BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Pelaksanaan magang dilakukan di dalam divisi Retail Digital, yang merupakan pecahan dari divisi Retail, sebagai *web developer*. Di bawah supervisi CTO, divisi Retail Digital dipimpin oleh Kak Gandhi Pranata sebagai *Product Owner* (PO) yang didampingi oleh dua anggota lainnya. Divisi Retail sendiri berfokus pada salah satu layanan yang disediakan IAK, yaitu Whitelabel.

Gambar 3.1. Logo Whitelabel
Sumber: *Website* Whitelabel [7]

Seperti yang sudah disinggung sebelumnya, terdapat dua opsi jenis penjualan bagi setiap *tenant* Whitelabel, yaitu *shop* dan *Payment Point Online Banking* (PPOB). Divisi Retail Digital bertanggung jawab atas seluruh bagian PPOB yang ada pada situs Whitelabel.

3.2 Tugas yang Dilakukan

Selama kerja magang berlangsung, terdapat dua tugas utama yang dilakukan, yang berhubungan dengan automasi pada situs web Whitelabel. Automasi sendiri merupakan teknologi yang dapat melakukan serangkaian proses tertentu secara otomatis, dan tanpa intervensi manusia secara langsung [9], yang dalam hal ini menggunakan suatu algoritma tertentu. Adanya sebuah automasi tidak hanya meningkatkan produktivitas pengguna, namun juga efisiensi dan fleksibilitas [10]. Fitur automasi yang kemudian dikembangkan berupa integrasi antara Whitelabel *tenant* dengan *supplier* A dan sinkronisasi daftar harga yang ada pada *supplier* B dengan Whitelabel.

Proses pengerjaan tugas-tugas tersebut dilakukan menggunakan sistem *cycle* yang diterapkan di dalam perusahaan. Setiap *cycle* berlangsung selama 8 minggu,

dimana 6 minggu digunakan untuk proses pengerjaan, dan 2 minggu *buffer* ditujukan untuk *bug fixing* serta *planning cycle* berikutnya. Pada setiap *cycle* sendiri terdapat 2 jenis tugas, yaitu *big batch* dan *small batch*. *Big batch* ditujukan untuk proyek yang cukup besar dan kompleks, sehingga membutuhkan waktu yang lama, sekitar 6 minggu. Sedangkan *small batch* merupakan tugas yang relatif lebih sederhana dan dapat diselesaikan dengan cepat, memungkinkan adanya beberapa *small batch* dalam sebuah *cycle*. Dari dua tugas utama yang ada, keduanya merupakan proyek *big batch* yang dilakukan selama kerja magang berlangsung.

Pada *cycle* pertama, dilakukan pengembangan fitur automasi pada integrasi *supplier* A dengan Whitelabel. Automasi yang diterapkan dimulai dari registrasi akun *supplier* A secara langsung di situs web Whitelabel, integrasi sistem, hingga penyediaan beberapa produk *template* favorit untuk *tenant*. Kemudian, pada *cycle* berikutnya, fitur automasi yang dikembangkan berupa sinkronisasi otomatis antara daftar harga di situs web *supplier* B dengan Whitelabel *tenant*. Sinkronisasi ini diterapkan menggunakan *webhook* dan *job*.

3.3 Uraian Pelaksanaan Magang

Kerja magang di PT Indobest Artha Kreasi dilakukan selama ... minggu, dan terdiri dari 3 *cycle*. Rincian kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan magang

Minggu Ke -	**Pekerjaan yang dilakukan**
1	Sesi *onboarding*, *setup* proyek Whitelabel di Windows, dan membuat *formatting* nama untuk produk template
2	*Setup* proyek CMS, menambah komponen yang dibutuhkan pada CMS, mempelajari *workflow* proyek, membuat tampilan awal untuk integrasi dengan registrasi otomatis, dan implementasi API untuk registrasi *tenant* baru *supplier* A
3	Menambah beberapa komponen untuk halaman integrasi, membuat validasi, membuat tampilan untuk *step* 2 dan 3 pada proses integrasi, implementasi API untuk mendapat, menyimpan, dan verifikasi data yang berhubungan dengan pengisian data akun dan aktivasi 2FA
Dilanjutkan pada halaman berikutnya	

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

Minggu Ke -	**Pekerjaan yang dilakukan**
4	Memperbaiki *bug* yang ada, optimasi *flow* integrasi, menambah beberapa fungsi dan implementasi API untuk penambahan produk secara massal
5	Memperbaiki *flow* integrasi dan implementasi API untuk penambahan dan *update* produk secara massal
6	Melakukan *filter* pada produk yang tidak valid, memperbaiki *bugs* yang ada, dan memperbaiki *code review*
7	Menambah tabel baru, optimasi *query* ke *database*, melakukan modularisasi *frontend* yang ada selama proses integrasi, dan mengubah *flow backend*
8	Memperbaiki *code review*, *planning* untuk *cycle* berikutnya, *bug fixing*, serta mempelajari *worflow* proyek berikutnya
9	Melakukan demo fitur registrasi otomatis, melakukan *setup* yang berhubungan untuk proyek berikutnya, membuat *cron* dan *webhook* untuk sinkronisasi otomatis
10	Membuat *cron* untuk *pruning* data, melakukan *improvement* pada *logic update* dan *delete* untuk mengoptimalkan proses sinkronisasi, dan mempelajari *job scheduling*
11	Mengimplementasikan *queue* untuk *job* pada *cron*, memperbaiki hasil *review*, dan *testing* pada *logic* sinkronisasi
12	Memperbaiki hasil *review*, mempersiapkan dokumentasi untuk QA, dan melakukan penyesuaian pada *cron*
13	Melakukan QA pada fitur sinkronisasi, memperbaiki *layout toolbar* pada beberapa halaman admin
14	Memperbaiki tampilan dan responsivitas beberapa halaman pada admin panel
15	*Setup* proyek MP dan *database* yang diperlukan, memperbaiki *code review* pada proyek sinkronisasi, dan membuat *cron* untuk otorisasi otomatis *topup* deposit
16	Mempelajari dan mengimplementasikan *cache lock*, melakukan *update* pada validasi *topup*, mengubah metode pembuatan kode unik, dan memperbaiki hasil *review* otorisasi otomatis
Dilanjutkan pada halaman berikutnya	

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

Minggu Ke -	Pekerjaan yang dilakukan
17	Membedakan dan melakukan *update* pada sistem penanganan deposit Whitelabel dari Mobilepulsa, dan implementasi *auto-increment* untuk id
18	Menambah kolom status dan *action* (otorisasi dan *void*) untuk deposit Whitelabel pada situs web CMS dan memperbaiki hasil *review*
19	Menambahkan menu pengingat deposit, membuat formulir dinamis dan validasi untuk daftar penerima pengingat, membuat beberapa *endpoint*, dan menyiapkan *cron* pengiriman pengingat
20	Melakukan *handling* tersendiri untuk penerima pengingat via Telegram, menambahkan validasi untuk penerima via Telegram, mempelajari dan mengimplementasikan penggunaan bot Telegram untuk pengiriman pengingat

Kerja magang yang terurai pada Tabel 3.1 dilaksanakan menggunakan beberapa perangkat lunak dan perangkat keras sebagai pendukung selama pengembangan berlangsung. Perangkat lunak yang digunakan, antara lain:

1. OS Windows 11
2. Visual Studio Code v1.85.1
3. Git v2.35.1 dan Fork v1.96.1
4. Laravel v5.8.38
5. WAMP v3.3.2, PHP v7.4.33, Apache v2.4.58, dan MySQL v8.2.0
6. Redis v3.0.50
7. MongoDB v5.0.23
8. TablePlus v5.9.6
9. Open VPN Connect v3.3.6
10. Postman v10.24.16

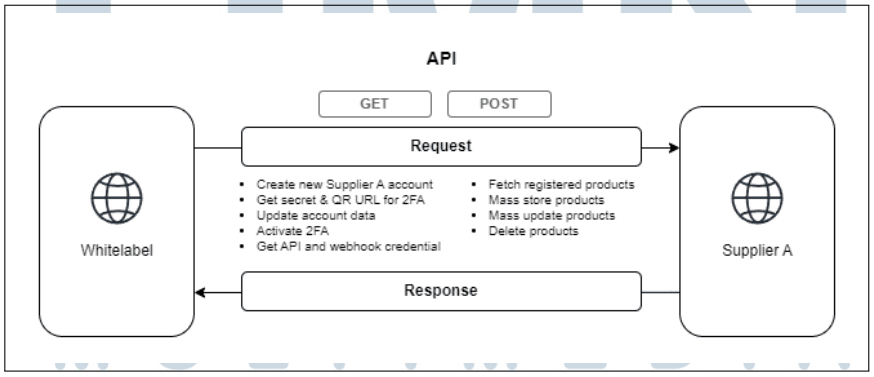
Sedangkan perangkat keras yang digunakan merupakan laptop Asus VivoBook F412DA, dengan spesifikasi sebagai berikut:

1. *Processor* AMD Ryzen 7 3700U
2. RAM 8GB
3. SSD 512 GB NVMe M.2 PCIe

Berikut adalah uraian lengkap dari dua proyek kerja utama yang dilaksanakan selama masa kerja magang, yang keduanya berhubungan dengan automasi untuk situs web Whitelabel. Setiap penjelasan mencakup *requirement*, perancangan sistem, hingga hasil implementasi dari sistem tersebut.

3.3.1 Registrasi Otomatis *Supplier* A dengan Whitelabel

Fitur registrasi otomatis yang dikembangkan hanya dikhususkan untuk *tenant* Whitelabel yang baru bergabung. Hal ini dikarenakan proses automasi juga disertai dengan *setup* produk *template* yang dapat mengganggu produk lainnya jika *tenant* sudah lama bergabung. Automasi yang diaplikasikan pada proses registrasi *supplier* A dengan Whitelabel bertujuan untuk mempersingkat waktu yang dibutuhkan oleh *tenant* ketika melakukan registrasi dan integrasi dengan *supplier* A. Bagian ini akan berisi penjelasan lebih detail mengenai proyek registrasi otomatis *supplier* A dengan situs web Whitelabel milik *tenant*.



Gambar 3.2. Diagram alur *API* antara Whitelabel dengan *supplier* A

Proses registrasi otomatis antara *supplier* A dengan Whitelabel akan membutuhkan banyak komunikasi. Oleh karena itu, digunakan API untuk memperlancar komunikasi yang berlangsung antara keduanya. Alur *request* dan

response yang dilakukan dijabarkan pada Gambar 3.2. Dalam proses ini, situs web Whitelabel akan melakukan beberapa *request* GET dan POST kepada *supplier* A. Kemudian, situs web *supplier* A akan mengembalikan berbagai jenis respons sesuai *request* yang dilakukan oleh Whitelabel. Beberapa komunikasi yang akan terjadi antara keduanya melalui API, antara lain:

1. Registrasi akun *supplier* A
2. Mendapatkan kode *secret* beserta URL untuk QR *code* 2FA
3. Melakukan *update* pada data bisnis dis akun *supplier* A
4. Melakukan aktivasi 2FA
5. Menerima *credential* API and *webhook tenant*
6. Mengambil daftar produk yang terdaftar pada akun *supplier* A
7. Mendaftarkan dan menyimpan produk-produk *template* pada akun *supplier* A secara massal
8. Melakukan *update* massal pada produk-produk di akun *supplier* A
9. Menghapus produk dari daftar produk pada akun *supplier* A

A. *Requirement*

Dalam pengerjaan fitur registrasi otomatis ini, terdapat beberapa *requirement* yang diperlukan untuk dapat mencapai tujuan secara maksimal.

A.1 Opsi Pemilihan Mode Integrasi

Saat *tenant* Whitelabel baru ingin melakukan integrasi dengan *supplier* A, *tenant* akan dihadapkan dengan 2 opsi mode integrasi, yaitu otomatis dan manual. Mode manual akan mengarahkan *tenant* untuk melakukan integrasi seperti yang sudah ada, di mana mereka akan diminta untuk membuka situs web *supplier* A dan melakukan registrasi secara mandiri, hingga integrasi berhasil. Ketika *tenant* memilih untuk melakukan integrasi secara manual, maka tahapan registrasi dan integrasi otomatis selanjutnya tidak akan dilanjutkan. Metode ini disediakan untuk *tenant* yang sebelumnya sudah memiliki akun pada *supplier* A. Namun diluar kasus

tersebut, secara umum, *tenant* akan disarankan untuk memilih mode otomatis yang juga menyediakan registrasi otomatis untuk dapat meningkatkan efisiensi selama proses integrasi.

A.2 Pendaftaran Akun *Supplier* A

Ketika *tenant* memilih mode otomatis, maka *tenant* akan diarahkan untuk mengisi *password* admin Whitelabel. Jika berhasil, maka secara otomatis, akun *supplier* A dengan data *tenant* akan dibuat tanpa perlu membuka situs web *supplier* A tersebut. Namun jika sebelumnya sudah ada akun *supplier* A dengan data *tenant*, maka *tenant* akan diarahkan untuk melakukan integrasi secara manual.

A.3 Formulir Data Akun

Kemudian untuk melanjutkan proses registrasi otomatis, *tenant* perlu mengisi beberapa data usaha untuk melengkapi data pada akun *supplier* A yang sudah didaftarkan. Terdapat beberapa data yang perlu diisi oleh *tenant*, dan ada juga beberapa data yang dapat diambil dari data yang terdaftar pada akun admin Whitelabel. Perbedaannya dengan proses registrasi yang ada saat ini adalah pemindahan proses langsung pada situs web Whitelabel.

A.4 Aktivasi Two-Factor Authentication (2FA)

Proses registrasi kemudian dilanjutkan dengan aktivasi *two-factor authentication* (2FA). *Tenant* akan diberikan *QR code* yang berlaku selama beberapa saat agar dapat dipindai menggunakan aplikasi 2FA untuk menerima kode unik. Sebagai alternatif, *tenant* juga dapat memasukkan kode ”secret” ke dalam aplikasi 2FA untuk menerima kode unik jikalau *QR code* sudah tidak valid untuk dipindai. Jika kode unik yang kemudian dimasukkan pada situs web Whitelabel valid, maka 2FA pada akun *supplier* A *tenant* akan diaktifkan.

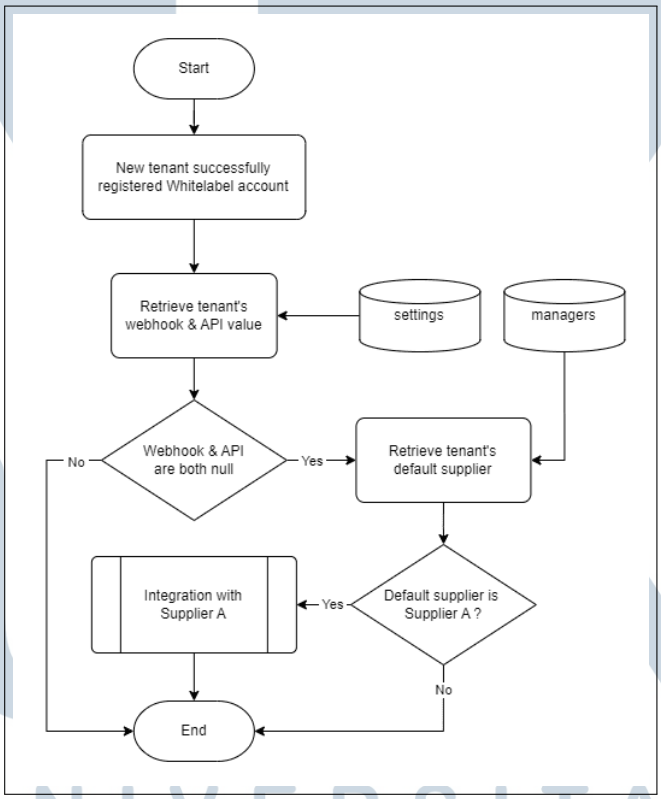
A.5 Auto Setup Onboarding Pricelist

Pada tahapan akhir, akun Whitelabel *tenant* akan diintegrasikan dengan akun *supplier* A yang sudah dibuat. Di tahapan ini, proses registrasi otomatis sudah selesai, begitu juga integrasi. Kemudian sebagai tambahan, akun Whitelabel *tenant*

akan dilengkapi dengan beberapa produk favorit sebagai *template*, agar *tenant* dapat mempersingkat waktu dalam memilih dan menambah produk yang akan dijual.

