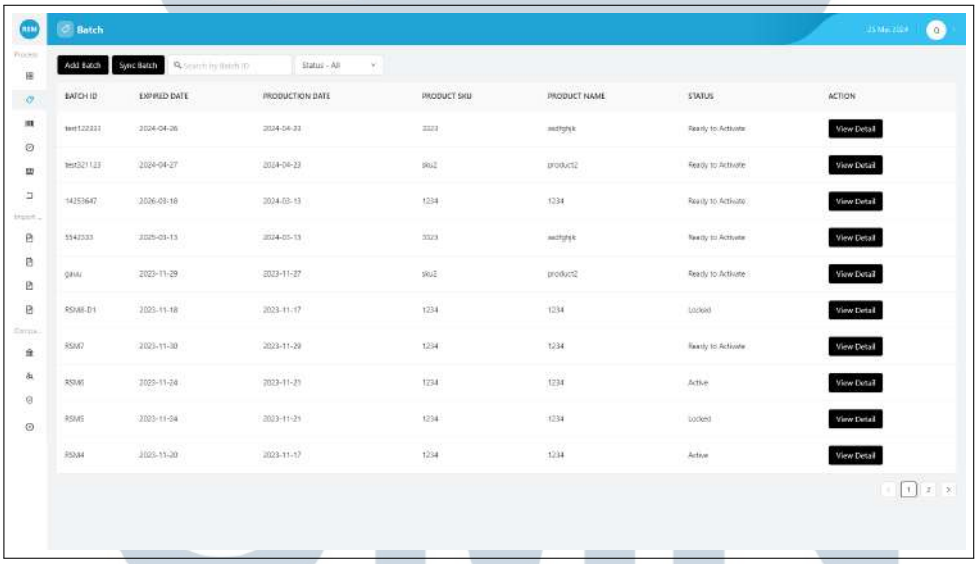
6. *Workorder* juga dapat ditambahkan melalui halaman daftar *workorder* dan menambahkan *batch* pada *pop-up* input data *workorder*

7. Buka halaman *shipment* dan buat *shipment* baru

8. Buka halaman detail *shipment* yang sudah dibuat dan tambahkan *batch-batch* yang akan dikirimkan dalam *shipment* tersebut

9. Jika *shipment* harus dikembalikan, buka halaman *return* dan buat data *return*.

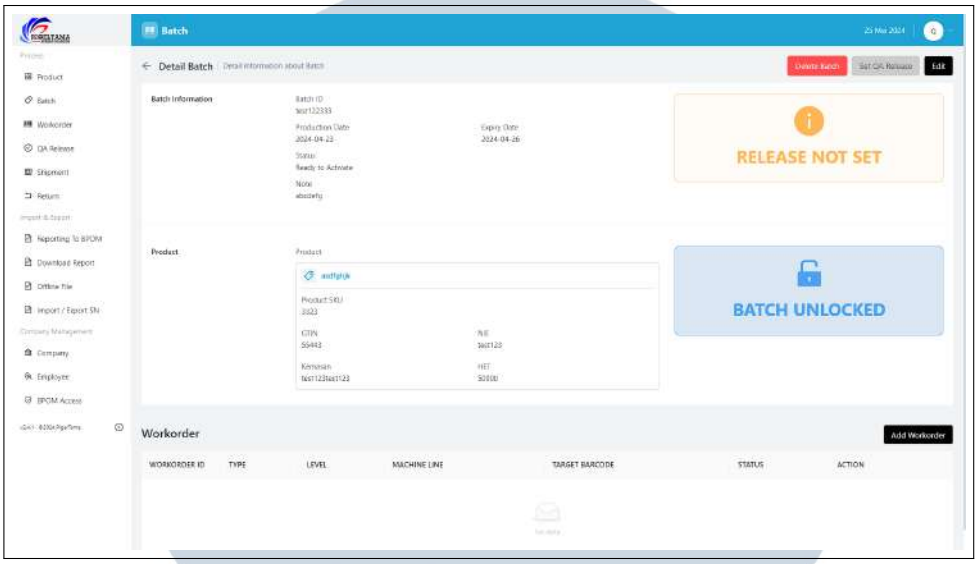
Pada setiap halaman daftar, ditampilkan data yang disimpan pada setiap fitur. Fitur *batch* akan digunakan sebagai contoh tampilan dari aplikasi CMS. Pada halaman daftar *batch* yang dapat dilihat pata gambar 3.17, ditampilkan *batch ID*, tanggal kadaluwarsa, tanggal produksi, *Stock Keeping Unit* (SKU) produk, nama produk, dan status *batch*.




Gambar 3.17. Tampilan halaman daftar batch

Jika pengguna ingin melihat informasi lebih detail mengenai salah satu *batch*, disediakan tombol ”View Detail” pada kolom ”Action” untuk membuka halaman detail dari *batch* tersebut. Contoh dari salah satu halaman detail *batch* dapat dilihat pada gambar 3.18. Di dalam halaman detail *batch*, dapat dilihat data dari produk yang ditambahkan di dalam *batch* dan data *workorder* yang menyimpan informasi dari produksi *batch*. Pengguna dapat melakukan perubahan data seperti mengganti produk dan mengatur tanggal produksi serta kadaluwarsa dengan menekan tombol ”Edit Batch”, menghapus data *batch* dengan menekan

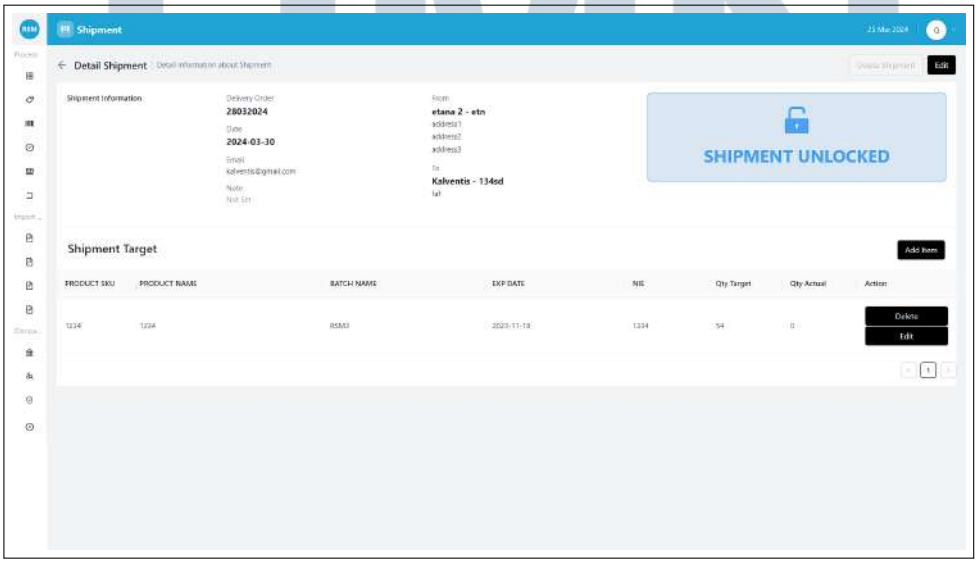
tombol ”Delete Batch”, dan menambahkan data *workorder* baru untuk *batch* yang sedang mereka buka.




