BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama magang sebagai Backend Developer intern di PT Omni Digitama Internusa, supervisor dipercayakan kepada Bapak Joseph Kondar Halomoan yang memimpin tim Shoped (Shopee dan Tokopedia). Selama magang, kegiatan magang dibimbing oleh Bapak Ainur Riza Abdul Haris Ramadhan, yang bertanggung jawab dalam memberikan tugas setelah berdiskusi dengan Bapak Joseph. Tugas diberikan melalui *backlog* di Jira, dengan pelaksanaan kerja mengikuti metode *sprint* yang berjalan selama dua minggu. Tenggat waktu penyelesaian tugas bersifat dinamis, tugas dapat diselesaikan pada *sprint* yang sedang berjalan atau dilanjutkan pada *sprint* berikutnya.

Setiap hari Jumat, tim Shoped melakukan *sprint grooming* untuk meninjau tugas-tugas yang sedang dikerjakan pada *sprint* yang sedang berjalan. Selain itu, setiap dua minggu sekali diadakan *sprint retrospective* dan *sprint planning* untuk mengevaluasi kinerja *sprint* sebelumnya dan merencanakan tugas untuk *sprint* berikutnya. Tim Shoped juga melakukan *daily Scrum* setiap harinya pada jam 10.30 pagi untuk membahas *progress* tugas atau membahas masalah yang ditemukan pada sistem. Seluruh proses tersebut tim Shoped menggunakan *Google Meet* sebagai media komunikasi. Untuk memperlancar komunikasi dan koordinasi antar anggota tim dan dengan tim lain, seperti Tim Quality Assurance dan *Product Manager*, tim Shoped menggunakan Discord sebagai media perantaranya.

3.2 Tugas yang Dilakukan

Uraian pelaksaan magang yang dilakukan dapat dilihat pada Tabel 3.1.

Tabel 3.1. Uraian pelaksanaan praktik kerja magang

Minggu Ke	Pekerjaan yang dilakukan
1	– Melakukan *meeting* untuk membahas konsep *multilocation*, pergudangan, toko,dan *shop* pada sistem Ruparupa. – Melakukan *prototyping* sistem *add multilocation*.
2	– Melakukan pengembangan sistem *backend* untuk fitur *add multilocation*.
3	– Melakukan pengembangan sistem *backend* untuk fitur *add multilocation*. – Melakukan pengembangan sistem UI *dashboard multilocation* dan fitur *add multilocation*. – Melakukan *self test* sistem *add multilocation*.
4	– Melakukan pengembangan sistem *backend* untuk fitur *switch multilocation*. – Melakukan pengembangan sistem UI *switch multilocation*. – Melakukan *self test* untuk fitur *switch multilocation*.
5	– Melakukan *testing* dengan QA untuk fitur *add and switch multilcation*.
	Lanjut pada halaman berikutnya

Tabel 3.1 Uraian pelaksanaan praktik kerja magang (lanjutan)

Minggu Ke	Pekerjaan yang dilakukan
6	– Melakukan *breefing* dan *listing service-service* yang harus di pindahkan dari repositori V1 ke V2. – Melakukan *screening endpoint-endpoint* yang harus dipindahkan.
7	– Memindahkan *endpoint* GET get-bulk-process-queue. – Memindahkan *endpoint* POST edit-bulk-process-queue. – Melakukan *self test endpoint* GET get-bulk-process-queue dan POST get-bulk-process-queue.
8	• Memindahkan *endpoint* GET get-bulk-process-log. • Memindahkan *endpoint* POST edit-bulk-process-log. • Melakukan *self test endpoint* GET get-bulk-process-log dan POST edit-bulk-process-log. • Memindahkan *endpoint* GET tokopedia-get-discount. • Memindahkan *endpoint* GET shopee-get-discount. • Melakukan *self test endpoint* GET tokopedia-get-discount dan GET shopee-get-discount.
	Lanjut pada halaman berikutnya

Tabel 3.1 Uraian pelaksanaan praktik kerja magang (lanjutan)