B. Perancangan Sistem

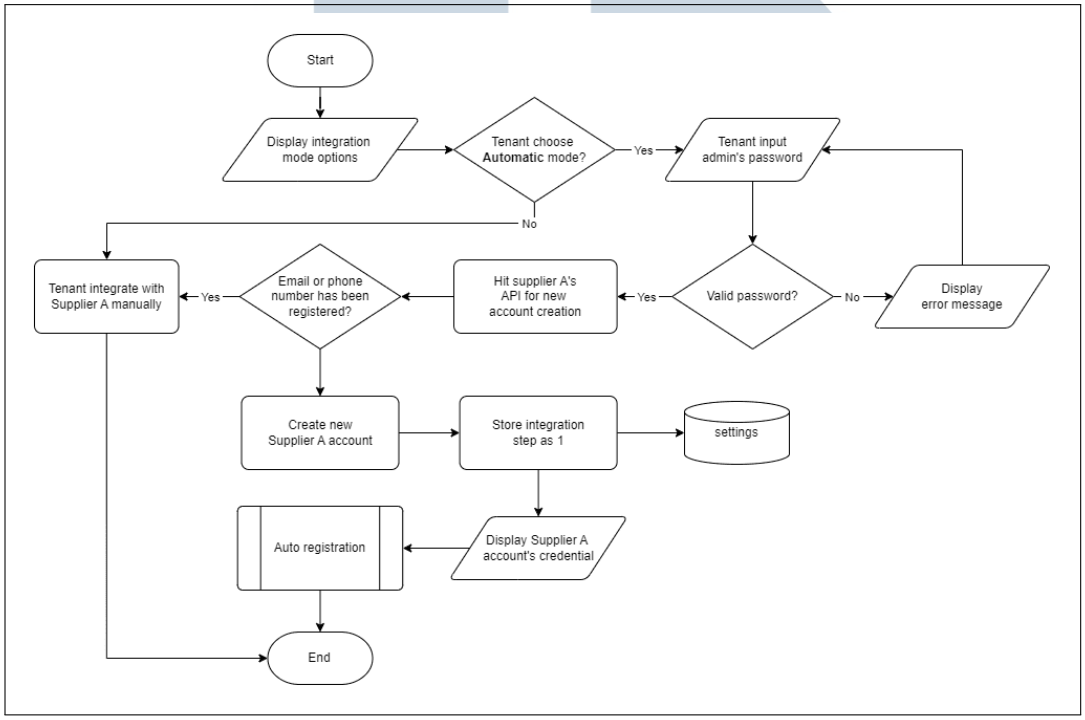
Fitur registrasi otomatis merupakan bagian dari proses integrasi. Sedangkan fitur integrasi supplier A dengan situs web Whitelabel milik *tenant* yang menyertakan registrasi secara otomatis sudah dimulai sejak awal dengan adanya pengecekan pada data *tenant* terlebih dahulu. Hal ini dilakukan untuk melakukan proses integrasi dengan *supplier* A. Proses ini dijabarkan dalam *flowchart* pada Gambar 3.3



Gambar 3.3. *Flowchart* utama proses integrasi supplier A

Proses diawali dengan pendaftaran akun Whitelabel *tenant* baru, kemudian diikuti dengan pemeriksaan beberapa variabel untuk memastikan apakah integrasi dengan *supplier* A akan dapat dilakukan. Hal yang dicek adalah nilai *webhook* dan API, jika salah satunya sudah memiliki nilai, maka proses integrasi tidak akan dilanjutkan karena hal tersebut menandakan bahwa akun Whitelabel *tenant* sudah terintegrasi dengan *supplier* A. Namun jika nilai *webhook* dan API tidak ada dalam

tabel ”settings” *tenant*, maka sistem akan melakukan pengecekan pada nilai *default supplier* yang disimpan pada tabel ”managers”. Apabila data *default supplier* yang ada dalam tabel bernilai ”*supplier* A”, maka proses integrasi dengan *supplier* A akan dilakukan.

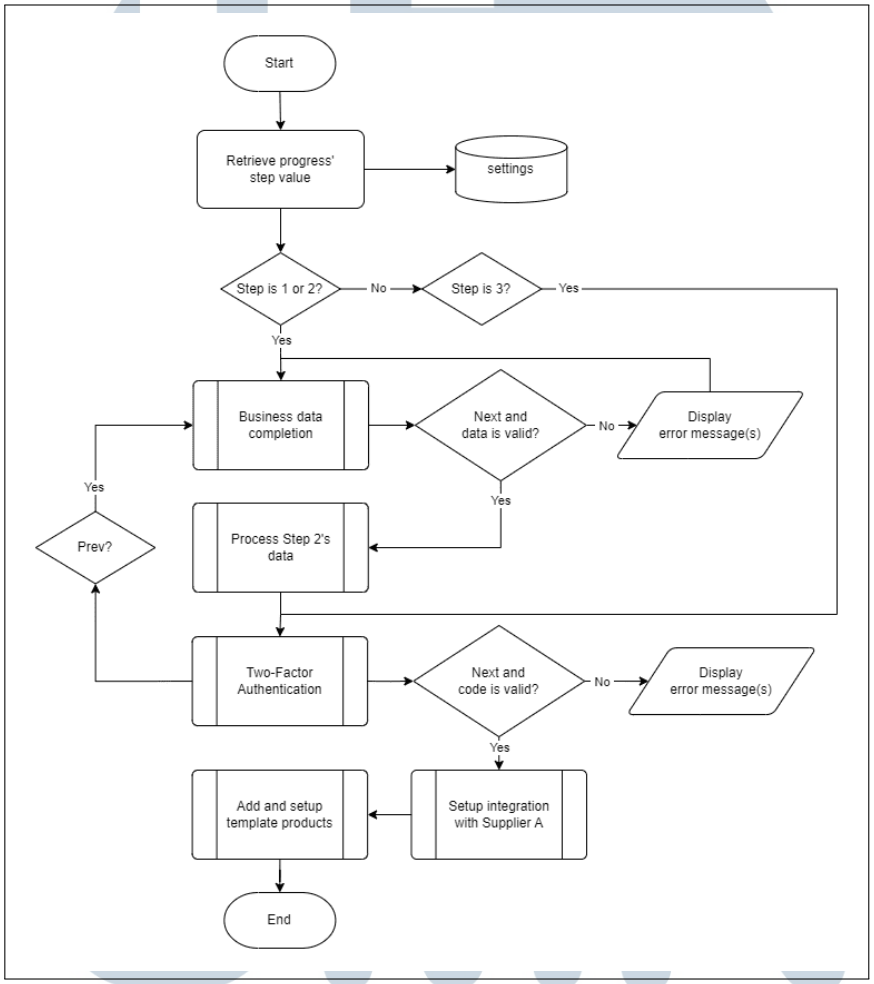


Gambar 3.4. *Flowchart* modul *Integration with supplier A*

Pada *flowchart* yang ada pada Gambar 3.4, digambarkan proses awal bagi *tenant* untuk melakukan integrasi dengan *supplier* A. *Tenant* akan dihadapkan dengan tampilan antarmuka yang memintanya untuk memilih mode integrasi, antara otomatis dan manual. Proses registrasi otomatis hanya akan berjalan ketika *tenant* memilih mode otomatis. Kemudian, ketika *tenant* memilih mode otomatis, sistem akan meminta *tenant* untuk mengisi *password* admin Whitelabel miliknya. Jika *password* yang dimasukkan sudah sesuai, sistem akan memanggil API dari *supplier* A untuk melakukan registrasi akun sekaligus melakukan beberapa pengecekan.

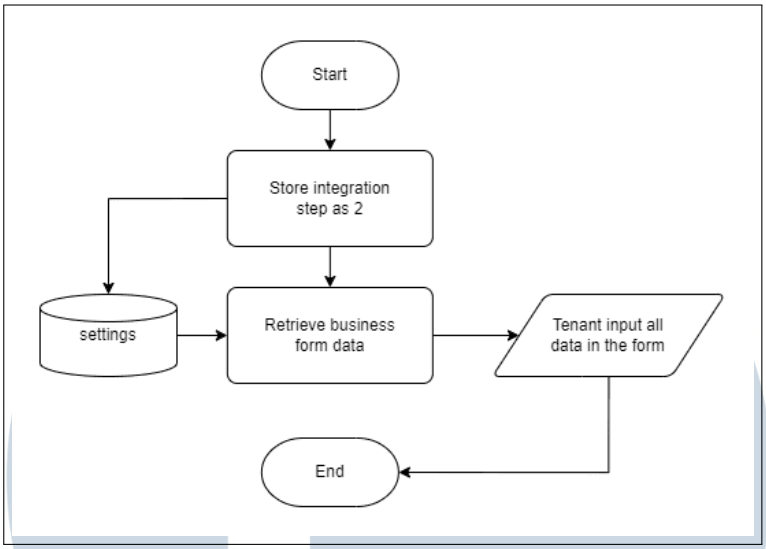
Pada pembuatan akun melalui API, jika ternyata sudah ada akun dengan *email* atau nomor telepon yang sama dengan milik *tenant*, maka *tenant* akan diarahkan untuk melakukan integrasi secara manual. Namun, jika akun *supplier* A berhasil terbuat, sistem akan menyimpan data *step* dengan nilai 1 ke dalam tabel ”settings” untuk memantau tahapan yang sedang dilalui *tenant* pada proses registrasi otomatis. Hal ini dilakukan untuk menghindari pengulangan proses

jikalau *tenant* melakukan *refresh* pada halaman tersebut, ataupun meninggalkannya untuk waktu yang cukup lama. Kemudian, tampilan akan berubah menjadi tampilan *credential* untuk akun *supplier* A milik *tenant*. *Credential* yang ada berupa *email*, nomor telepon, dan *password*. Kemudian sistem akan menjalankan modul registrasi otomatis dengan *supplier* A.



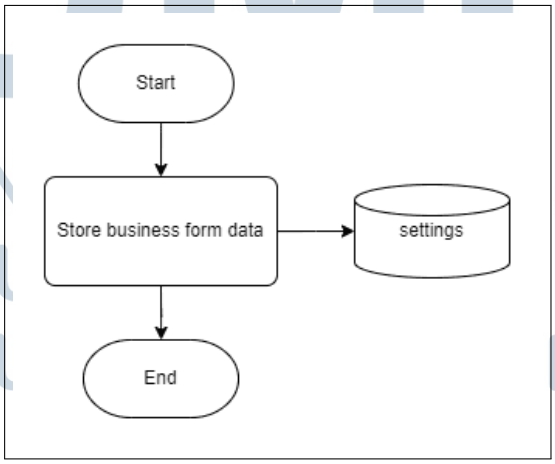
Gambar 3.5. *Flowchart* modul *Auto registration*

Proses registrasi otomatis pada *supplier* A digambarkan pada Gambar 3.5. Modul ini mencakup rangkaian proses registrasi secara otomatis sebagai lanjutan dari *flowchart* sebelumnya, yang dapat terpanggil juga ketika *tenant* melakukan *refresh* halaman. Diawali dengan pengambilan data *progress* dari tabel ”settings”, sistem akan melakukan pengecekan terhadap nilai *step* yang ada. Jika *step* bernilai 1 atau 2, maka *tenant* akan diarahkan ke dalam proses pendataan data usaha. Proses ini dijabarkan dalam *flowchart* pada Gambar 3.6.



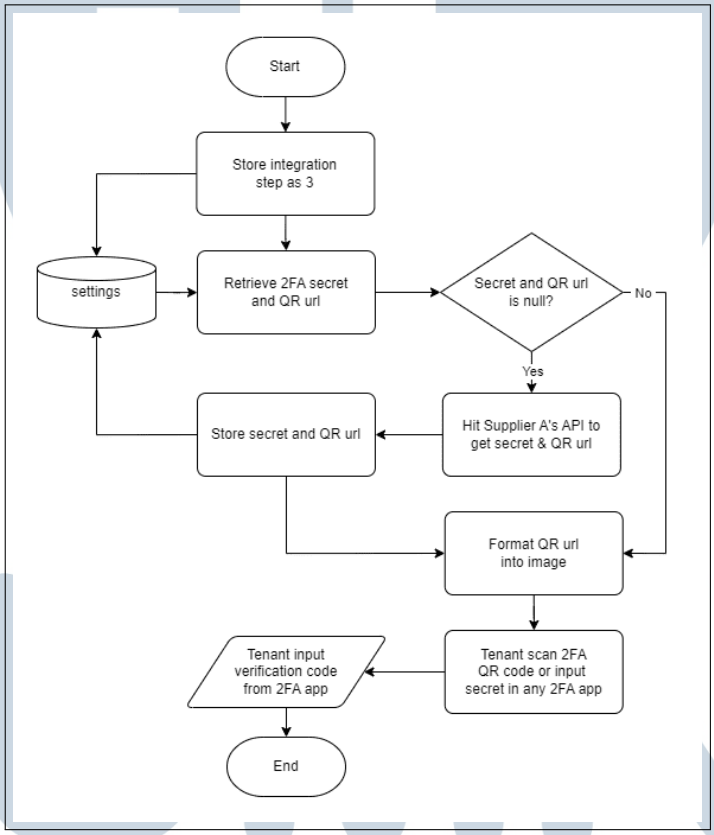
Gambar 3.6. *Flowchart* modul *Business data completion*

Gambar 3.6 menggambarkan proses yang dimulai dari penyimpanan data *step* dengan nilai 2 pada tabel ”settings”. Setelah itu, sistem juga akan mengambil data usaha yang pernah tersimpan dalam tabel tersebut jika ternyata *tenant* sudah pernah melakukan pengisian formulir tersebut. Data ini disimpan dalam tabel untuk memungkinkan *tenant* kembali ke halaman ini di waktu berikutnya. Dengan demikian, *tenant* tidak perlu mengisi ulang formulir yang sama, karena data yang telah dimasukkan sebelumnya akan tersimpan dalam tabel ”settings” dan diambil kembali untuk ditampilkan. Kemudian *tenant* dapat mengisi, melengkapi, ataupun mengubah data usaha mereka. Modul ini selesai ketika *tenant* memilih untuk lanjut ke tahap selanjutnya.



Gambar 3.7. *Flowchart* modul *Process step 2’s data*

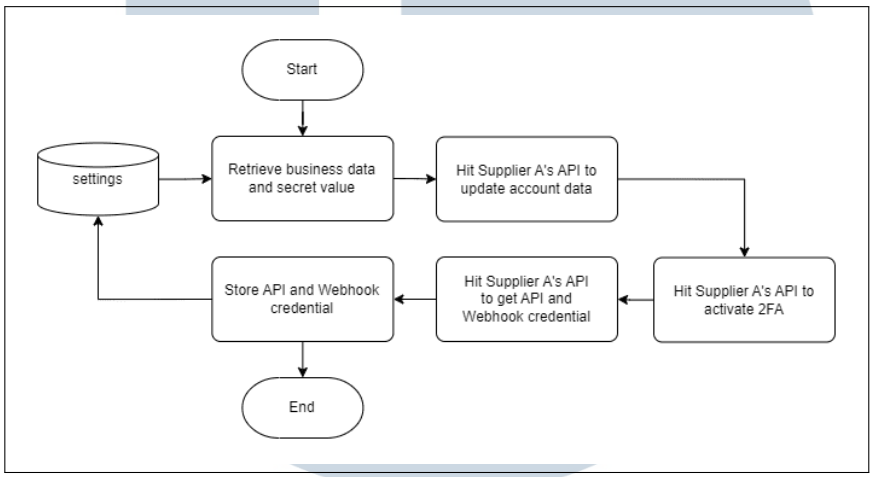
Tahapan selanjutnya, sesuai dengan yang ada pada Gambar 3.5, yaitu validasi data yang diisi. Jika data yang diisi belum sesuai, maka sistem akan menampilkan pesan *error* kepada *tenant* dan mengarahkannya untuk melakukan pengecekan pada data yang dimasukkan untuk diperbaiki. Di sisi lain, jika data yang dimasukkan sudah valid, sistem akan memproses data yang telah diisi. Gambar 3.7 diatas menggambarkan modul yang memproses data usaha yang telah diisi. Proses ini selesai dengan menyimpan data tersebut ke dalam tabel ”settings” agar nantinya dapat dipanggil kembali untuk ditampilkan jika dibutuhkan.