Gambar 3.18. Tampilan halaman daftar *batch*

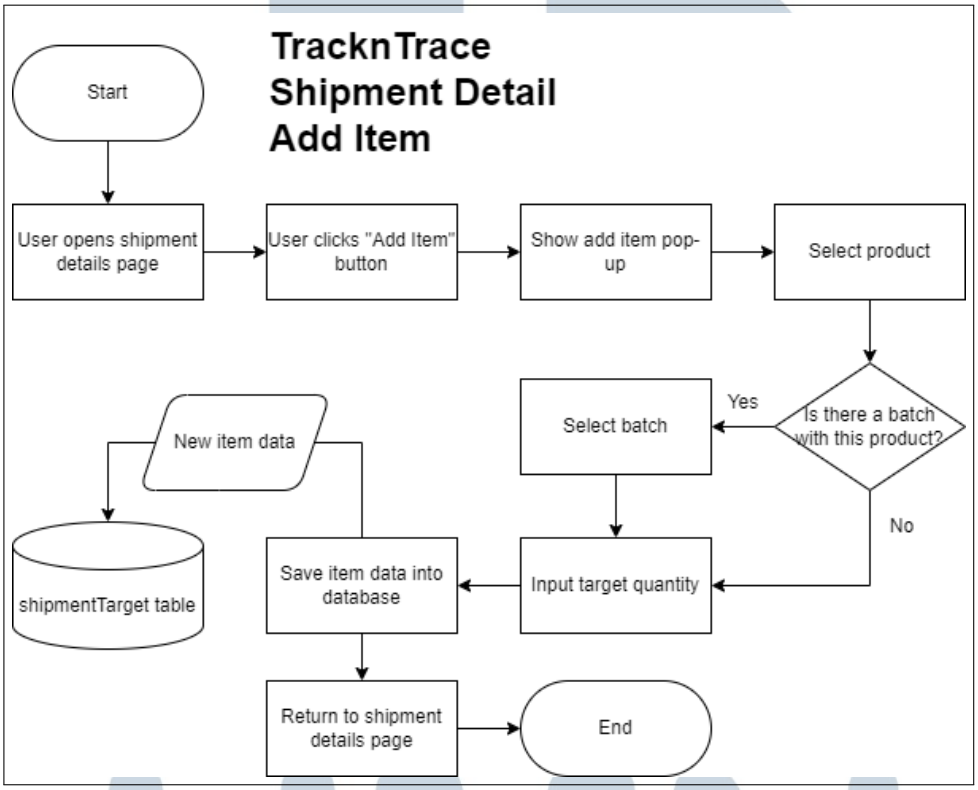
## A. Fitur *Shipment*

Fitur *shipment* pada CMS TracknTrace digunakan sebagai pendataan dan riwayat/rekam jejak dari semua pengiriman *batch* yang dilakukan oleh klien. Tampilan dari halaman detail *shipment* dapat dilihat pada gambar 3.19.



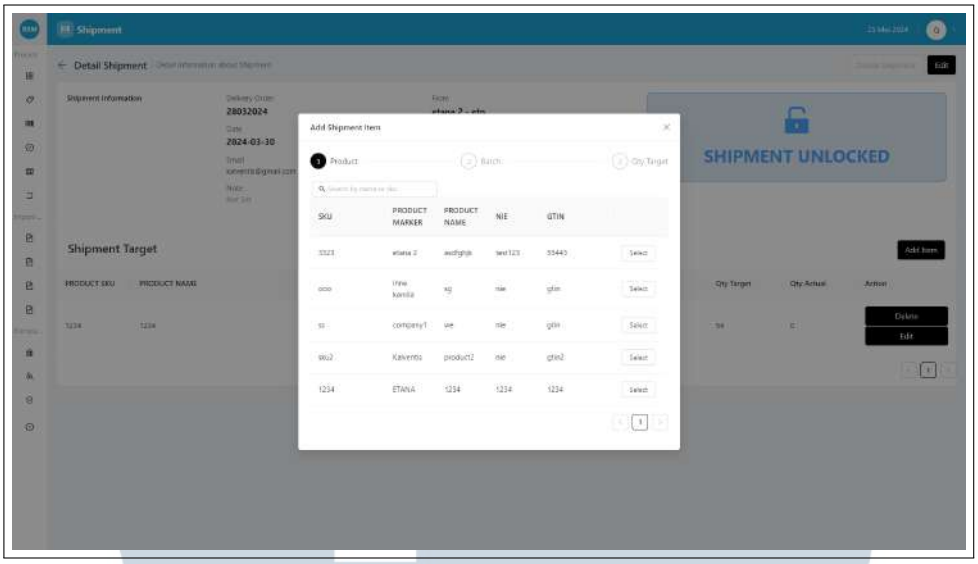

Gambar 3.19. Tampilan halaman detail *shipment*

Di dalam halaman detail *shipment*, pengguna dapat melakukan beberapa aktivitas seperti update data *shipment*, menghapus data *shipment*, dan menambahkan *batch* atau produk yang dikirimkan dalam *shipment*. *Flowchart* aktivitas penambahan *batch* atau produk ditampilkan pada gambar 3.21.




Gambar 3.20. *Flowchart* aktivitas halaman detail *shipment*

Pengguna dapat menambahkan *batch* atau produk ke *shipment* dengan menekan tombol ”Add Item”. Setelah itu, CMS akan memunculkan *pop-up* seperti pada gambar 3.21 yang meminta user untuk memilih produk, memilih *batch* jika *batch* memiliki produk yang dipilih, dan memasukkan jumlah barang/*target quantity*.




Gambar 3.21. *Pop-up* ”Add Item” pada halaman detail *shipment*

Isu yang dialami oleh klien pada fitur ini adalah:

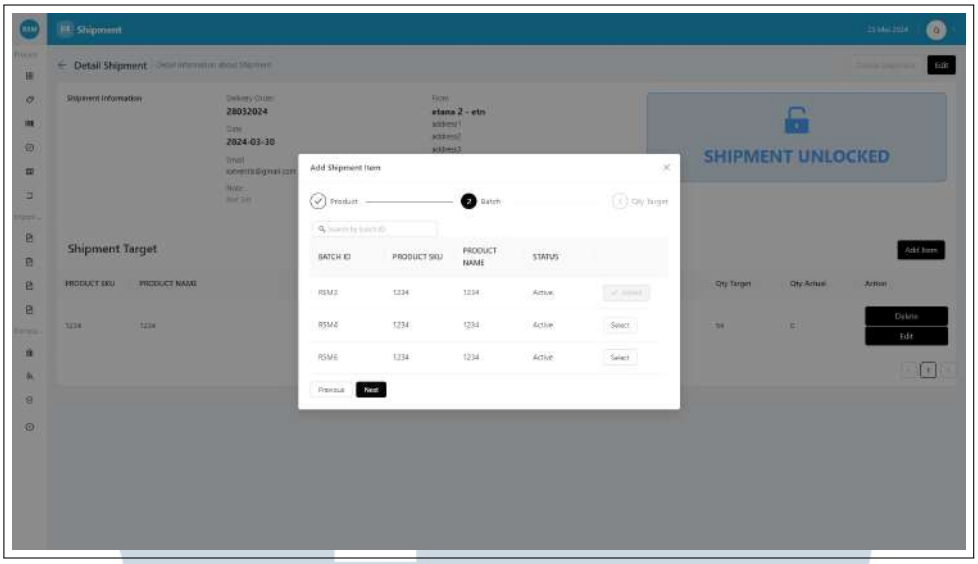
1. Saat menambahkan *batch* atau produk ke data *shipment*, pengguna dapat menambahkan *batch* duplikat pada data *shipment*, mengakibatkan anomali data dimana *batch* yang sama terlihat dikirimkan dua kali dalam satu *shipment*.