Minggu Ke	Pekerjaan yang dilakukan
9	– Memindahkan *endpoint* POST tokopedia-edit-discount. – Memindahkan *endpoint* POST shopee-edit-discount. – Melakukan *self test endpoint POST tokopedia-edit-discount* dan *POST edit-bulk-process-log*. – Memindahkan *endpoint* POST tokopedia-item-update. – Memindahkan *endpoint* GET tokopedia-get-item-all. – Melakukan *self test endpoint* GET POST tokopedia-item-update dan GET tokopedia-get-item-all.
10	• Memindahkan *endpoint* GET tokopedia-get-item. • Memindahkan *endpoint* GET shopee-get-item-all. • Memindahkan *endpoint* GET shopee-get-item. • Melakukan *self test endpoint* POST tokopedia-edit-discount, POST edit-bulk-process-log, dan GET shopee-get-item. • Memindahkan *endpoint* GET shopee-get-item-detail. • Memindahkan *endpoint* PUT shopee-edit-item-detail. • Melakukan *self test endpoint* GET shopee-get-item-detail dan PUT shopee-edit-item-detail.
	Lanjut pada halaman berikutnya

Tabel 3.1 Uraian pelaksanaan praktik kerja magang (lanjutan)

Minggu Ke	Pekerjaan yang dilakukan
11	• Memindahkan *endpoint* GET shopee-get-store. • Memindahkan *endpoint* GET vendor-get-warehouses. • Memindahkan *endpoint* GET tokopedia-get-store • Melakukan *self test endpoint* GET shopee-get-store, GET tokopedia-get-store, dan GET vendor-get-warehouses.
12	• Melakukan *breefing* permasalahan dan solusi dari notifikasi *flash sale*. • Mengembangkan notifikasi Discord ketika ada *error flash sale*.
13	• Mengembangkan fitur notifikasi Discord ketika ada *error flash sale*. • Melakukan *self test* fitur notifikasi Discord ketika ada *error flash sale*.
14	• Melakukan *breefing* mengenai konsep *promo* dan *existing batch promo* barang Tokopedia. • Mempelajari fitur *existing batch promo* barang Tokopedia. • Mengembangkan fitur *batch auto retry promo* barang di Tokopedia.

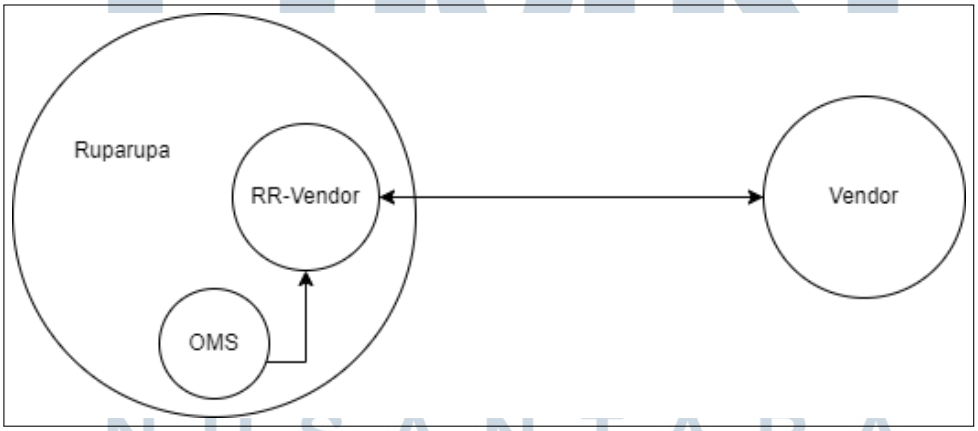
Lanjut pada halaman berikutnya

Tabel 3.1 Uraian pelaksanaan praktik kerja magang (lanjutan)

Minggu Ke	Pekerjaan yang dilakukan
15	Mengembangkan fitur *batch auto retry promo* barang di Tokopedia.
16	– Mengembangkan fitur *batch auto retry promo* barang di Tokopedia. – Melakukan *self test* fitur *batch auto retry promo* barang di Tokopedia.
17	Melakukan *testing* fitur *batch auto retry promo* barang di Tokopedia bersama QA.