Gambar 3.8. *Flowchart* modul *Two factor authentication*

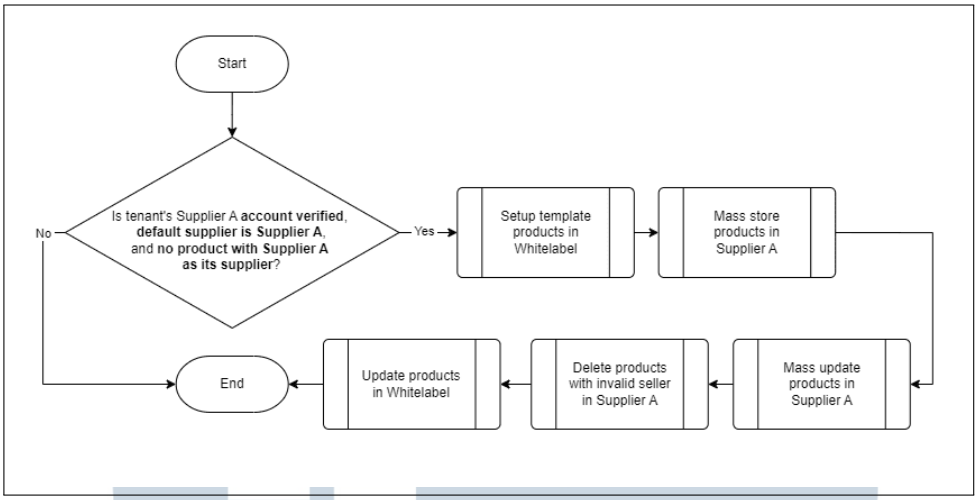
Proses dilanjuti dengan aktivasi *two-factor authentication* (2FA). Pada proses ini, *tenant* dapat bebas untuk kembali ke tahapan pengisian data usaha ataupun menyelesaikan pengaktifan 2FA. Modul aktivasi 2FA sendiri dijabarkan dalam *flowchart* yang ada pada Gambar 3.8. Sama seperti modul pengisian data usaha, modul ini juga diawali dengan penyimpanan data *step* dengan nilai 3 ke dalam tabel ”settings”. Setelah itu, sistem juga akan mengambil beberapa data yang berkaitan dengan 2FA dari tabel yang sama. Jika data *secret* dan URL QR *code* belum memiliki nilai, maka Whitelabel akan memanggil API *supplier* A

untuk mengambil data yang dibutuhkan dan menyimpannya dalam tabel. Setelah data QR *code* dan *secret* sudah bernilai, maka sistem akan menampilkan QR *code* yang sudah diformat menjadi gambar yang dapat dipindai oleh *tenant* menggunakan aplikasi 2FA. Namun jika ternyata QR *code* sudah tidak berlaku, *tenant* tetap dapat mengaktifkan 2FA dengan memasukkan kode *secret* yang disediakan. Nantinya dari aplikasi 2FA tersebut, *tenant* akan mendapatkan kode unik yang dapat dimasukkan ke dalam tempat yang disediakan.



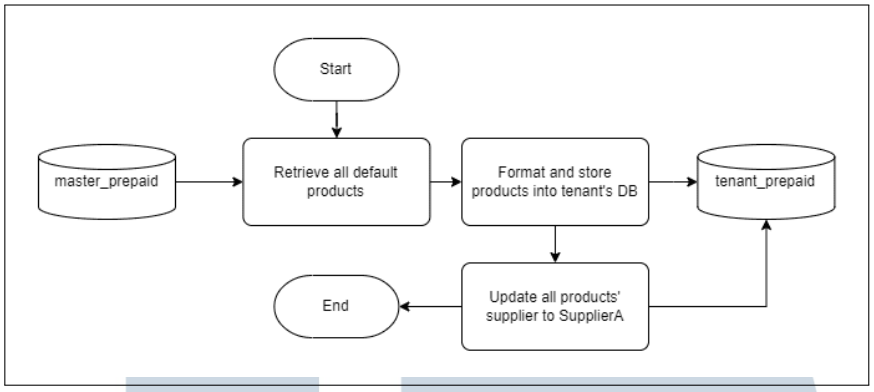
Gambar 3.9. *Flowchart* modul *Setup integration with supplier A*

Lalu, sesuai dengan yang ada pada Gambar 3.5, proses tersebut dilanjutkan dengan melakukan validasi pada kode 2FA yang dimasukkan *tenant*. Jika validasi gagal, Whitelabel akan menampilkan pesan *error* dan meminta *tenant* untuk mengisi kembali. Namun jika validasi berhasil, sistem akan menjalankan modul *setup* integrasi seperti yang ada pada Gambar 3.9. Saat ini, proses registrasi otomatis sudah selesai, dan kemudian dilanjuti oleh proses integrasi. Pertama-tama, akan dilakukan pengambilan data usaha yang diisi pada tahap 2 dan nilai *secret* dari tabel ”settings”. Kemudian sisi Whitelabel akan memanggil API *supplier* A untuk melakukan *update* data usaha pada akun *tenant*. Setelah selesai, sistem juga akan memanggil API yang berfungsi untuk melakukan aktivasi 2FA pada akun *supplier* A *tenant* menggunakan data *secret* yang ada. Proses ini dilanjuti dengan pemanggilan API untuk melakukan integrasi dengan *supplier* A dan mendapatkan *credential* API dan *webhook* untuk akun Whitelabel *tenant*. Nantinya kedua nilai ini akan disimpan dalam tabel ”settings” milik *tenant* agar dapat dipanggil dan ditampilkan pada halaman integrasi.



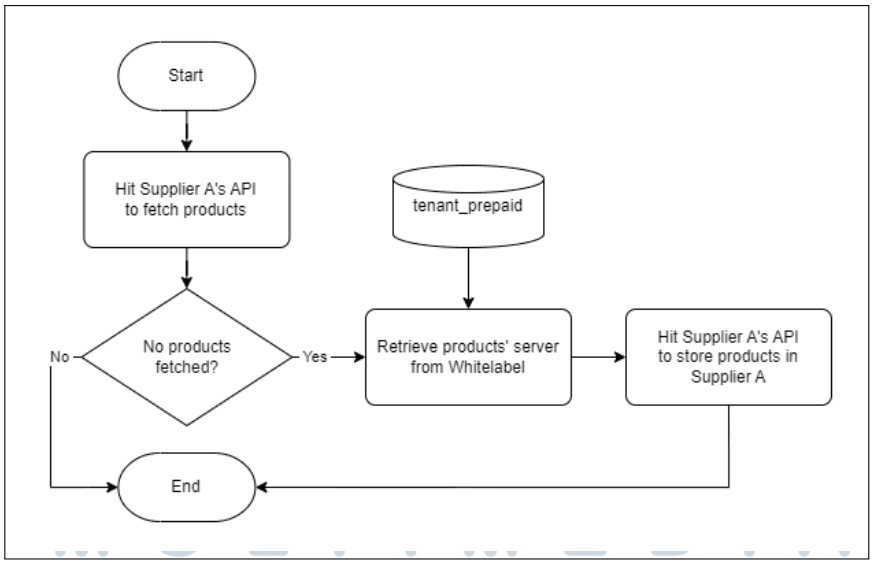
Gambar 3.10. *Flowchart* modul *Add and setup template products*

Setelah proses registrasi dan integrasi selesai dilakukan, sistem akan secara otomatis melakukan *setup* produk-produk favorit sebagai produk *template* untuk *tenant* baru tersebut. Modul ini digambarkan pada Gambar 3.10. Namun sebelum *setup* ini dilakukan, ada beberapa hal yang perlu dicek, antara lain status akun *supplier* A, nilai *default supplier*, dan produk yang saat ini ada. Saat pertama kali *tenant* baru membuat akun Whitelabel, terdapat beberapa produk yang sudah disediakan sebagai produk *template*, namun bukan dengan pilihan *supplier* A, melainkan Mobilepulsa. Sehingga pada proses inilah produk *template* yang ada akan disesuaikan kembali. Syarat yang diperlukan agar *setup* produk *template* dapat berjalan adalah akun *supplier* A *tenant* yang sudah ter-verifikasi, terdata *supplier* A sebagai nilai *default supplier*, dan tidak ada produk dengan *supplier* dari *supplier* A yang terdaftar dalam akun *tenant*. Jika ketiga syarat ini terpenuhi, maka sistem akan melakukan beberapa rangkaian proses *setup*. Prosesnya mencakup penambahan produk di akun Whitelabel *tenant* dan di akun *supplier* A *tenant*, pembaruan produk di akun *supplier* A, penghapusan produk yang tidak valid dari akun *supplier* A, dan terakhir, pembaruan produk di akun Whitelabel *tenant*.



Gambar 3.11. *Flowchart* modul *Setup template products in Whitelabel*

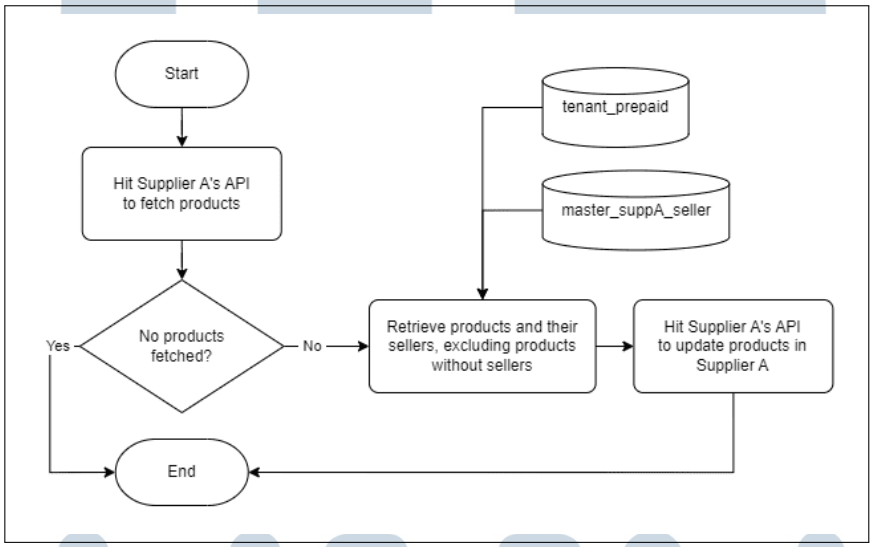
Gambar 3.11 menampilkan proses *setup* produk dengan *supplier* A pada akun Whitelabel *tenant*. Proses diawali dengan pengambilan produk-produk dengan *supplier* A yang sudah ditandai sebagai produk *default* dari tabel utama ”master_prepaid”. Kemudian dilakukan format pada produk-produk ini, seperti nama, *supplier*, dan beberapa atribut lainnya. Produk yang sudah diformat kemudian disimpan ke dalam tabel produk milik *tenant*. Setelah semua produk berhasil dimasukkan ke dalam tabel ”tenant_prepaid”, barulah semua produk yang ada diubah *supplier*-nya menjadi *supplier* A.



Gambar 3.12. *Flowchart* modul *Mass store products in supplier A*

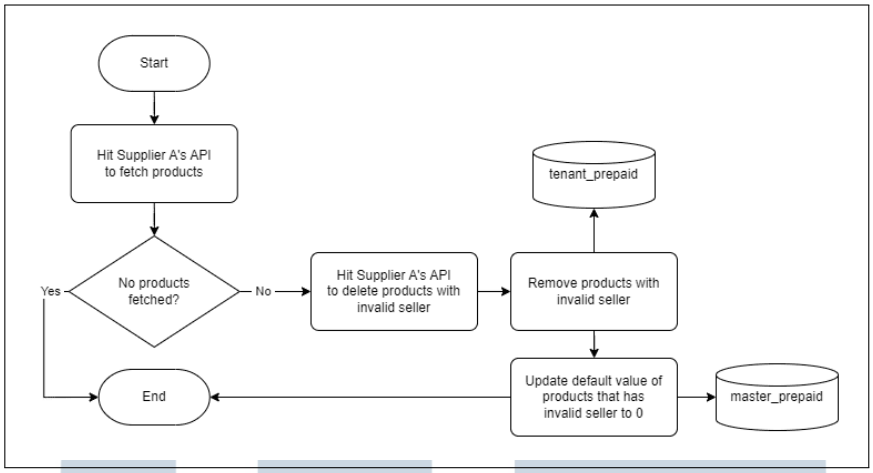
Proses setup dilanjutkan dengan penambahan produk-produk yang sudah terdaftar dalam akun Whitelabel *tenant*, ke dalam akun *supplier* A milik *tenant*. Proses ini dijabarkan dalam *flowchart* pada Gambar 3.12. Namun, sebelum

melakukan penambahan pada akun *supplier* A *tenant*, sistem akan memanggil API *supplier* A terlebih dahulu untuk mendapatkan produk yang saat ini terdaftar dalam akun tersebut. Agar produk dapat ditambahkan, perlu dipastikan bahwa tidak ada produk yang terdaftar dalam akun *supplier* A *tenant* saat itu. Setelah dipastikan bahwa penambahan dapat dilakukan, maka akan diambil nilai *server* dari setiap produk yang terdaftar dalam akun Whitelabel milik *tenant* melalui tabel ”tenant_prepaid”. Nilai-nilai *server* inilah yang akan dilanjutkan ke pihak *supplier* A melalui API untuk didaftarkan produknya pada akun *supplier* A milik *tenant* secara massal.



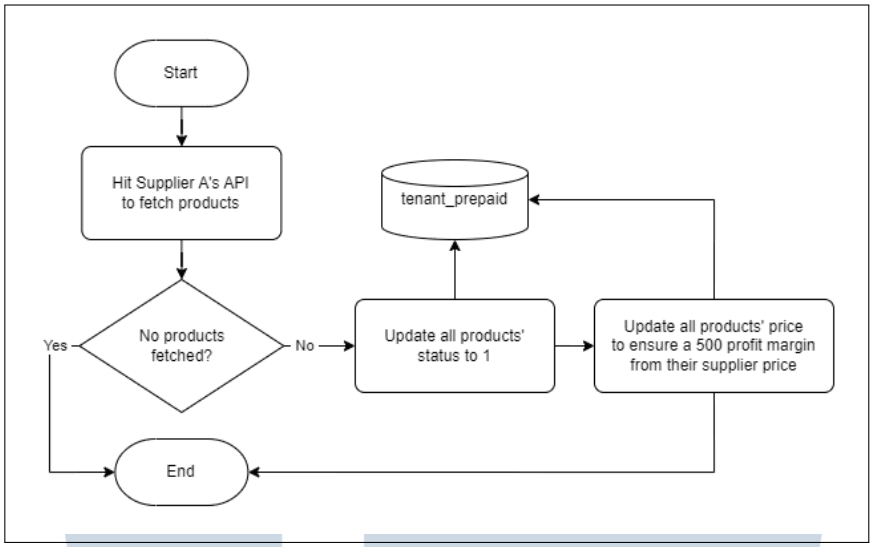
Gambar 3.13. *Flowchart* modul *Mass update products in supplier A*

Penambahan pada akun *supplier* A secara massal dilanjutkan dengan proses *update* secara massal yang ada pada Gambar 3.13. Berbeda dari proses penambahan massal sebelumnya, *update* massal mengharuskan adanya produk yang terdaftar dalam akun *supplier* A milik *tenant*. Jika ternyata belum ada produk pada akun *supplier* A, maka proses ini akan langsung berakhir. Setelah diketahui ada produk yang terdaftar melalui API, sistem akan mengambil semua data produk beserta data *seller* setiap produknya dari tabel ”tenant_prepaid” dan ”master_suppA_seller”. Data-data ini kemudian diolah dan akan dikirim melalui API yang disediakan *supplier* A untuk melakukan *update* produk secara massal pada akun *supplier* A *tenant*. *Update* yang dimaksud adalah perubahan data setiap produk yang sebelumnya masih kosong.