2. Tidak ada *search bar* pada tahap pemilihan *batch*, sehingga *batch* yang ingin dipilih sulit ditemukan jika jumlah data *batch* di dalam sistem sudah sangat banyak.

3. Data yang dipilih/dimasukkan pada *pop-up* tidak hilang ketika *pop-up* ditutup, sehingga data yang sebelumnya tidak dikonfirmasi oleh pengguna muncul pada saat *pop-up* dibuka kembali.

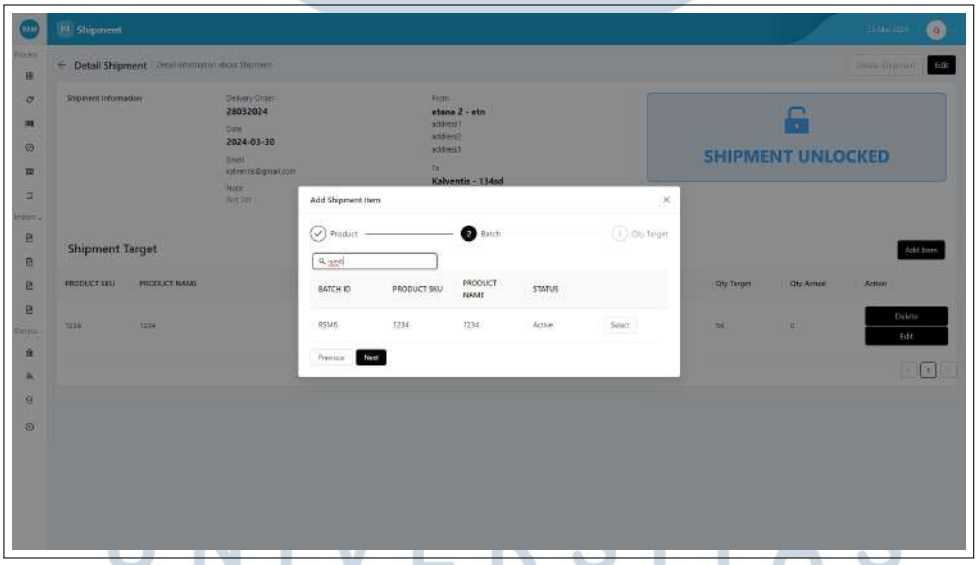
4. Tidak ada pengecekan data ”vDeliveryOrder” ketika pembuatan *shipment*, sehingga data tersebut dapat duplikat antara satu *shipment* dengan *shipment* lainnya. Data ”vDeliveryOrder” digunakan sebagai kode identifikasi unik dari setiap data *shipment*.

Isu pertama dapat diselesaikan dengan mengimplementasikan sebuah filter dimana jika data *shipment* sudah memiliki suatu *batch*, maka pada *pop-up* ”Add Item”, tombol untuk memilih *batch* tersebut akan di-*disable* dan diberi tanda centang beserta tulisan ”Added” seperti pada gambar 3.22.




Gambar 3.22. Tombol *batch* di-*disable* dalam *pop-up* ”Add Item”

Untuk menyelesaikan isu kedua, penambahan *search bar* pada *pop-up* tahap pemilihan *batch* dapat dilihat pada gambar 3.23.

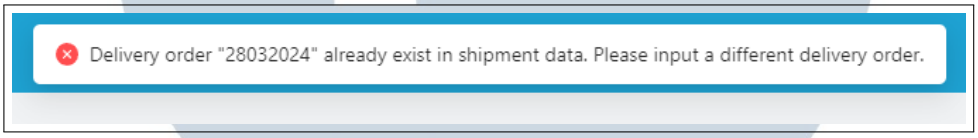



Gambar 3.23. Fungsi pencarian *batch* pada *pop-up* ”Add Item”

Isu ketiga dapat diselesaikan dengan menghapus isi dari setiap variabel data yang akan dikirimkan ke *backend* ketika *pop-up* ”Add Item” ditutup oleh pengguna. Isi dari setiap variabel data harus dihapus secara kode karena variabel-variabel tersebut beserta isinya tidak akan terhapus jika pengguna tidak berpindah ke halaman lain atau melakukan *refresh* halaman detail *shipment*, sehingga, tanpa

adanya penghapusan secara kode, *pop-up* akan menyimpan data sebelumnya yang telah dimasukkan oleh pengguna dan tidak dikonfirmasi untuk ditambahkan ke tabel database.

Isu keempat dapat diselesaikan dengan melakukan cek input data vDeliveryOrder pada *backend* proyek ketika menerima *API request* dari *frontend*. *Backend* akan mencari data *shipment* pada tabel shipments yang memiliki data vDeliveryOrder yang sama dengan data vDeliveryOrder yang diterima dari *frontend*. Jika ditemukan data yang sudah ada dan memiliki vDeliveryOrder yang sama, pembuatan data *shipment* akan dibatalkan dan *frontend* akan menampilkan pesan *error* berisi ”Delivery order already exist in shipment data. Please input a different delivery order.” seperti pada gambar 3.24

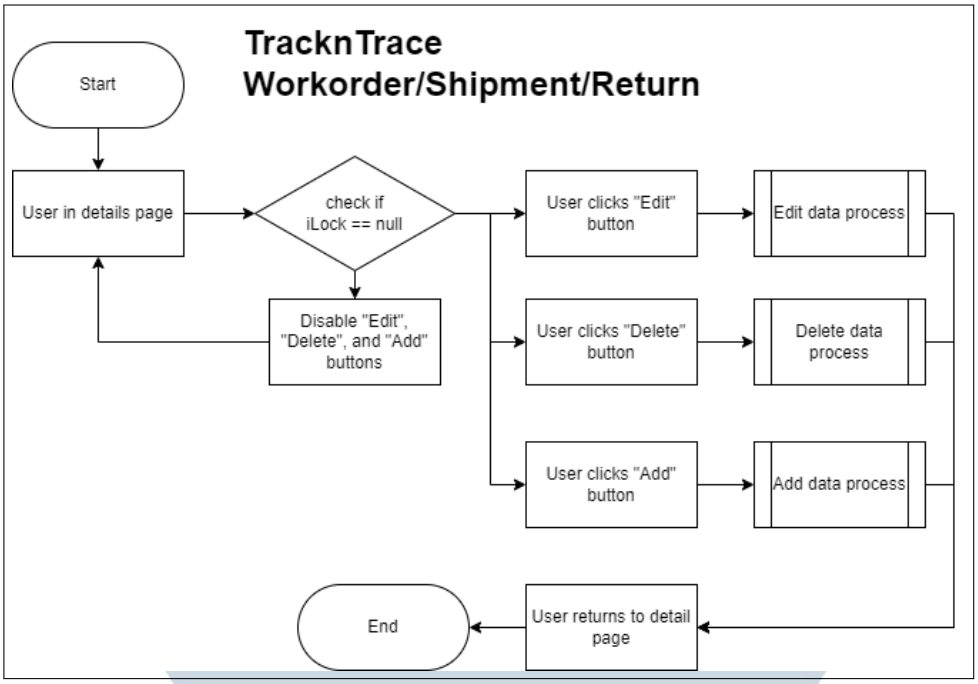



Gambar 3.24. Pesan *error* jika data vDeliveryOrder duplikat

## B. Implementasi iLock pada fitur *workorder*, *shipment*, dan *return*

Karena perusahaan klien beroperasi di bidang produksi dan distribusi, pendataan menjadi salah satu alat penting dalam melacak akar masalah jika terjadi gangguan dalam aktivitas operasional. Untuk mencegah perubahan data dalam CMS TracknTrace, klien ingin mengimplementasikan sistem pengunci agar tidak terjadi perbedaan data antara realita dan data pada CMS. Klien ingin mencegah terjadinya perubahan atau penghapusan data *workorder* apabila *workorder* sedang proses produksi dan perubahan atau penghapusan data *shipment* dan *return* apabila *shipment* maupun *return* sedang dalam perjalanan.