Gambar 3.14. *Flowchart* modul *Delete products with invalid seller in supplier A*

Flowchart pada Gambar 3.14 melanjutkan proses setup yang sedang berjalan. Kali ini, akan dilakukan penyaringan terhadap produk-produk dengan *seller* yang sudah tidak valid. Berubahnya status *seller* menjadi tidak valid dapat disebabkan karena beberapa hal, contohnya *seller* sudah tidak aktif ataupun *seller* tidak lagi terdaftar pada *marketplace supplier* A. Hal pertama yang dilakukan sama seperti yang ada pada modul sebelumnya, yaitu pengecekan mengenai ada atau tidaknya produk dalam akun *supplier* A *tenant*. Jika tidak ada produk yang terdaftar, maka proses diberhentikan. Tetapi, apabila ada produk yang terdaftar, maka produk-produk tersebut akan dicek kembali untuk menemukan produk-produk dengan *seller* yang tidak valid. Nantinya produk-produk ini akan dihapus dari akun *supplier* A menggunakan API yang disediakan. Penghapusan tidak hanya terjadi pada akun *supplier* A, namun juga pada akun Whitelabel *tenant*. Setelah semua produk yang tidak sesuai dihapus, maka produk-produk tersebut juga tidak akan ditandai sebagai produk *template* kedepannya untuk meminimalkan adanya kasus serupa. Hal ini dilakukan dengan melakukan *update* pada kolom penanda yang menandakan bahwa produk tersebut merupakan produk *template* atau bukan, pada tabel utama ”master_prabayar”.



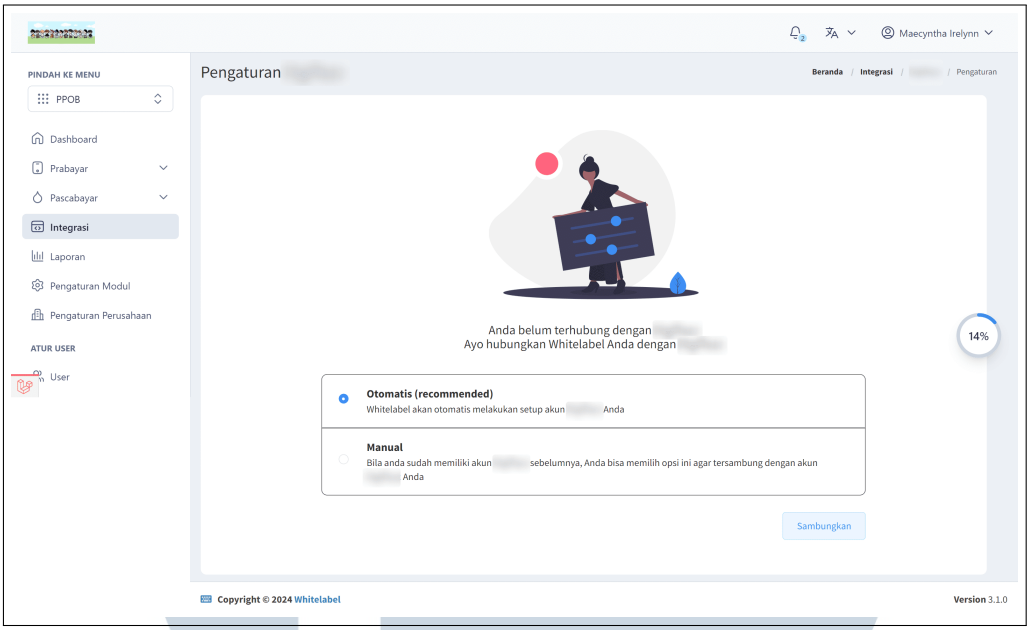
Gambar 3.15. *Flowchart* modul *Update products in Whitelabel*

Tahapan terakhir pada proses setup, sekaligus pada keseluruhan proses integrasi Whitelabel dengan *supplier* A, dijelaskan pada Gambar 3.15. Proses dimulai dengan mengambil produk yang terdaftar dalam akun *supplier* A, lalu memeriksa keberadaannya. Apabila ada produk terdaftar, maka akan dilakukan *update* pada produk-produk tersebut, tapi pada akun Whitelabel *tenant*. Data yang di *update* berupa status produk, agar produk dapat dijual pada situs *tenant*. Tidak hanya perubahan status, tapi dilakukan juga *update* harga awal untuk memastikan bahwa *tenant* mendapat keuntungan Rp. 500 dari setiap produk yang dijual. *Update* ini dilakukan terhadap produk-produk yang ada di dalam tabel ”tenant_prepaid” untuk memastikan bahwa produk sudah siap untuk ditawarkan untuk dijual.

Semua tabel yang digunakan pada proyek automasi registrasi yang ada pada proses integrasi antara akun Whitelabel dengan *supplier* A sudah ada dan digunakan dari sebelum fitur automasi ini dibuat. Tabel-tabel yang dimaksud adalah tabel ”settings”, ”tenant_prepaid”, ”master_prepaid”, dan ”master_suppA_seller”.

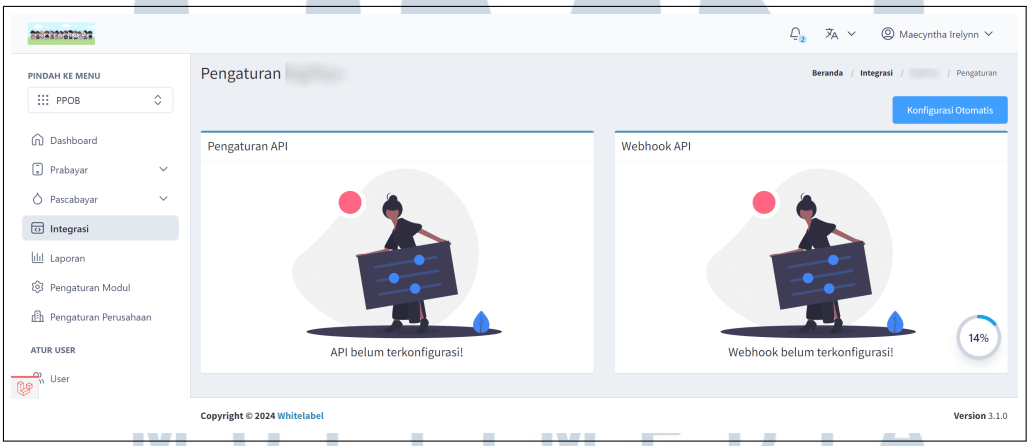
C. Implementasi Sistem

Dari sisi tampilan yang ditampilkan kepada *tenant*, proses registrasi otomatis antara akun Whitelabel dengan akun *supplier* A *tenant* dibagi menjadi 3 tahap. Diluar 3 tampilan utama untuk registrasi otomatis, di awal proses integrasi, sebelum menentukan apakan registrasi otomatis akan dilakukan, *tenant* akan dihadapkan dengan tampilan sesuai pada Gambar 3.16.



Gambar 3.16. Implementasi tampilan awal integrasi

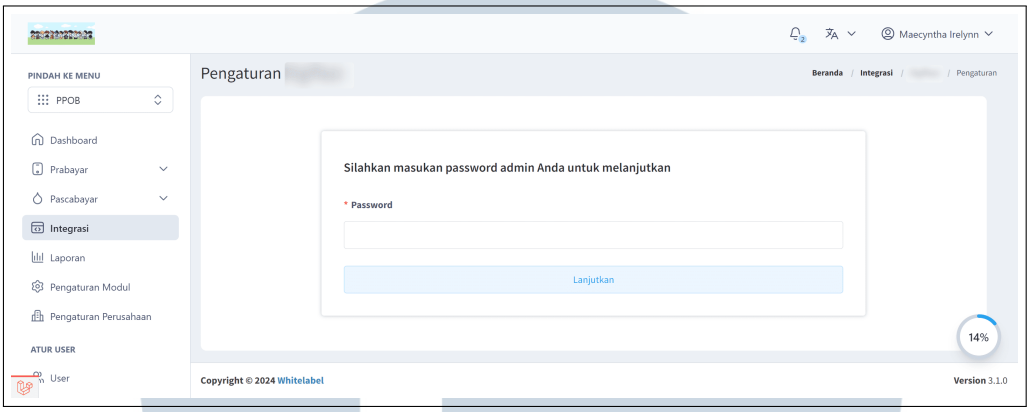
Tampilan pada Gambar 3.16 memungkinkan *tenant* untuk dapat memilih mode integrasi mana yang akan digunakan, antara otomatis dan manual. Pilihan disediakan dengan tampilan radio *button* yang dapat dilanjutkan dengan tombol ”Sambungkan” di kanan bawah. Hanya jika *tenant* memilih mode otomatis, baru proses registrasi otomatis dapat dilakukan.



Gambar 3.17. Implementasi tampilan integrasi manual

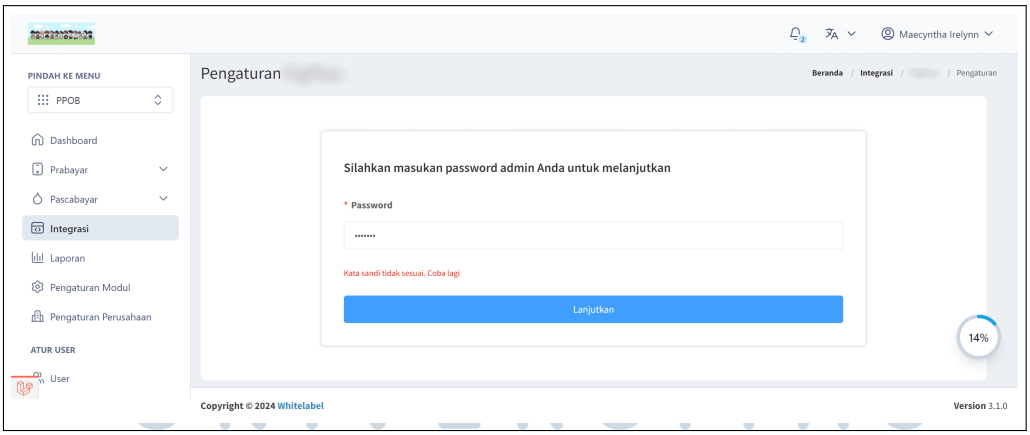
Apabila *tenant* memilih mode manual, maka tampilan yang tampil akan seperti yang ada pada Gambar 3.17. Tampilan tersebut berisi beberapa tombol yang dapat membantu *tenant* untuk melakukan integrasi dengan akun *supplier* A

miliknya. Selain itu, *tenant* akan perlu untuk melakukan registrasi akun *supplier* A secara manual.



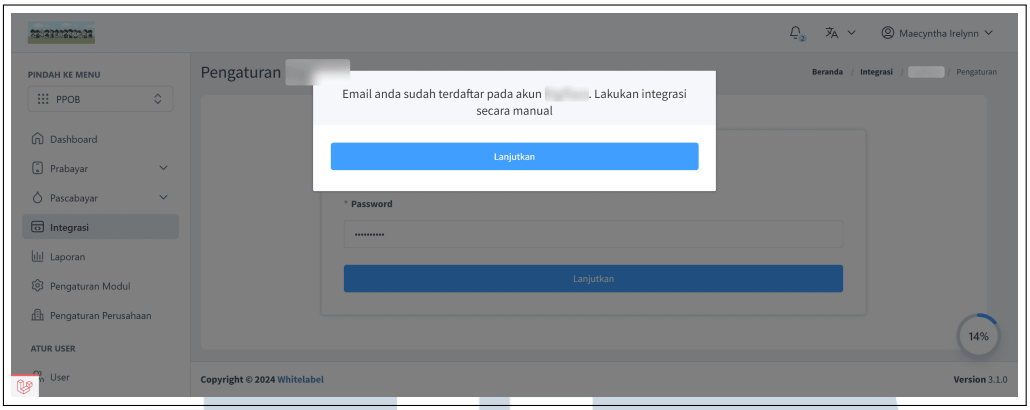
Gambar 3.18. Implementasi tampilan pengisian *password*

Gambar 3.18 merupakan tampilan yang ditampilkan ketika *tenant* memilih opsi otomatis. Pada tahap ini, *tenant* sudah memasuki tahap registrasi otomatis, dimana *tenant* hanya perlu mengisi *password* admin Whitelabel pada tempat yang tersedia. Jika *tenant* melakukan *refresh* pada halaman, maka tampilan akan kembali ke tampilan semula untuk memulai ulang integrasi karena belum ada progres yang dihasilkan.



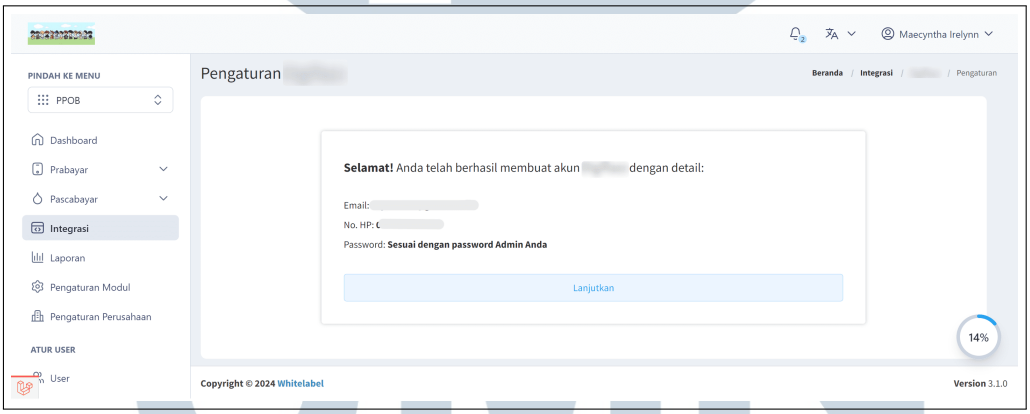
Gambar 3.19. Implementasi validasi password

Validasi yang diterapkan hanya berupa kewajiban untuk mengisi *password* dan pengecekan dengan *password* admin Whitelabel sebenarnya. Tampilan pada Gambar 3.19 merupakan contoh tampilan yang sama, namun dengan pesan validasi bahwa *password* yang dimasukkan tidak sesuai.



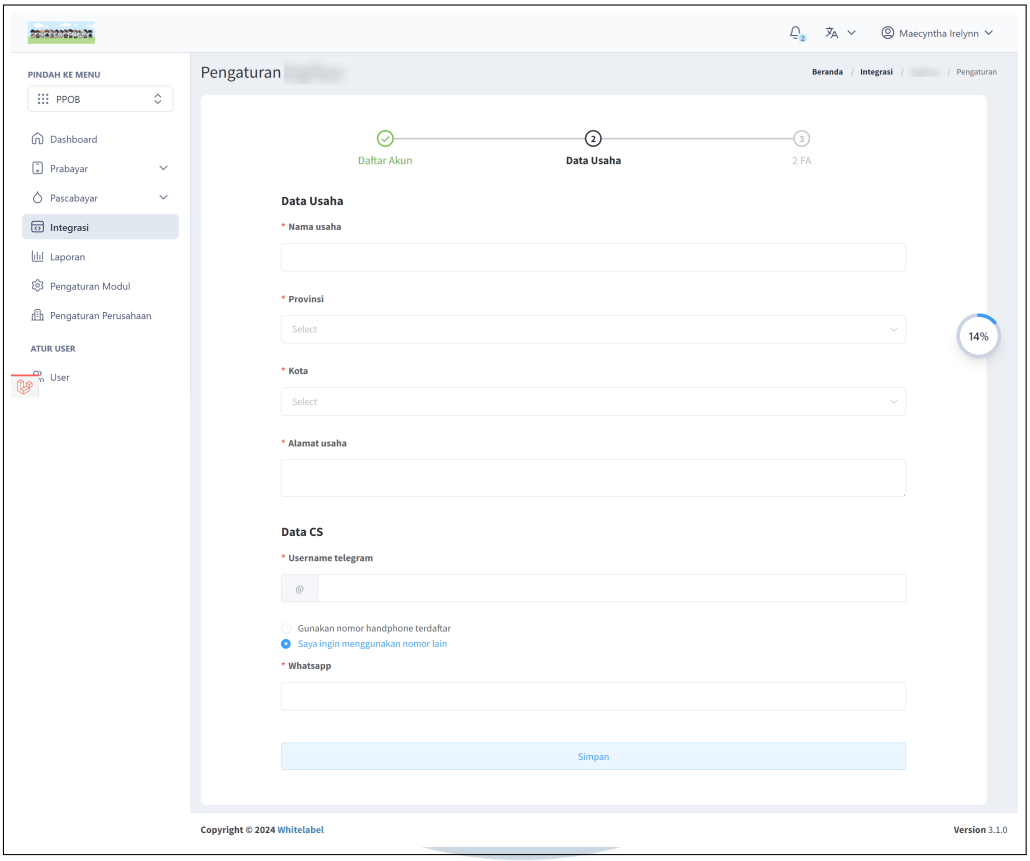
Gambar 3.20. Implementasi modal arahan integrasi

Gambar 3.20 merupakan tampilan modal yang akan muncul jika ternyata sudah ada akun *supplier* A yang sebelumnya sudah pernah terdaftar dengan *email* atau nomor telepon yang sama dengan milik *tenant*. Tombol ”Lanjutkan” mengarahkan *tenant* ke tampilan pada Gambar 3.17 untuk melakukan registrasi secara manual.



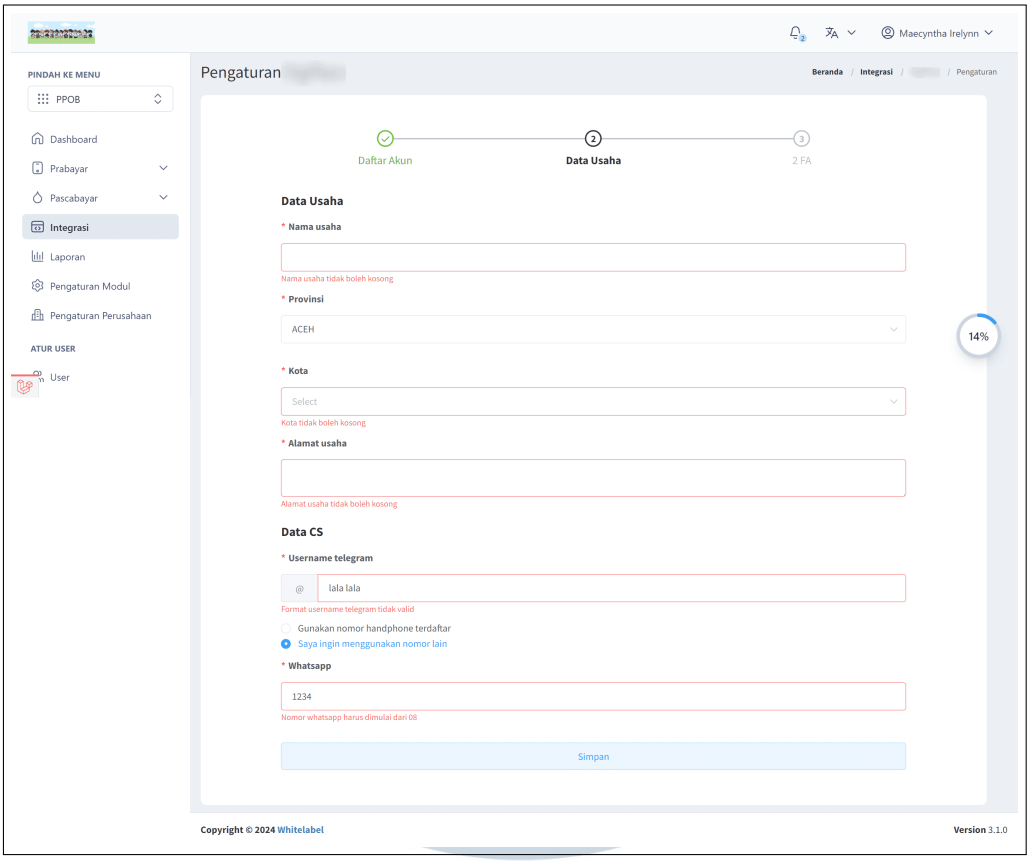
Gambar 3.21. Implementasi tampilan *credential* akun *supplier* A *tenant*

Tampilan pada Gambar 3.21 akan ditampilkan jika akun *supplier* A berhasil didaftarkan. Tampilan ini menampilkan data-data yang akan diperlukan *tenant* jika mereka ingin melakukan *login* mandiri ke dalam situs web *supplier* A.



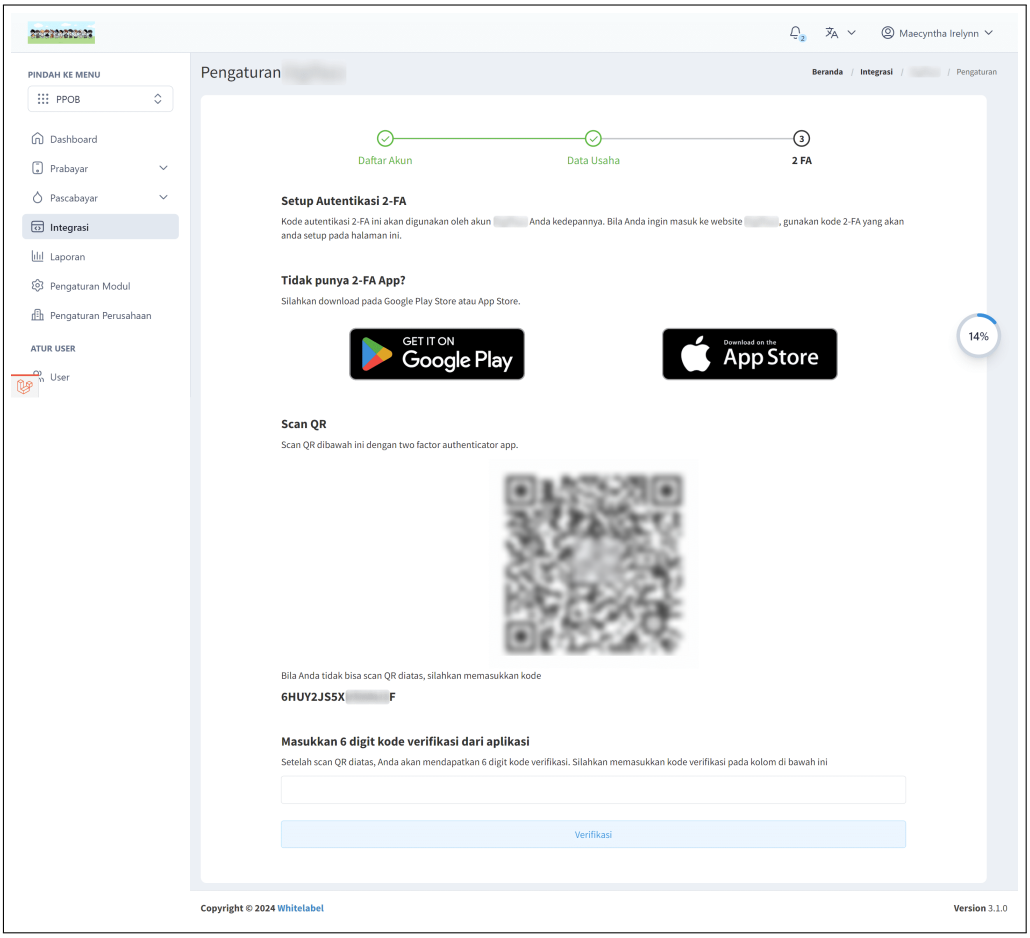
Gambar 3.22. Implementasi tampilan *Step* 2

Gambar 3.22 merupakan implementasi tampilan untuk pengisian data usaha *tenant*. Terdapat beberapa kolom yang wajib diisi oleh *tenant* untuk melengkapi data pada akun *supplier* A milik *tenant*.



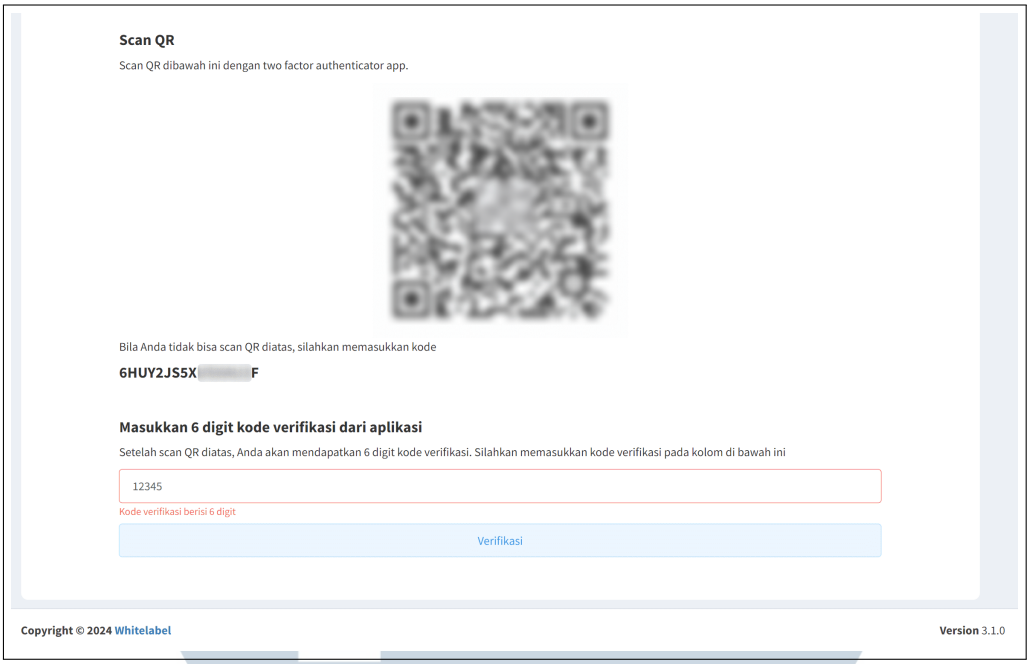
Gambar 3.23. Implementasi validasi *Step* 2

Pada Gambar 3.23 ditampilkan formulir dengan validasi *error*. Hampir setiap kolom wajib diisi namun memiliki validasi isi yang berbeda-beda. Contohnya, kolom ”Whatsapp” tidak harus diisi ketika *tenant* memilih untuk menggunakan nomor *whatsapp* yang sama dengan nomornya. Validasi lainnya seperti *username* Telegram yang tidak boleh memiliki spasi, nomor Whatsapp yang harus diawali dengan angka 08 dan hanya boleh memiliki 10-13 angka, dan lainnya.



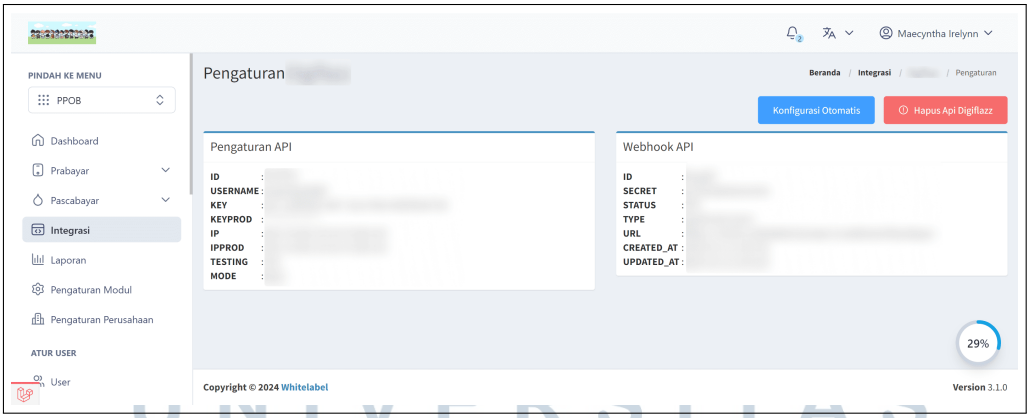
Gambar 3.24. Implementasi tampilan *Step* 3

Implementasi tampilan aktivasi 2FA sesuai dengan ada yang pada Gambar 3.24. Tampilan ini berisi informasi-informasi yang diperlukan *tenant* untuk melakukan aktivasi 2FA pada akun *supplier* A miliknya. Disediakan *link* yang mengarahkan *tenant* untuk mengunduh aplikasi 2FA jika belum dimilikinya. Selanjutnya, tersedia juga QR *code* yang bisa dipindai oleh *tenant* menggunakan aplikasi 2FA untuk mendapatkan kode unik. Selain QR *code*, *tenant* juga bisa memasukkan kode yang terdapat di bawahnya ke dalam aplikasi 2FA sebagai alternatif jika QR *code* tidak dapat dipindai karena telah melewati batas waktu.



Gambar 3.25. Implementasi validasi *Step* 3

Validasi yang diimplementasikan, seperti yang ditunjukkan pada Gambar 3.25 mencakup kewajiban untuk mengisi kolom tersebut, yang hanya dapat berisi enam angka.



Gambar 3.26. Implementasi tampilan integrasi berhasil

Setelah proses integrasi telah selesai, tampilan pada Gambar 3.26 akan muncul. Tampilan tersebut berisi data-data *credential* untuk pengaturan API dan *webhook* tenant.

3.3.2 Sinkronisasi Otomatis Daftar Harga *Supplier* B dengan Whitelabel

Pada fitur sinkronisasi otomatis antar daftar harga yang tertera pada akun *supplier* B *tenant* dengan Whitelabel, digunakan beberapa metode seperti *webhook* dan *scheduled job*. Dalam melakukan sinkronisasi atau *mirroring* daftar harga secara otomatis, disediakan sebuah *endpoint* dari sisi Whitelabel yang dapat diimplementasikan pada sistem *supplier* B untuk dapat melakukan komunikasi. Fitur automasi ini bertujuan untuk mengeliminasi proses sinkronisasi secara manual yang sebelumnya perlu dilakukan oleh *tenant*. Tidak hanya untuk meningkatkan produktivitas, hal ini juga dapat mencegah terjadinya kerugian bagi *tenant* apabila *tenant* lupa melakukan sinkronisasi manual pada akun Whitelabel mereka.

A. *Requirement*

Sama seperti pengerjaan proyek sebelumnya, sinkronisasi otomatis daftar harga pada akun *supplier* B dan Whitelabel *tenant* juga memiliki beberapa *requirement* yang perlu diperhatikan.

A.1 *Endpoint Webhook*

Dalam mendukung adanya sinkronisasi secara otomatis dengan waktu *delay* seminim mungkin, digunakan *webhook* sebagai sarana berkomunikasi antar situs web. Proses ini menggunakan *webhook* karena tingkat efisiensinya yang lebih tinggi dibandingkan dengan API dalam *one-way communication*. Pada proses sinkronisasi otomatis, saat dimana *tenant* melakukan perubahan daftar harga pada situs web *supplier* B akan memicu pengiriman *request* perubahan daftar harga ke *endpoint* yang disiapkan pada situs web Whitelabel. *Endpoint* ini nantinya akan menyimpan permintaan perubahan daftar harga ke dalam *database* utama Whitelabel.

A.2 *Cron* Pengambilan *Request*

Terlepas dari *webhook*, penanganan semua *request* perubahan daftar harga yang masuk akan dilakukan oleh *cron* atau *scheduled job* yang akan berjalan setiap 1 menit. Proses yang dilakukan mencakup pengolahan *request* yang masuk, serta menjalankan *job* asinkron yang akan melakukan sinkronisasi pada akun Whitelabel *tenant*.

A.3 *Job* Penanganan *Request*

Job penanganan *request* akan dipanggil oleh *cron* pengambilan *request*. *Job* ini sendiri akan dijalankan untuk setiap *website* yang ada dalam *request* tersebut. Di dalam *job* akan disimpulkan produk apa saja yang perlu dilakukan *update* dan *delete* untuk dapat diproses secara lebih optimal dari sisi *query* ke *database*. Selanjutnya, akan dijalankan proses *update*, *create*, dan *delete* untuk masing-masing produk yang berubah pada *database tenant* berdasarkan kategori yang sudah ditetapkan sebelumnya.

A.4 *Cron* Penghapusan *Request*

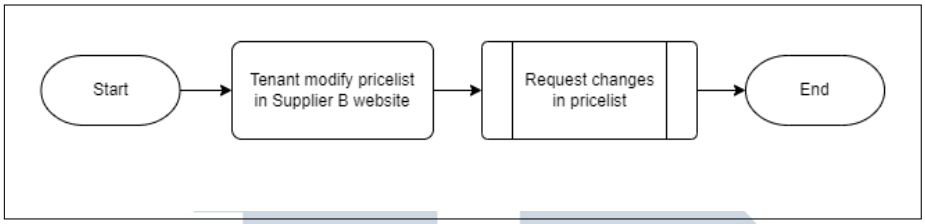
Hal terakhir yang perlu diperhatikan adalah penghapusan pada beberapa *request* perubahan yang sudah diproses dari *database* agar tidak terlalu banyak data yang tersimpan yang tidak diperlukan lagi. Proses ini dijalankan oleh *cron* yang berjalan setiap harinya.