Sistem pengunci ini diimplementasikan dengan menambahkan kolom ”iLock” pada tabel *workorder*, *shipment*, dan *return*. Kolom ”iLock” dapat tidak diisi/null dan memiliki isi berupa angka. Data di dalam kolom ”iLock” hanya dapat diubah melalui aplikasi eksternal milik klien yang terhubung dengan database yang sama dengan aplikasi CMS TracknTrace. CMS TracknTrace akan me-*disable* tombol ”Edit”, ”Delete”, dan ”Add” pada halaman detail data jika kolom ”iLock” pada data tersebut tidak null. *Flowchart* cara kerja sistem penguncian ini dapat dilihat pada gambar 3.25.




Gambar 3.25. *Flowchart* sistem penguncian fitur menggunakan ”iLock”

### 3.3.3 WhatsApp Business API & Instagram Graph API

Untuk memenuhi proyeksi kebutuhan pasar, perusahaan ingin mempersiapkan integrasi media sosial ke *template* CMS. Integrasi media sosial pertama adalah WhatsApp Business API yang digunakan untuk melakukan *broadcast* pesan melalui CMS. Integrasi kedua adalah Instagram Graph API yang digunakan untuk mendapatkan data akun Instagram profesional, baik akun bisnis maupun akun *creator*.

Kedua integrasi API tersebut memerlukan *Meta Application* yang dapat dibuat dengan mendaftarkan akun Facebook sebagai *Meta Developer* dan membuat *Meta Application* dengan tipe *business* pada halaman *Meta for Developers*. Untuk menggunakan WhatsApp Business API dan Instagram Graph API, ada beberapa produk Meta yang harus ditambahkan pada *Meta Application* yang sudah dibuat. Produk Meta tersebut adalah WhatsApp, Facebook Login, dan Instagram Graph API.

WhatsApp Business API dapat digunakan setelah menambahkan produk WhatsApp ke *Meta Application*. Pemanggilan API dilakukan melalui sebuah *Uniform Resource Locator* (URL) yang ditampilkan pada halaman *API Setup* di dalam *Meta Application*. Di halaman yang sama juga dapat terlihat token

akses sementara, *test number*, daftar nomor penerima, *ID test number*, dan *ID WhatsApp Business Account*. Token akses sementara dan *ID test number* harus dikirimkan bersama dengan URL API di dalam parameter *http request* sebagai metode autentikasi. Selain itu, perlu dikirimkan juga *body* yang berisi informasi mengenai *request* API yang dilakukan seperti isi dari pesan yang dikirimkan, tipe penerima, dan tipe pesan yang dikirimkan. Jika *http request* terhadap URL API tersebut berhasil, maka akan dikirimkan pesan dari *test number* kepada nomor penerima yang didaftarkan pada *Meta Application* dengan isi sesuai dengan jenis dan konten dari pesan yang dicantumkan pada *body http request*.

Instagram Graph API memerlukan akun Instagram profesional yang dapat bertipe bisnis maupun *creator* yang memiliki Facebook Page dan sudah terhubung dengan akun Facebook *Meta Developer*. Setelah itu diperlukan juga beberapa *permission* yang dapat ditambahkan pada *Meta Application* melalui halaman *Graph API Explorer*. Pada halaman *Graph API Explorer* juga terdapat token akses sementara yang harus dicantumkan pada parameter *http request* setiap kali melakukan pemanggilan URL API. Ketika akun Instagram sudah terhubung dan *permission* yang diperlukan sudah ditambahkan ke *Meta Application*, perlu dilakukan beberapa *http request* terhadap sejumlah URL API sebelum mendapatkan data dari akun Instagram tersebut seperti jumlah *follower*, jumlah *post*, jumlah *like*, *comment*, dan media URL dari setiap *post*.

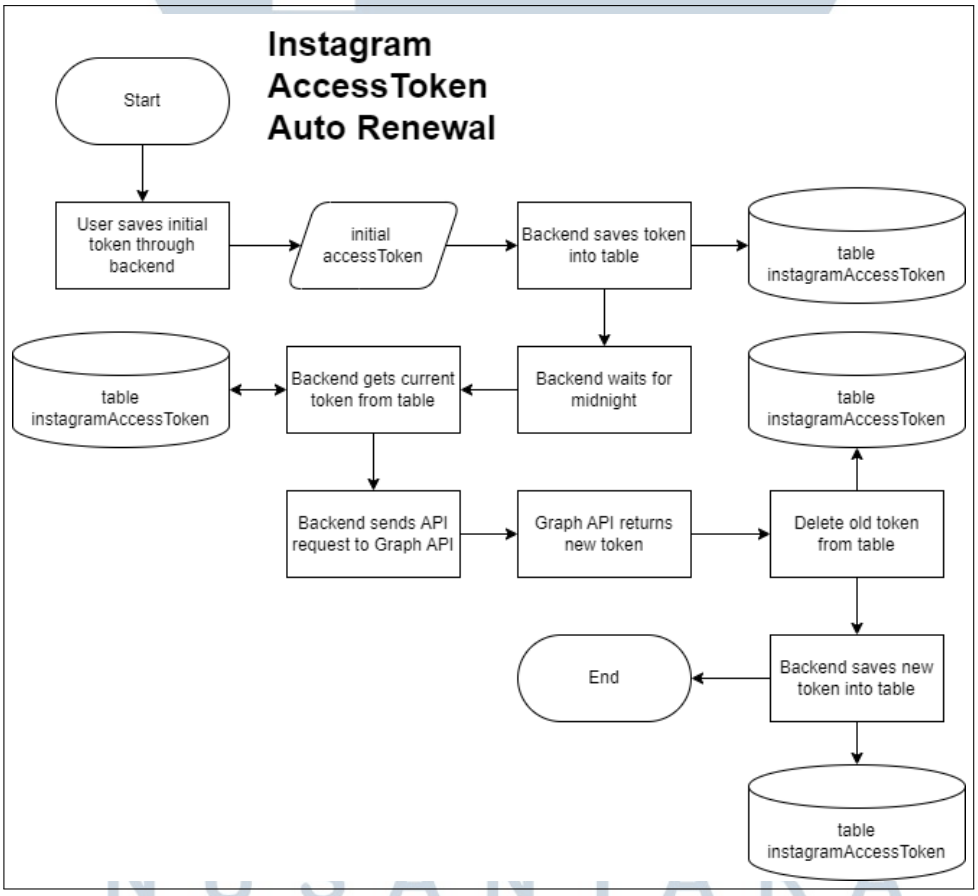
URL Graph API yang digunakan untuk mendapatkan data akun Instagram adalah:

1. ”/me/accounts” untuk mendapatkan data ”pageId”
2. ”/[pageId]?fields=instagram_business_account” untuk mendapatkan data ”instagramId”
3. ”/[instagramId]?fields=username” untuk mendapatkan data ”username”
4. ”/[instagramId]?fields=business_discovery.username([username])” untuk mendapatkan data dari akun Instagram

Setiap pengiriman *API request* ke URL Graph API memerlukan token akses yang aktif dan setiap *Meta Application* hanya dapat memiliki 1 token akses yang aktif. Token akses yang didapatkan dari halaman *Graph API Explorer* hanya memiliki durasi aktif selama 1 jam. Token ini dapat ditukarkan menjadi *long-lived token* yang masa aktifnya menjadi 2 bulan. Pertukaran ini dapat dilakukan

pada halaman *Graph API Debug* atau mengirimkan *API request* ke URL Graph API disertai dengan data app Id, app *secret*, dan token lama yang ingin ditukarkan menjadi *long-lived token*.