B. Perancangan Sistem

Perancangan sistem untuk proses sinkronisasi otomatis antara daftar harga pada akun *supplier* B dan situs web Whitelabel *tenant* dibagi menjadi beberapa bagian terpisah sesuai dengan *requirement* yang ada. Rancangan yang diperlukan mencakup pembuatan *endpoint webhook*, *cron* pengambilan *request*, *job* penanganan *request*, dan diakhiri dengan *cron* penghapusan *request*.

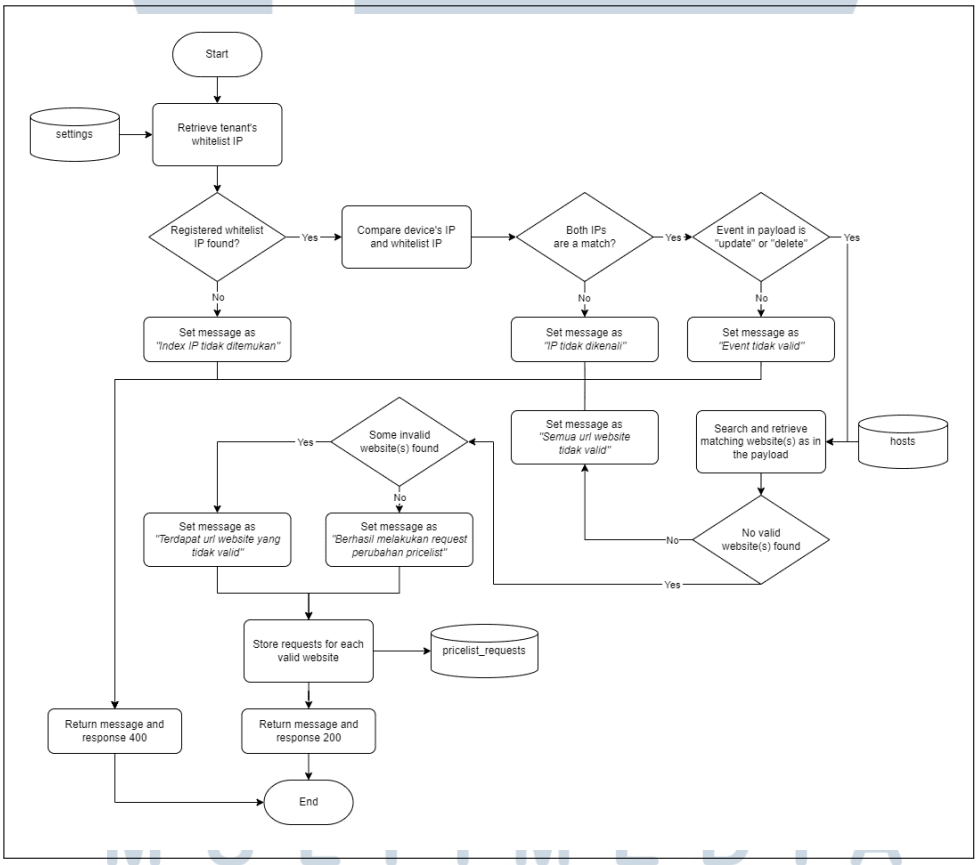
B.1 *Endpoint Webhook*

Dalam pemicuan *webhook*, ada beberapa hal yang perlu diperhatikan, dimana tidak semua *tenant* dapat memicu hal tersebut. Sinkronisasi otomatis antara *supplier* B dengan situs web Whitelabel hanya berlaku bagi *tenant* yang sudah melakukan integrasi antara akun Whitelabel dengan akun *supplier* B miliknya. Ketika syarat ini sudah sesuai, barulah proses sinkronisasi dapat dimulai.



Gambar 3.27. *Flowchart* utama *endpoint webhook*

Secara garis besar, proses tersebut digambarkan pada Gambar 3.27. Sinkronisasi otomatis diawali dengan adanya perubahan yang dilakukan oleh *tenant* pada situs web *supplier* B. Ketika perubahan dilakukan oleh *tenant*, maka *endpoint webhook* Whitelabel akan terpicu untuk membuat *request* perubahan daftar harga yang dikirim dari *website supplier* B.



Gambar 3.28. *Flowchart* proses penyimpanan *request* perubahan daftar harga

Kemudian Gambar 3.28 menggambarkan keseluruhan proses yang terjadi dalam *endpoint webhook* tersebut. Terdapat beberapa validasi yang diperlukan agar *request* dapat tersimpan untuk ditindaklanjuti nantinya, yaitu:

1. IP *address* terdaftar

   Hal pertama yang dicek adalah nilai IP dari *tenant*. Sistem akan mengambil nilai *whitelist* IP yang sudah didaftarkan *tenant* saat melakukan registrasi dan integrasi dengan akun *supplier* B dari tabel ”settings”. Jika ternyata tidak ditemukan adanya data IP *address* pada tabel tersebut, maka proses akan berhenti dan gagal dengan mengembalikan respons ”Index IP tidak ditemukan”.

2. *Whitelist* IP

   Jika ditemukan adanya nilai *whitelist* IP, maka akan dibandingkan apakah IP yang terdaftar sesuai dengan alamat IP *tenant* saat ini. Apabila IP tidak sesuai, maka proses akan mengembalikan respons *error* ”IP tidak dikenali” dan selesai.

3. Jenis *event*

   Ketika membuat suatu *request* perubahan daftar harga, di dalam *payload* yang disertakan, dibutuhkan jenis *event* yang dilakukan. *Event* yang valid hanya *update* dan *delete*. Sehingga *event* diluar itu akan menimbulkan *error* dengan pesan ”*Event* tidak valid”.

4. *Website*

   Jika *event* yang disertakan sudah sesuai, maka sistem akan melakukan pengecekan pada nilai-nilai *website* yang dicantumkan dalam *payload*. *Website* yang ada dapat berjumlah lebih dari 1, dan berfungsi untuk memberi tahu *website* Whitelabel mana saja yang ter-integrasi dengan 1 akun *supplier* B yang sama untuk dapat dilakukan sinkronisasi daftar harga. Jika tidak ada *website* Whitelabel yang valid, maka proses akan berhenti dengan pesan ”Semua url *website* tidak valid”.

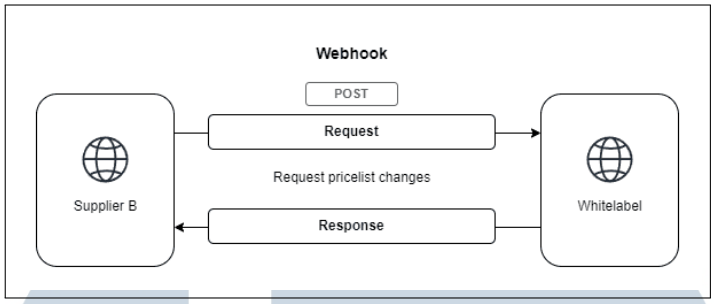
Jika *request* yang masuk sudah berhasil melalui validasi-validasi tersebut, perlu dicek lagi apakah minimal salah satu *website* yang dicantumkan valid atau tidak. Ketika ada *website* yang valid, maka proses akan terus berlanjut dengan mengatur pesan berhasil ”Terdapat url *website* yang tidak valid”, untuk nantinya dikembalikan ketika seluruh proses pada *webhook* ini sudah selesai. Diluar itu, jika semua validasi berjalan dengan lancar tanpa adanya hambatan, maka pesan berhasil yang ditetapkan adalah ”Berhasil melakukan *request* perubahan *pricelist*”.

Setelah menetapkan pesan berhasil, sistem akan menyimpan *request* tersebut ke dalam tabel ”pricelist_requests”. Pada kasus ini, jumlah *request* yang disimpan

bergantung pada jumlah *website* yang valid. Terakhir, sistem akan mengembalikan respons sesuai validasi dengan status 200 untuk keberhasilan, dan 400 untuk validasi yang gagal.

Gambar 3.29. Struktur sederhana tabel ”pricelist_requests”

Gambar 3.29 merupakan struktur sederhana untuk tabel ”pricelist_requests” yang telah dibuat guna menyimpan *request* perubahan yang masuk. Pada tabel ini terdapat beberapa kolom yang dapat membantu proses sinkronisasi daftar harga secara otomatis. Kolom id menjadi nilai unik identitas setiap baris, kolom ”website_id” menentukan pada *website* mana perubahan akan dilakukan. Kolom *”event”* menyimpan jenis perubahan yang akan dilakukan, *update* atau *delete*. Kolom *”payload”* merupakan kolom yang berisi semua data dan perubahan yang dilakukan oleh *tenant* pada situs web *supplier* B. Kolom *”timestamp”* merupakan waktu dilakukan perubahan daftar harga oleh *tenant*, dan berguna sebagai pembanding untuk meningkatkan efisiensi *query* ke *database*. Kolom lainnya berupa nilai-nilai waktu sebagai penanda kapan data terbuat, diperbarui, dan terhapus.

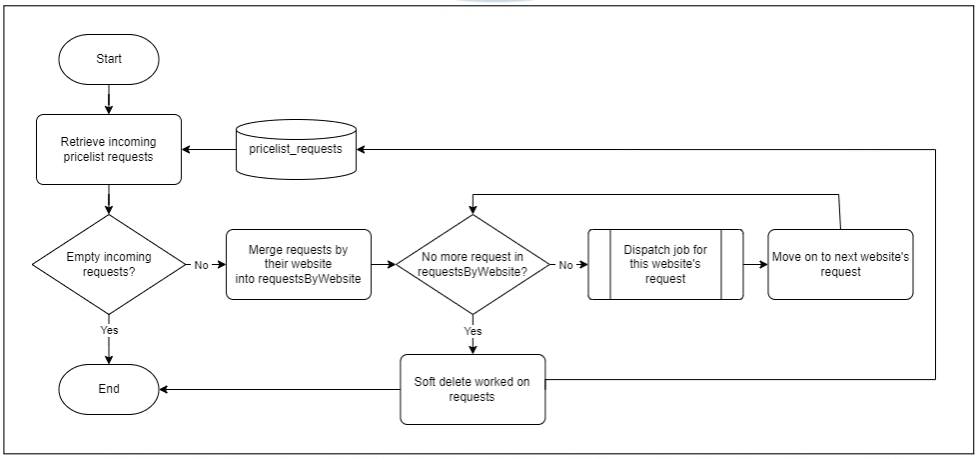


Gambar 3.30. Diagram alur *webhook* antara *supplier* B dengan Whitelabel

Alur *webhook* antara situs web *supplier* B dengan Whitelabel digambarkan pada Gambar 3.30. *Supplier* B akan melakukan *request* POST kepada Whitelabel, yang kemudian akan mengembalikan kembali *response* berupa status dan pesan.

B.2 *Cron* Pengambilan *Request*

Terlepas dari penyimpanan *request* ke dalam *database*, diperlukan *cron* yang berjalan setiap 1 menit untuk mengambil dan mengolah *request-request* perubahan daftar harga yang masuk.



Gambar 3.31. *Flowchart cron* pengambilan *request*

Gambar 3.31 menggambarkan proses pengambilan *request* yang dilakukan oleh *cron* atau *scheduled job* yang dimulai dari pengambilan *request* masuk dari tabel ”pricelist_requests”. Apabila ternyata tidak ada *request* perubahan daftar harga yang masuk dalam 1 menit tersebut, maka proses akan selesai dan menunggu pengambilan berikutnya. Tetapi jika terdapat *request* yang masuk, maka semua *request* tersebut akan digabungkan berdasarkan *website*-nya. Dalam hal ini,

mungkin saja terdapat lebih dari 1 *request* yang masuk untuk sinkronisasi pada 1 *website* yang sama. Contohnya jika *tenant* melakukan *update* dan *delete* produk sekaligus pada situs web *supplier* B.

Lalu untuk setiap *website* yang ada, akan dilakukan pemanggilan *job* untuk mengelola *request* tersebut. Setelah itu, jika tidak ada lagi permintaan yang tersisa, maka semua permintaan sebelumnya akan dihapus menggunakan metode *soft-delete*. Metode *soft-delete* akan memberikan nilai *deleted_at* pada data sebagai penanda bahwa permintaan telah dikerjakan.

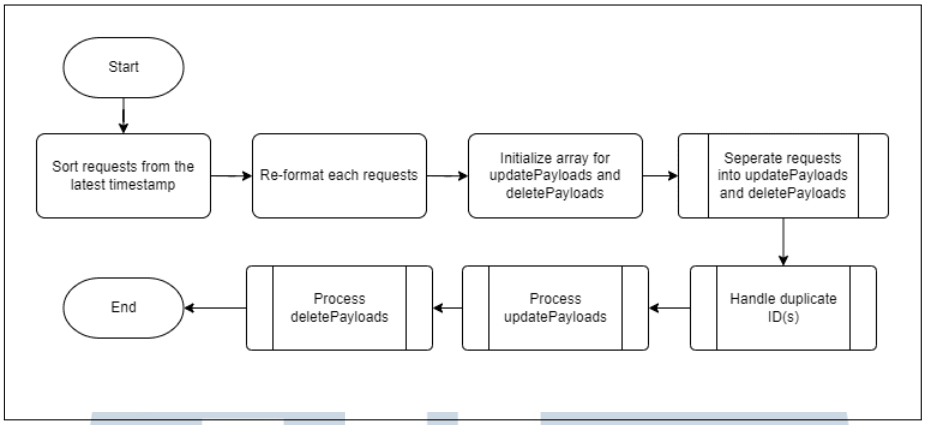
B.3 *Job* Penanganan *Request*

Job akan dijalankan untuk setiap *website* terlepas dari jumlah *request update* dan *delete* yang diterima untuk sinkronisasi daftar harga *website* tersebut. *Request* yang masuk ke dalam *job* memiliki format seperti yang ada pada Kode 3.1 di bawah ini.

```
$sorted_requests = [
  {
    "id": 1,
    "website_id": 1,
    "event": "delete",
    "payload": "",
    "timestamp": "2024-04-03 16:10:20",
    "deleted_at": null
  },
  {
    "id": 2,
    "website_id": 1,
    "event": "update",
    "payload": "",
    "timestamp": "2024-04-03 16:10:08",
    "deleted_at": null
  }
]
```

Kode 3.1: Contoh request setelah diurutkan

Pada *request* yang ada, terdapat beberapa data penting, seperti *website_id*, *event*, *payload*, dan *timestamp*. Nilai-nilai ini akan berguna selama *job* berjalan.