Untuk menghindari membuka halaman *Graph API Debug* setiap 2 bulan sekali untuk memperbarui token akses pada CMS, diimplementasikan *scheduler* yang akan melakukan *API request* secara otomatis. Pengguna hanya perlu memasukkan token akses pertama yang diambil dari halaman *Graph API Explorer* ke tabel instagramAccessToken di dalam database. Setelah itu, *backend* akan secara otomatis mengambil token pertama tersebut untuk mendapatkan *long-lived token*, menghapus token akses sebelumnya dari tabel, dan menyimpan token akses yang baru ke tabel instagramAccessToken. Proses pengambilan, penghapusan, dan penyimpanan ini dilakukan setiap hari jam 12 malam. Proses ini digambarkan pada *flowchart* yang dapat dilihat pada gambar 3.26



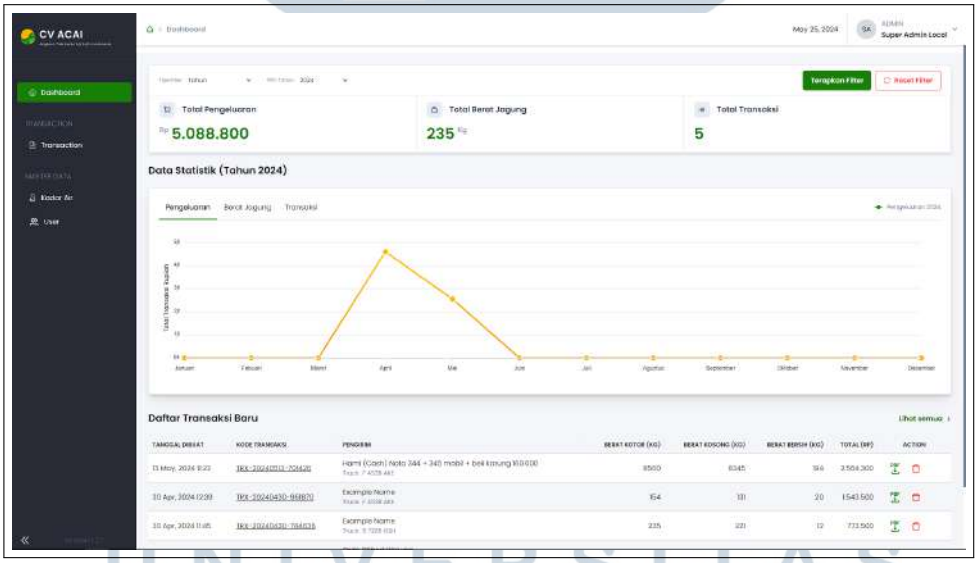

Gambar 3.26. *Flowchart* penjadwalan pembaruan akses token

WhatsApp Business API dan Instagram Graph API diintegrasikan pada

*backend* CMS dan pengiriman pesan melalui WhatsApp dapat dilakukan melalui *frontend* CMS serta data dari akun Instagram yang terhubung dapat disimpan dan ditampilkan pada *frontend* CMS. Kedua integrasi API tersebut dibuat menjadi *library* privat terpisah untuk masing-masing API dan dapat di-*import* ke dalam proyek yang membutuhkan kedua API tersebut.

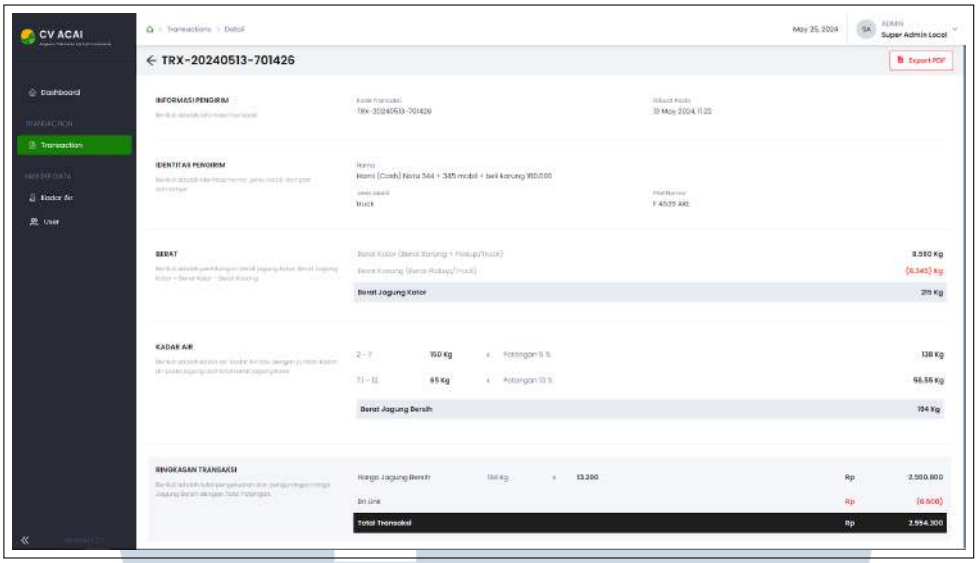
### 3.3.4 Proyek Gudang Acai

Proyek Gudang Acai adalah proyek CMS dengan klien yang memiliki perusahaan distribusi jagung. Dalam proyek ini, CMS digunakan sebagai rekapitulasi pengeluaran perusahaan untuk membeli jagung dari petani. Pengeluaran perusahaan dapat dilihat pada halaman dashboard dan dapat diatur untuk menampilkan pengeluaran total dalam bulan, tahun, atau rentang waktu tertentu. Pada dasbor CMS juga ditampilkan daftar transaksi beserta dengan beberapa detail untuk setiap transaksi antara perusahaan dengan petani. Tampilan dasbor dari CMS Gudang Acai dapat dilihat pada gambar 3.27.



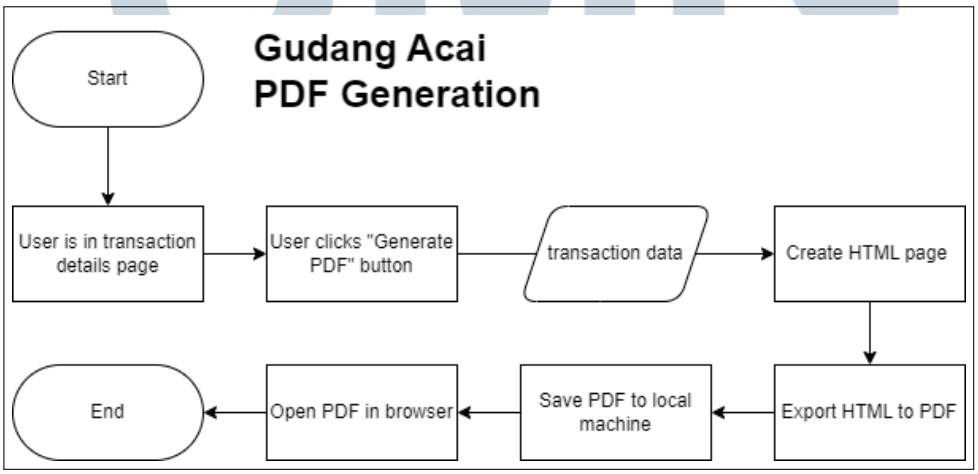

Gambar 3.27. Tampilan dasbor CMS Gudang Acai

Pengguna CMS Gudang Acai juga dapat melihat detail lebih lengkap dari setiap data transaksi. Halaman detail dapat dilihat pada gambar 3.28. Halaman tersebut dapat diakses dengan menekan kode transaksi pada daftar transaksi yang ditampilkan pada halaman dasbor maupun halaman daftar transaksi.



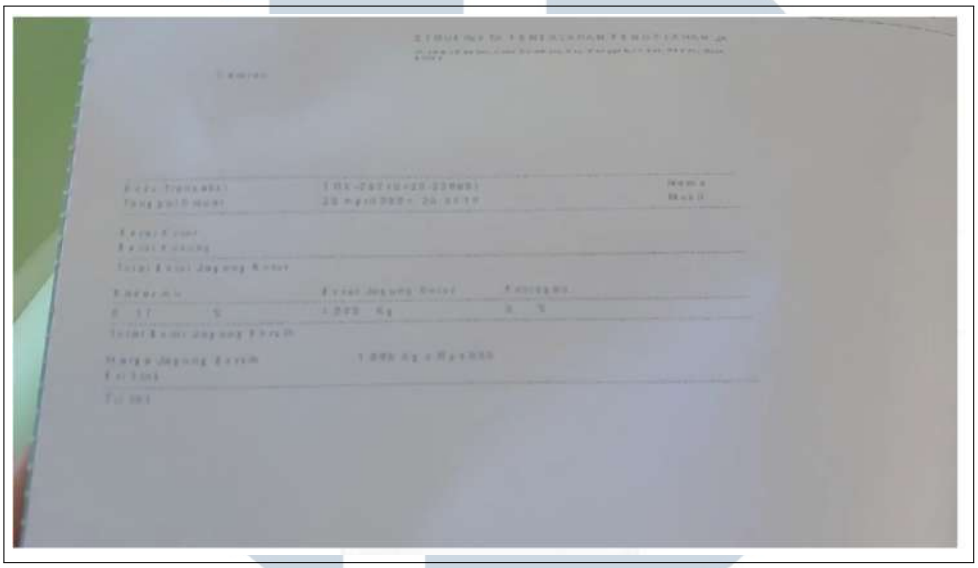

Gambar 3.28. Tampilan detail transaksi CMS Gudang Acai

Pada halaman detail transaksi dan pada kolom ”Action” pada dashboard, terdapat tombol untuk mengekspor data transaksi ke sebuah dokumen *Portable Document Format* (PDF). Dokumen ini akan disimpan langsung ke perangkat perusahaan untuk keperluan cetak kuitansi dan akan diberikan kepada petani setelah melakukan transaksi. *Flowchart* dari pembuatan dokumen PDF melalui CMS dapat dilihat pada gambar 3.29. Mesin cetak yang digunakan oleh klien adalah *Dot Matrix Printer*. Jenis mesin cetak tersebut mencetak dokumen menggunakan kepala cetak yang berupa sejumlah jarum berbentuk matriks. Ketika mencetak, mesin cetak ini akan menekan matriks jarum tersebut dan menghasilkan sejumlah titik-titik yang berbentuk karakter atau gambar yang diinginkan.




Gambar 3.29. *Flowchart* pembuatan PDF pada CMS

Perusahaan klien mengalami masalah ketika mereka mencetak kuitansi yang telah di-*generate* oleh CMS Gudang Acai. Hasil cetak yang klien dapatkan tidak sesuai ukuran cetak dan tulisan tidak dapat dibaca sama sekali. Contoh hasil cetak klien yang gagal dapat dilihat pada gambar 3.30



Gambar 3.30. Hasil cetak yang tidak sesuai dari dokumen yang dibuat CMS

Penyelesaian masalah hasil cetak yang tidak sesuai dimulai dengan mengidentifikasi faktor-faktor dalam kode generasi dokumen PDF yang dapat menyebabkan salah cetak pada mesin cetak *dot matrix*. Beberapa faktor yang mungkin menimbulkan permasalahan adalah:

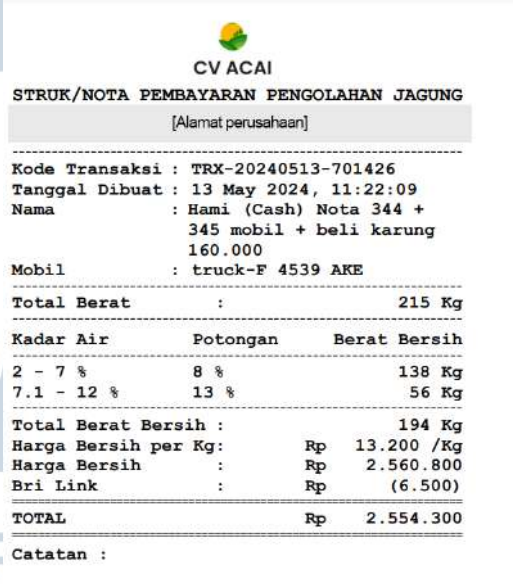
1. Jenis *font* yang digunakan pada dokumen tidak dapat dicetak dengan jelas pada mesin cetak *dot matrix*.
2. Ukuran tulisan yang terlalu kecil.
3. Ketebalan huruf/*font weight* yang kurang tebal.
4. Resolusi atau ukuran dokumen tidak sesuai dengan kertas yang digunakan pada mesin cetak.
5. *Software* yang digunakan untuk melakukan pencetakan tidak sesuai dengan mesin cetak.

Dari beberapa faktor yang diidentifikasikan di atas, diimplementasikan beberapa solusi untuk mengoptimisasi generasi dokumen PDF agar tulisan pada hasil cetak dapat lebih terbaca. Solusi-solusi tersebut adalah:

1. Merubah jenis *font* menjadi ”Courier” agar lebih mudah dicetak oleh mesin cetak *dot matrix*.
2. Memperbesar ukuran *font* dan membuat ukuran *font* menyesuaikan dengan ukuran dokumen agar tidak perlu merubah ukuran *font* satu per satu ketika ingin merubah ukuran dokumen.
3. Menggunakan jenis *font* ”Courier Bold” untuk membuat tulisan lebih tebal.
4. Menyesuaikan ukuran dokumen generasi PDF dengan ukuran kertas yang digunakan pada mesin cetak dan merubah orientasi dokumen dari *landscape* menjadi *portrait*.
5. Klien awalnya menggunakan *print PDF* dari *web browser* dan mengganti menjadi menggunakan *software* cetak lain.

Perbedaan dan hasil dari semua implementasi solusi generasi dokumen PDF dapat dilihat dari gambar 3.31 ke gambar 3.32.

Gambar 3.31. Hasil generasi PDF sebelum perubahan




Gambar 3.32. Hasil generasi PDF setelah perubahan

Berdasarkan informasi dari klien, hasil generasi PDF sudah dapat tercetak dengan jelas dan mudah terbaca oleh mesin cetak klien.