Gambar 3.32. *Flowchart* modul *Dispatch job*

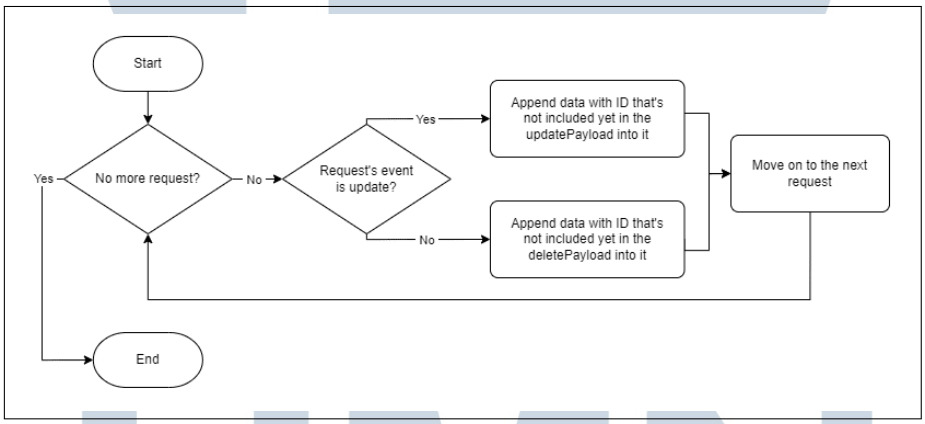
Job yang dijalankan untuk setiap *website* dijabarkan dalam Gambar 3.32. Di awal proses, beberapa *request* yang masuk yang ditujukan untuk 1 *website* yang sama, akan diurutkan dari *request* yang paling baru masuk berdasarkan nilai *timestamp*-nya. Pada contoh *request* yang tertera dalam Kode 3.1, nilai *payload* tidak ditampilkan, namun format *payload* berupa *string* hasil dari *array* yang di-*serialize*. Setelah diurutkan, maka setiap nilai *payload* yang ada dalam *request* akan diformat ulang untuk memudahkan proses berikutnya. Hasil format seperti yang ada pada Kode 3.2.

```
$formatted_payloads = [
  {
    "event": "delete",
    "payload": {
      "20": {
        "id": 20,
        "code": "go20",
        "timestamp": "2024-04-03 16:10:20"
      },
      "21": {
        "id": 21,
        "code": "dana10",
        "timestamp": "2024-04-03 16:10:20"
      }
    }
  },
  {
    "event": "update",
    "payload": {
      "22": {
```

```
            "id": 22,
            "code": "testGantiI5",
            "price": 5500,
            "desc": "Pulsa Indosat Rp 5.000",
            "timestamp": "2024-04-03 16:10:08"
        }
    }
  },
]
```

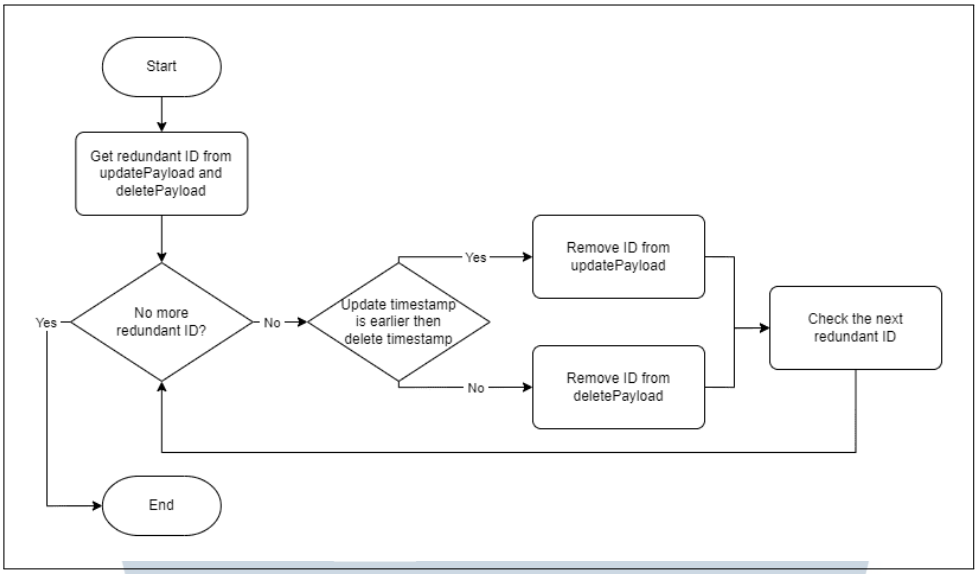
Kode 3.2: Contoh *payload* setelah diformat ulang

Setelah *payload* yang ada diformat ulang, maka akan disiapkan 2 buah *array* terpisah. Kedua *array* ini digunakan untuk membedakan produk mana saja yang akan diperbarui dan mana saja yang akan dihapus.



Gambar 3.33. *Flowchart* modul *Seperate requests into updatePayloads and deletePayload*

Modul untuk memisahkan *payload* yang ada ke dalam *array updatePayloads* dan *deletePayloads* digambarkan pada Gambar 3.33. Pada proses ini, akan dilakukan pengecekan pada variabel *event* yang ada pada setiap *payload*, jika nilai *event* adalah *update*, maka *payload* tersebut akan masuk ke dalam *array updatePayloads*. Hal ini juga berlaku pada *payload* dengan *event delete*, yang akan dimasukkan ke dalam *array deletePayloads*. Namun pada proses ini, tidak semua produk dalam *payload* langsung ditambahkan ke dalam *array* yang bersangkutan, melainkan hanya dimasukkan produk dengan ID yang belum ada dalam *array* tersebut. Hal ini dilakukan agar *query* tidak dilakukan secara berulang untuk produk yang sama untuk menjaga efisiensi. Jika sudah tidak ada *payload* yang belum disortir ke dalam *array* tertentu, maka modul ini selesai.



Gambar 3.34. *Flowchart* modul *Handle Duplicate Ids*

Setelah *payload* dipisahkan ke dalam *array* yang sesuai dengan nilai *event* dengan menjaga kombinasi ID produk seminim mungkin, langkah selanjutnya adalah meminimalkan kombinasi ID produk antar *array*. Modul ini bertujuan untuk meningkatkan efisiensi *query* ke *database*. Proses ini digambarkan pada Gambar 3.34. Pertama-tama, akan diambil terlebih dahulu ID-ID produk yang sama di antara *array updatePayloads* dan *deletePayloads*. Kemudian akan dilakukan pengecekan nilai *timestamp* untuk produk dengan ID tersebut. Berikut contoh penyisihan yang dilakukan jika ID yang berulang adalah ”go20”:

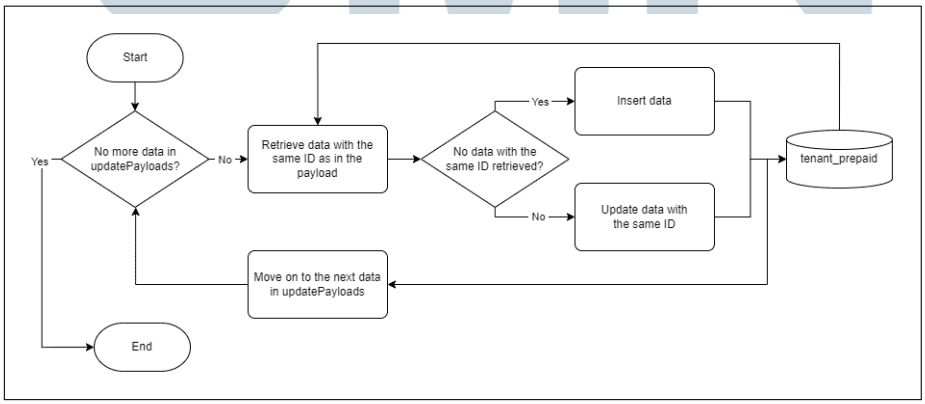
1. *Delete*
   Nilai *timestamp* produk dengan ID ”go20” pada *array updatePayloads* dan *deletePayloads* akan dibandingkan. Jika nilai *timestamp* yang ada pada *updatePayloads* lebih dahulu dibanding yang ada pada *deletePayload*, dapat disimpulkan bahwa meskipun produk tersebut di-*update*, pada akhirnya produk tersebut dihapus oleh *tenant*. Oleh karena itu, produk tersebut dapat dihapus dari *array updatePayloads* agar nantinya produk dapat langsung dihapus tanpa perlu diperbarui terlebih dahulu.

2. *Update*
   Di sisi lain, jika *timestamp* pada *deletePayloads* lebih dahulu dibanding yang ada pada *updatePayload*, maka produk tersebut dapat dihapus dari *deletePayloads* karena meskipun dihapus, nantinya produk tersebut akan dibuat lagi.

Hal ini akan terus diulangi hingga semua produk dengan ID yang terdaftar pada kedua *array* sudah tidak ada lagi. Hasil yang diharapkan dari modul ini dan sebelumnya adalah daftar produk tanpa ID yang berulang, baik di antara setiap *array* maupun antar *array*. Kode 3.3 di bawah ini merupakan contoh *array updatePayloads* dan *deletePayloads*.

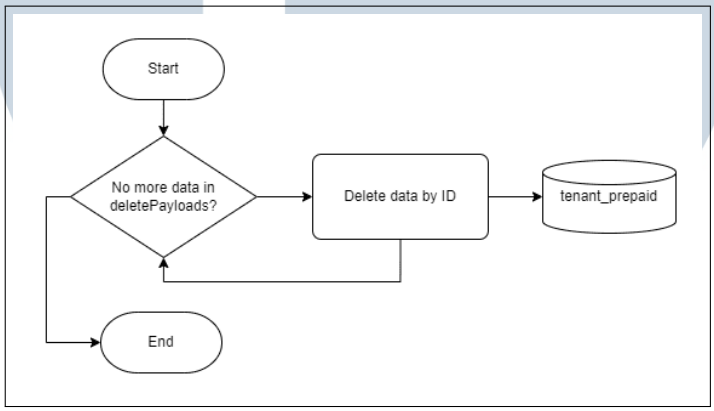
```
$updatePayloads = {
  "22": {
    "id": 22,
    "code": "testGantiI5",
    "price": 5500,
    "desc": "Pulsa Indosat Rp 5.000",
    "timestamp": "2024-04-03 16:10:08"
  }
}

$deletePayloads = {
  "20": {
    "id": 20,
    "code": "go20",
    "timestamp": "2024-04-03 16:10:20"
  },
  "21": {
    "id": 21,
    "code": "dana10",
    "timestamp": "2024-04-03 16:10:20"
  }
}
```

Kode 3.3: Contoh *array updatePayloads* dan *deletePayloads*



Gambar 3.35. *Flowchart* modul *Process updatePayloads*

Ketika produk-produk yang ada dalam *array updatePayloads* dan *deletePayloads* sudah rampung, maka akan dilakukan pemrosesan pada setiap *array*. Pertama, produk yang ada pada *array updatePayloads* akan diproses, seperti yang ada pada Gambar 3.35. Proses diawali dengan pengambilan data dari tabel ”tenant_prepaid” berdasarkan ID produk yang ada pada *array updatePayloads*. Jika ada produk yang ditemukan dalam tabel, maka akan dilakukan pembaruan data sesuai dengan yang ada pada *array*. Namun jika ternyata tidak ada produk yang tersimpan dalam tabel dengan ID tersebut, maka akan dibuat produk baru berdasarkan data pada *array*. Proses ini diulang terus-menerus untuk setiap produk yang terdaftar dalam *array updatePayloads* hingga produk terakhir.

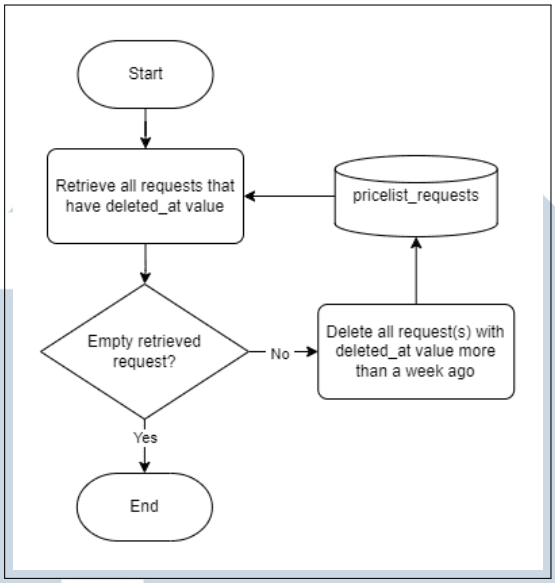


Gambar 3.36. *Flowchart* modul *Process deletePayloads*

Kemudian, pada Gambar 3.36, dijabarkan proses pengolahan produk pada *deletePayloads*. Sama seperti proses pengolahan pada *array updatePayloads*, iterasi akan dilakukan untuk setiap produk yang ada dalam *array*. Dalam iterasi tersebut, akan dilakukan penghapusan data produk pada tabel ”tenant_prepaid” berdasarkan nilai ID-nya.

B.4 *Cron* Penghapusan *Request*

Terakhir, diluar dari *webhook*, *cron*, dan *job* yang sebelumnya terurai, terdapat *cron* lain yang dijalankan setiap harinya untuk melakukan penghapusan data *request* dari tabel yang menyimpan *request-request* perubahan daftar harga.



Gambar 3.37. *Flowchart cron* penghapusan *request*

Gambar 3.37 merupakan *flowchart cron* yang digunakan untuk menghapus *request* masuk yang sudah dikerjakan. *Cron* ini diawali dengan pengambilan *request* yang sudah memiliki nilai pada kolom *deleted_at* pada tabel pricelist_requests. Adanya nilai *deleted_at* pada *request* menandakan bahwa *request* tersebut telah diproses, seperti yang sudah dijelaskan pada perancangan sistem *cron* pengambilan *request* sebelumnya. Setelah itu, jika ada *request* yang terambil, maka semua *request* dengan nilai *deleted_at* yang sudah lebih dari 1 minggu yang lalu dari hari ini akan dihapus dari tabel. Namun, jika tidak ada *request* yang terambil, *cron* akan langsung diberhentikan.

C. Implementasi Sistem

Implementasi sistem automasi pada sinkronisasi daftar harga yang dapat ditampilkan berupa testing respons *endpoint webhook* dan hasil perubahan data sesuai dengan perubahan yang dilakukan *tenant*.

C.1 *Endpoint Webhook*

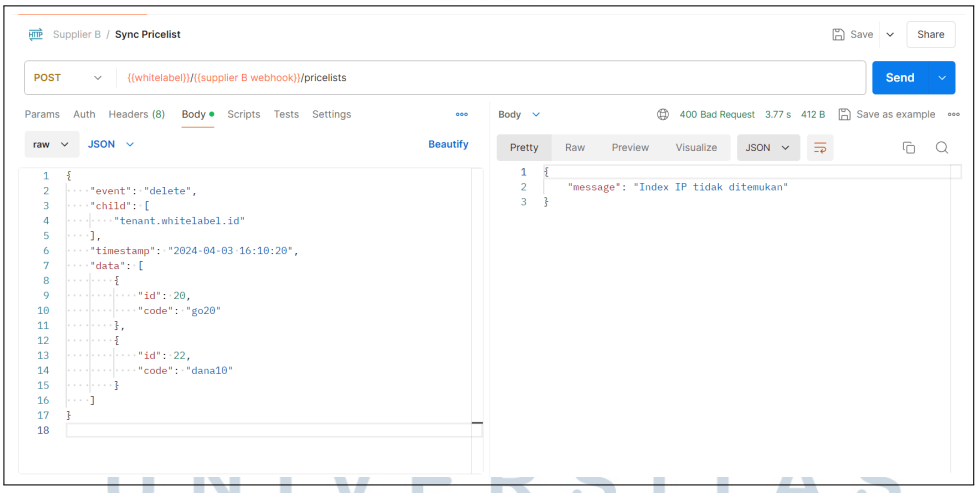
Implementasi *endpoint webhook* diterapkan pada aplikasi Postman dengan beberapa *test case* yang ada. Pengujian *endpoint webhook* dilakukan untuk memastikan apakah respons yang dihasilkan oleh *endpoint* sudah sesuai dengan

yang diharapkan. *Request* yang dikirim dalam bentuk JSON memiliki beberapa nilai, yaitu sebagai berikut:

- *Event*
  Jenis aksi yang ingin dilakukan, apakah *update* atau *delete*
- *Child*
  Array yang berisi daftar *website* yang terhubung dengan akun *supplier* B yang mengirimkan *request*
- *Timestamp*
  Berupa nilai jam dan tanggal *request* dilakukan
- Data
  Array yang berisi daftar produk yang ingin disinkronisasi beserta detailnya

Beberapa pengujian respons *endpoint webhook* yang dilakukan meliputi hal-hal berikut:

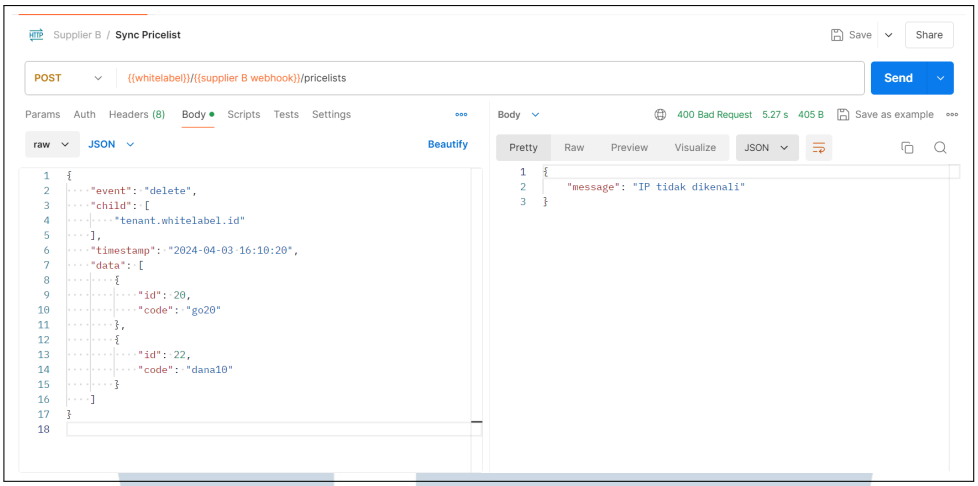
1. Gagal - Alamat IP tidak ditemukan



Gambar 3.38. *Test case* - Tidak ada indeks *whitelist* IP terdaftar

Request yang dikirimkan dalam Gambar 3.38 memiliki semua nilai yang valid. Namun pengecekan sudah dimulai dari pemeriksaan alamat IP. Pada kasus ini, nilai *whitelist* alamat IP tidak ditemukan dalam tabel ”settings” *tenant*. *Request* tersebut mengembalikan respons gagal dengan pesan sesuai dengan gambar di atas.