### 3.3.5 Proyek Mayora

Proyek Mayora adalah proyek CMS untuk memenuhi kebutuhan klien dalam *monitoring* kegiatan produksi dan pelaporan semua kendala yang terjadi saat kegiatan produksi. Proyek ini sudah diselesaikan, di-*deploy* ke server produksi klien, dan digunakan di perusahaan klien pada tahun 2021. Namun pada bulan Mei tahun 2024, staf dari perusahaan klien menemukan beberapa anomali data yang terjadi pada CMS Mayora. Sayangnya, perusahaan tidak menyimpan kode proyek versi manapun di repositori perusahaan, sehingga harus mengambil kode yang ada pada server produksi klien dan melakukan *setup* ulang di perangkat lokal untuk memastikan bahwa kode yang didapatkan/di-*recover* dapat berjalan sebelum kemunculan anomali data dapat dicari akar permasalahannya.

*Setup* untuk CMS proyek Mayora dimulai dengan menginisialisasi semua tabel pada database yang akan terhubung dengan *backend* CMS. Berdasarkan kode yang di-*recover* dari server produksi klien, terdapat 2 cara untuk melakukan inisialisasi tabel:

1. Melakukan *migration* dengan menjalankan perintah ”npx sequelize-cli db:migrate” pada *terminal*. Perintah tersebut akan membaca file-file yang ada di dalam folder ”resources/migrations/sql” dan membuat tabel beserta kolom-kolomnya sesuai dengan kode yang ada pada setiap file *migration*.

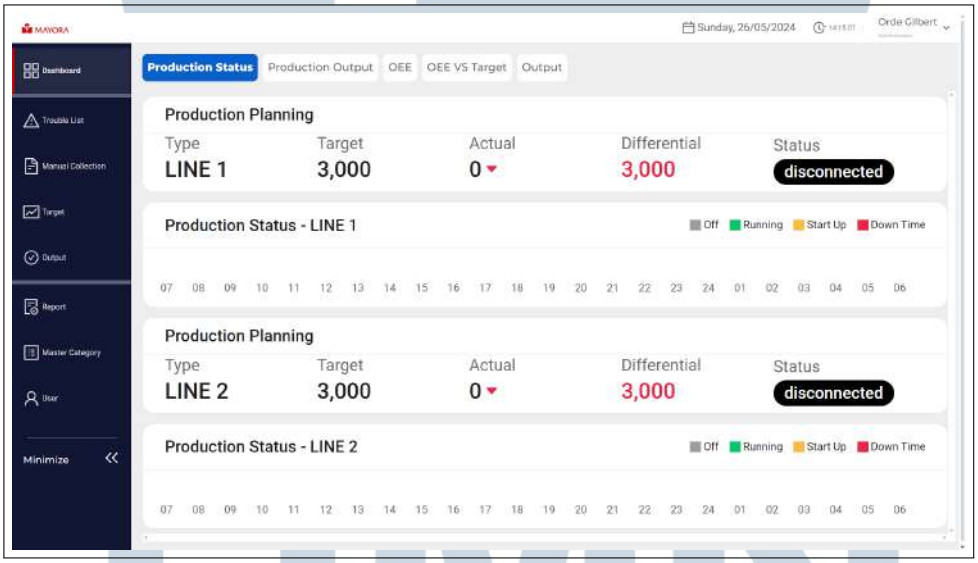
2. Pada kode yang di-*recover*, terdapat satu file teks (TXT) yang berisi *script* SQL yang dapat dijalankan sebagai query pada database melalui aplikasi SQL Server Management Studio (SSMS). *Script* tersebut akan membuat semua tabel beserta kolom-kolomnya pada database yang di-*query*.

Kedua cara inisialisasi tersebut dilakukan pada dua database yang berbeda, tujuannya adalah untuk melihat apakah ada perbedaan jumlah tabel, jumlah kolom pada setiap tabel, nama tabel, nama kolom pada setiap tabel, dan tipe data dari setiap kolom antara kedua cara inisialisasi. Setelah dilakukan cek secara manual, tidak ada perbedaan antara kedua cara inisialisasi. Apabila database sudah diinisialisasi, perlu juga dilakukan *seeding* untuk memasukkan data awal agar tidak terjadi *error* karena data kosong ketika CMS dijalankan.

*Seeding* dilakukan dengan menjalankan perintah ”npm sequelize-cli db:seed:all” pada *terminal*. Perintah ini akan membaca semua file yang ada di dalam folder ”resources/sql/seeders” dan menjalankan semua *script* SQL yang ada di dalam setiap file untuk mengisi tabel-tabel yang memerlukan data sebelum CMS

dapat dijalankan. Salah satu isu yang dihadapi sebelum CMS dapat dijalankan adalah tidak adanya akun *superadmin* yang dibuat melalui *seeding*. Isu ini dapat diselesaikan dengan menjalankan *backend* CMS dan mengirim *POST API request* dengan URL ”/user” dan *body* berisi data akun yang juga diberikan ”roleId” yang sesuai dengan *role superadmin*. *API request* ini dilakukan melalui aplikasi ”Postman”.

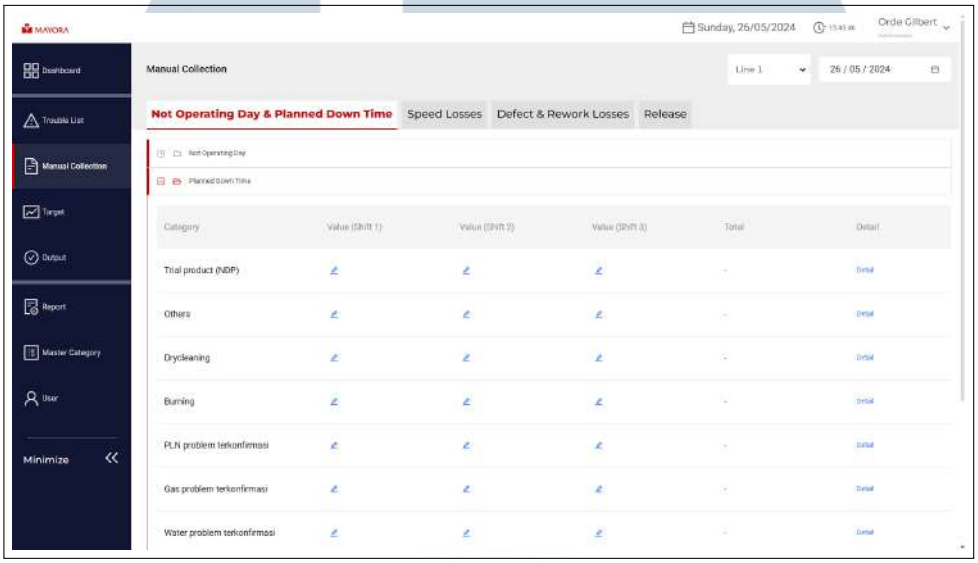
Setelah akun *superadmin* berhasil dibuat dan dapat melakukan login dengan akun tersebut, CMS Mayora dapat diakses dan ditampilkan dasbor CMS seperti pada gambar 3.33. Pada dasbor CMS, ditampilkan target produksi, hasil produksi, perbedaan antara target dan hasil, dan status dari baris mesin, serta terdapat graf waktu yang menunjukkan aktivitas setiap baris mesin setiap jam.




Gambar 3.33. Dasbor CMS Mayora

Anomali data yang dialami oleh staf klien terdapat pada halaman ”Manual Collections” dimana terdapat kategori-kategori dan subkategori yang berisi berbagai kejadian *downtime* mesin dan penyebab produk defektif yang terjadi dalam kegiatan produksi. Tampilan dari halaman ”Manual Collections” dan beberapa kategori, subkategori, dan kejadian *downtime* mesin serta penyebab produk defektif dapat dilihat pada gambar 3.37. Staf perusahaan klien dapat memasukkan input ke kolom data sesuai dengan kendala yang terjadi, berapa lama atau berapa banyak produk, dan pada *shift* berapa kendala tersebut terjadi. Contohnya, jika terjadi kendala ”PLN problem terkonfirmasi” pada jam 12 siang hingga jam 2 siang, staf perusahaan akan mengisi ”120 menit” pada baris ”PLN problem terkonfirmasi”

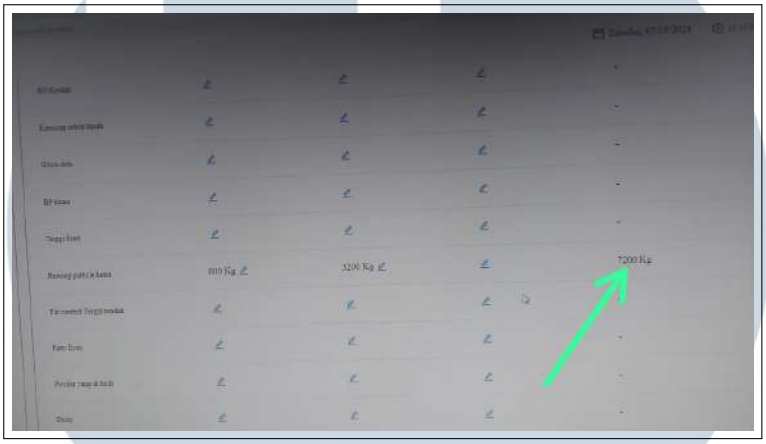
dan kolom ”Shift 2”. Data kendala ini disimpan berdasarkan hari kendala tersebut terjadi. Contohnya jika kendala ”PLN problem terkonfirmasi” yang sebelumnya terjadi pada tanggal 12 Mei 2024, database juga akan menyimpan tanggal terjadinya kendala dan data lamanya terjadi kendala tersebut hanya akan muncul jika tanggal pada halaman ”Manual Collections” di kanan atas layar diubah menjadi tanggal 12/05/2024




Gambar 3.34. Halaman ”Manual Collections” CMS Mayora

Anomali data yang ditemukan oleh staf perusahaan klien adalah sebagai berikut:

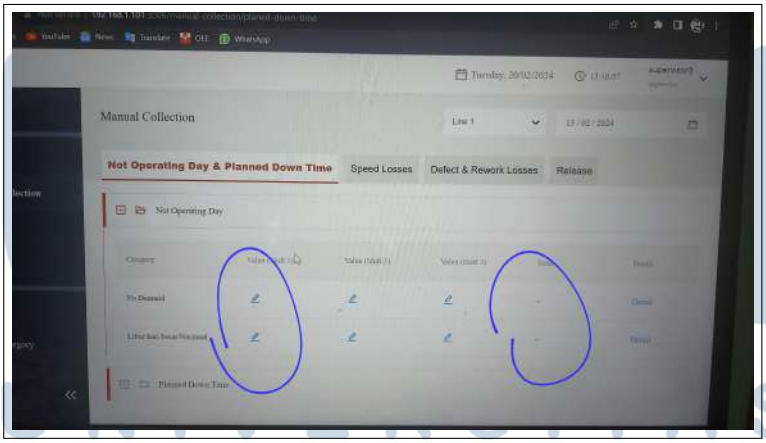
1. Kolom total tidak menunjukkan angka yang benar jika dibandingkan dengan data pada setiap *shift*.



Gambar 3.35. Total tidak sesuai dengan jumlah total pada *shift*

Sumber: Foto dari staf klien

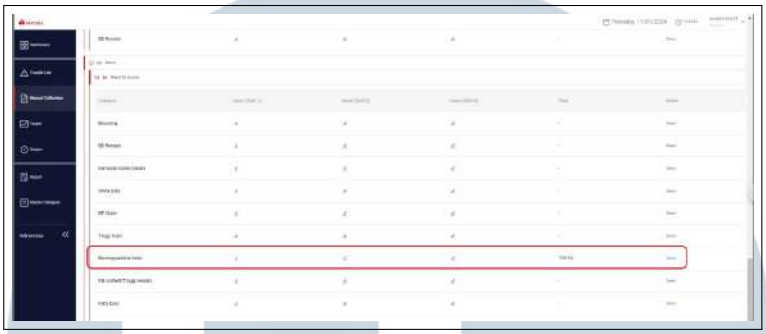
2. Data yang dimasukkan pada beberapa *shift* dapat hilang dengan sendirinya.




Gambar 3.36. Data *shift* yang hilang

Sumber: Foto dari staf klien

3. Kolom total menunjukkan angka meskipun semua data *shift* pada baris tersebut kosong



Gambar 3.37. Total menunjukkan angka, tetapi baris data kosong

Sumber: Foto dari staf klien

Per tanggal 26 Mei 2024, belum ditemukan akar permasalahan maupun solusi apapun dari ketiga anomali data yang ditemukan oleh staf perusahaan klien. Namun, *testing* internal menemukan isu lain dimana *frontend* mengirimkan data tanggal yang salah ke *backend* ketika merubah tanggal pada halaman ”Manual Collections”. Isu ini terjadi karena komponen pemilihan tanggal mengirimkan data berupa angka dalam *milisecond* dan *frontend* akan melakukan perhitungan ulang untuk mendapatkan data tanggal yang dipilih. Perhitungan/kalkulasi ulang yang dilakukan pada *frontend* dapat menyebabkan kesalahan pembulatan/*rounding error* dan mengakibatkan tanggal yang dikirim ke *backend* bertambah atau berkurang 1 hari. Isu ini diselesaikan menggunakan solusi sementara per 26 Mei 2024 dengan membuat komponen pemilihan tanggal khusus yang mengirimkan data tanggal langsung ke *frontend* tanpa diperlukan perhitungan/kalkulasi ulang.

## 3.4 Kendala dan Solusi yang Ditemukan

### 3.4.1 Kendala

Dalam program kerja magang di PT. Vanz Inovatif Teknologi, terdapat beberapa kendala yang dialami. Kendala yang dialami adalah sebagai berikut:

1. Pada projek *template* CMS, *frontend* dan *backend* harus dibuka dan dijalankan melalui *Integrated Development Environment* (IDE) yang terhubung dengan sebuah *instance* dari *Virtual Machine* yang menjalankan *Operating System* Linux. Oleh karena itu, pada perangkat dengan *Operating*

*System* Windows, harus meng-*install* dan menjalankan *Windows Subsystem Linux* (WSL). Aplikasi WSL ini dapat membebani memori perangkat karena membutuhkan *Random Access Memory* (RAM) yang besar. Kebutuhan RAM yang besar dapat menyebabkan perangkat untuk melambat, sehingga menghambat proses pengembangan proyek.

2. Karena setiap proyek tidak hanya dikerjakan oleh satu orang yang sama antar proyek, cara penulisan kode dan pendekatan yang digunakan untuk menjalankan suatu proses sangat berbeda dari satu projek ke projek yang lain. Hal ini menyebabkan kebingungan ketika masuk ke proyek baru untuk melakukan pengembangan ataupun *bug fixing*.

### 3.4.2 Solusi

1. Memastikan tidak ada *background task* yang sedang berjalan dan menambah kapasitas RAM pada perangkat.

2. Diberikan waktu tambahan setiap kali berpindah projek agar dapat memahami kode dalam modul yang akan dikembangkan atau dilakukan *bug fix* dan menggunakan situs web ”TypeScript Playground” untuk melakukan *testing* agar dapat memahami proses pengolahan data dalam modul tersebut.