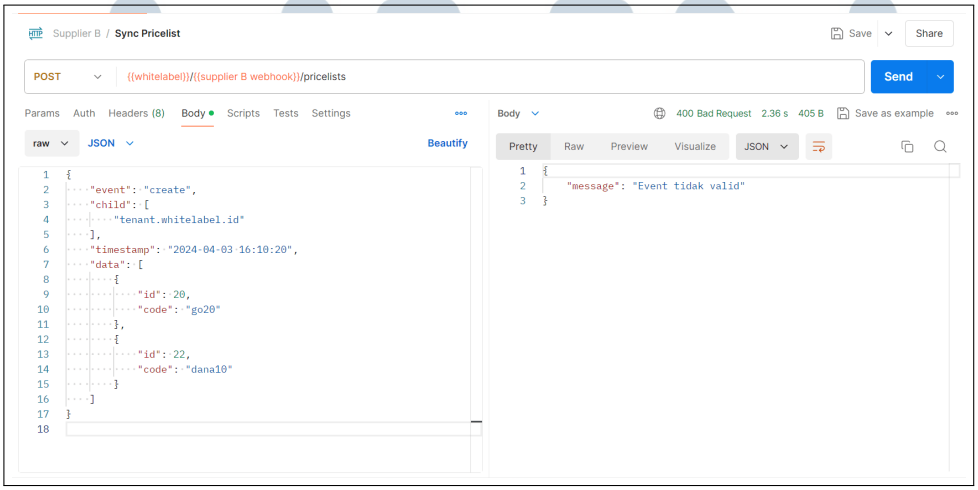
2. Gagal - Alamat IP tidak dikenali



Gambar 3.39. *Test case* - IP tidak dikenali

Request yang dikirimkan dalam Gambar 3.39 juga memiliki nilai yang valid serta *whitelist* IP juga ditemukan dalam *database*. Namun, pada kasus ini, alamat IP yang mengirimkan *request* ini tidak sesuai dengan nilai alamat IP yang terdaftar dalam *whitelist*. Oleh karena itu, *request* mengembalikan respons gagal juga.

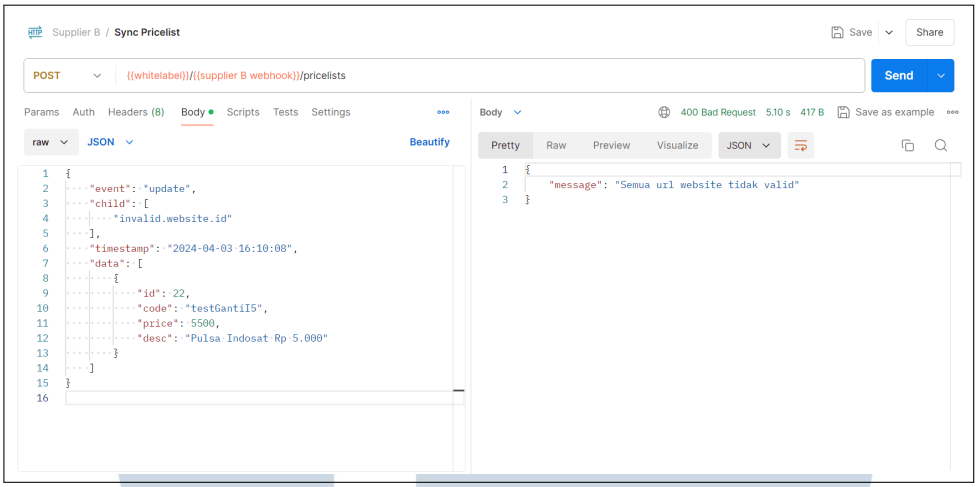
3. Gagal - Jenis *event* tidak valid



Gambar 3.40. *Test case* - *Event* tidak valid

Request yang dikirimkan dalam Gambar 3.40 memiliki nilai *event* yang tidak valid, yang pada contoh ini adalah *create*. Maka, *request* mengembalikan respons gagal yang memberitahu bahwa *event* yang disertakan tidak valid.

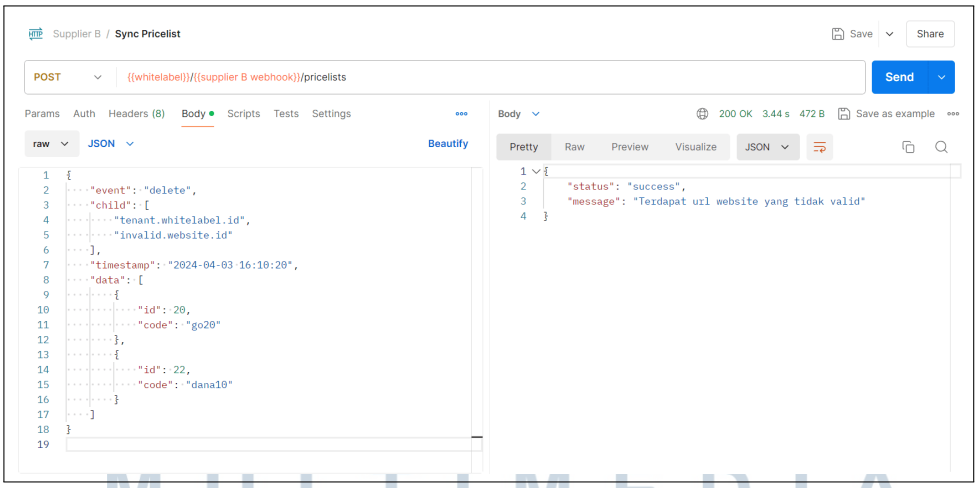
4. Gagal - Semua *website* tidak valid



Gambar 3.41. *Test case* - Semua *website* tidak valid

Request yang dikirimkan dalam Gambar 3.41 memiliki nilai *website* Whitelabel yang tidak valid. Sehingga, *request* mengembalikan respons gagal yang memberitahu bahwa *website* yang disertakan dalam *request* tidak ada yang valid.

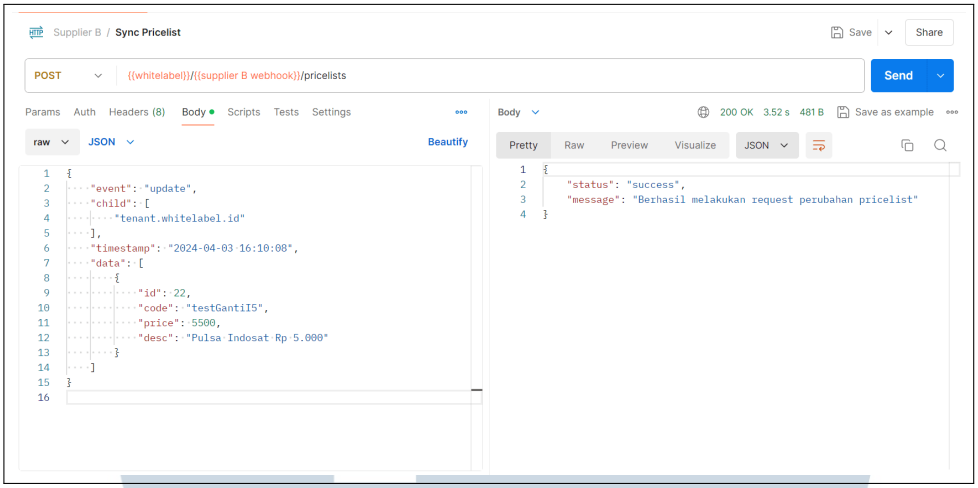
5. Berhasil - Terdapat *website* tidak valid



Gambar 3.42. *Test case* - Terdapat *website* tidak valid

Request yang dikirimkan dalam Gambar 3.42 memiliki nilai *website* Whitelabel yang valid dan tidak valid. Karena salah satu *website* valid, maka respons yang dikembalikan tetap berhasil, hanya saja disertakan pesan bahwa ada *website* yang tidak valid.

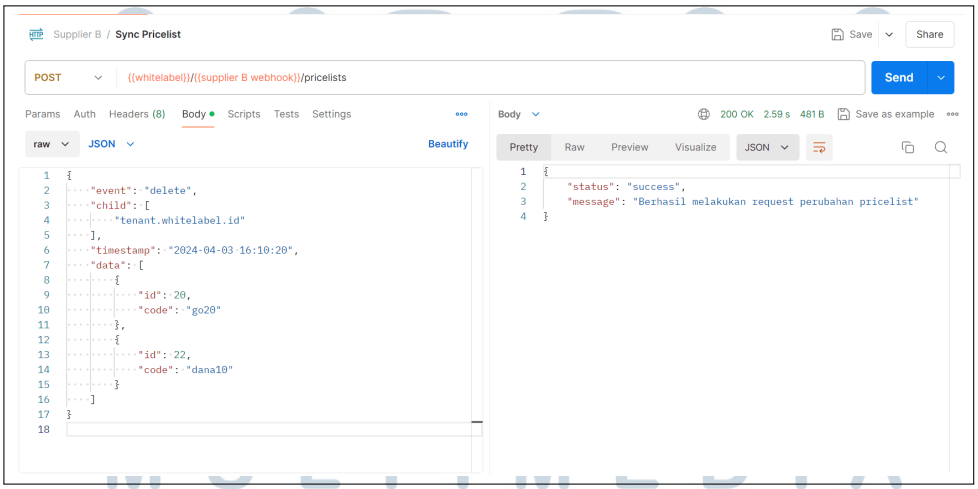
6. Berhasil - *Event update*



Gambar 3.43. *Test case* - Request untuk *event update*

Request yang dikirimkan dalam Gambar 3.43 berhasil melalui semua validasi yang ada. Oleh karena itu, respons yang dikembalikan juga merupakan respons berhasil. Pada contoh ini, *request* yang dikirimkan adalah untuk *event update*.

7. Berhasil - *Event delete*

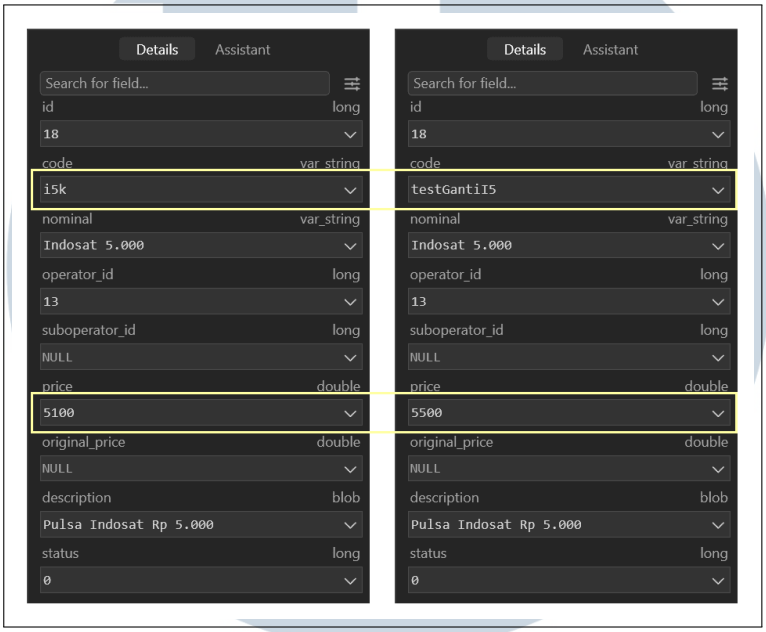


Gambar 3.44. *Test case* - Request untuk *event delete*

Request yang dikirimkan dalam Gambar 3.44 juga berhasil melalui semua validasi yang ada dan mengembalikan respons berhasil. Berbeda dengan sebelumnya, *request* yang dikirimkan kali ini adalah untuk *event delete*.

C.2 *Cron* dan *Job*

Hasil dari penggunaan cron dan job dapat langsung dilihat pada perbandingan daftar harga sebelum dan sesudah adanya perubahan dan sinkronisasi.



Gambar 3.45. Hasil perbandingan sebelum dan sesudah sinkronisasi daftar harga

Gambar 3.45 menampilkan hasil implementasi *cron* dan *job* dalam melakukan sinkronisasi daftar harga untuk *event update*. Pada kasus ini, *tenant* telah melakukan perubahan daftar harga pada situs web *supplier* B. Detail produk yang diganti yaitu harga dan kode produk. Dalam skenario ini, kode produk bermula dari ”i5k”, dan kemudian diubah menjadi ”testGantiI5”. Selain itu, dilakukan juga perubahan pada harga produk, dari 5100 menjadi 5500. Gambar 3.45 menggambarkan perubahan yang terjadi pada daftar harga di situs web Whitelabel milik *tenant*.

Gambar di sisi kiri merupakan detail produk sebelum perubahan pada situs web *supplier* B terjadi. Kemudian, ketika *tenant* melakukan perubahan harga dan kode produk tersebut, proses sinkronisasi otomatis berjalan untuk menyelaraskan detail produk. Berjalannya proses sinkronisasi menggunakan *cron* dan *job* pada akhirnya mengubah dan memperbarui kode dan harga pada produk tersebut pada daftar harga Whitelabel, seperti yang ada pada gambar di sisi kanan. Selain *update*, ketika terjadi *event delete*, maka produk yang dihapus dari situs web *supplier* B juga akan terhapus dari daftar produk Whitelabel milik *tenant*.

D. Evaluasi

Dari hasil implementasi sistem tersebut, kemudian dilakukan evaluasi pada fitur automasi sinkronisasi yang telah dibuat. Evaluasi ini dilakukan bersama anggota tim QA (*Quality Assurance*). Pada proses QA, *test case* yang ada dibuat bersama juga dengan anggota tim QA. Pada setiap *test case* yang ada, dilakukan pengecekan pada dua *environment* yang berbeda, yaitu *staging* dan lokal. *Environment staging* merupakan lingkungan yang telah dibuat sedemikian rupa untuk lebih menyerupai situasi *production* nantinya, sedangkan *environment* lokal merupakan lingkungan yang ditemui selama proses *development* berlangsung. Pengecekan pada *environment staging* dilakukan oleh tim QA, sedangkan pada *environment* lokal dilakukan berbarengan ketika mengembangkan sistem. Jika semua *test case* QA telah berhasil mendapat *output* yang diharapkan, maka proses evaluasi dari keseluruhan pengembangan fitur ini juga telah selesai.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Saat berlangsungnya program magang, berbagai hambatan dan kendala telah muncul di sepanjang perjalanan. Beberapa di antaranya termasuk:

1. Kurang familier dengan *framework* dan beberapa *tools* yang digunakan selama pengerjaan proyek.
2. Penulisan kode yang masih kurang rapi, efektif, dan konsisten seperti penulisan yang sudah ada sebelumnya.
3. *Commit* Github kurang rapi yang redundan dan sebenarnya dapat diperbaiki dengan sebuah metode tertentu, namun belum terbiasa dengan *tools commit* yang tersedia dan GUI Git yang digunakan.
4. Adanya implementasi beberapa metode baru yang kurang dimengerti dan dikuasai, seperti penanganan API, pembuatan *job*, dan penggunaan *cache lock*.

3.4.2 Solusi

Dalam mengatasi kendala-kendala yang dihadapi, berikut merupakan solusi yang diupayakan:

1. Membaca berbagai dokumentasi yang berkaitan dengan *framework* dan *tools* tersebut, serta bertanya kepada senior yang sudah lebih paham.
2. Belajar menulis kode yang lebih baik dan *query* yang lebih efisien dengan memperhatikan referensi sistem penulisan kode yang sudah ada sebelumnya dan dari hasil *code review* yang diterima.
3. Mempelajari dokumentasi penggunaan GUI Git yang lebih mendalam untuk dapat menghasilkan *commit* yang lebih bersih dan rapi.
4. Mengambil dan mempelajari metode-metode baru dari referensi kode yang sudah ada, dokumentasi, dan bertanya kepada senior.