3.3 Uraian Pelaksanaan Magang

3.3.1 Gambaran Umum Sistem



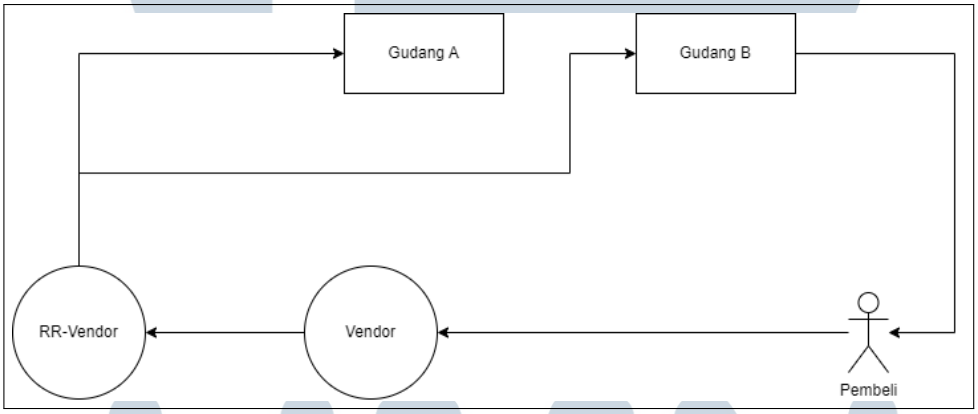
Gambar 3.1. Gambaran Umum Sistem di Ruparupa

Gambaran umum sistem pada Ruparupa dapat dilihat di Gambar 3.1. Di sistem Ruparupa terdapat penghubung antara vendor dan Ruparupa yang dapat

disebut RR-Vendor. RR-Vendor berperan sebagai jembatan dan integrasi antara sistem vendor dan Ruparupa. RR-Vendor bertanggung jawab untuk memastikan semua logika proses bisnis pada vendor berjalan sesuai dengan ketentuan dan standar operasional Ruparupa. Dengan adanya RR-Vendor, alur kerja antara Ruparupa dan vendor menjadi lebih efisien, terstruktur, dan terkontrol.

OMS atau *Order Management System* digunakan oleh Ruparupa untuk memantau dan mengelola seluruh proses terkait pemesanan di vendor. Melalui OMS, Ruparupa dapat mengakses informasi seperti *invoice*, daftar gudang, pengelolaan stok, hingga pengaturan harga produk.

3.3.2 Gambaran Umum Proses Bisnis



Gambar 3.2. Gambaran Umum Proses Bisnis di Ruparupa

Gambar 3.2 merupakan gambaran umum dari proses bisnis di Rupaupa. Ketika ada *order* dari vendor. Jika terjadi *order* di vendor maka vendor akan mengirim informasi order ke RR-Vendor bahwa terjadi pembelian. Selanjutnya sistem RR-Vendor akan mengolah dan melakukan validasi terhadap order tersebut. Jika lolos dari tahap validasi maka barang-barang yang ada di order tersebut akan dikirim sesuai dengan ketersediaan di gudang terdekat.

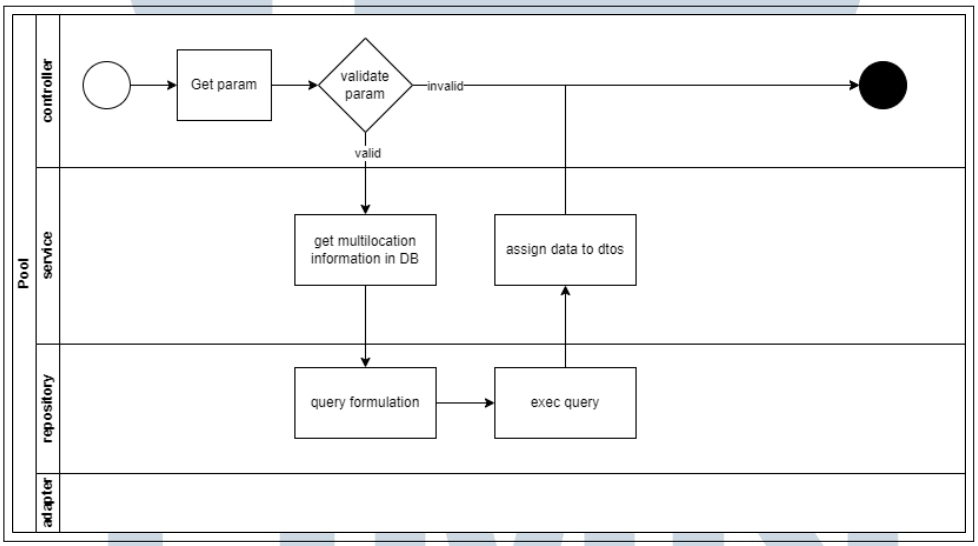
3.3.3 Perancangan

A. Membuat Fitur Add and Switch Multilocation

Fitur *Add and Switch Multilocation* adalah fitur baru yang ada di OMS. *Add multilocation* berfungsi untuk melakukan penambahan gudang atau toko baru bagi suatu *shop* tertentu. Penambahan gudang dilakukan agar distribusi

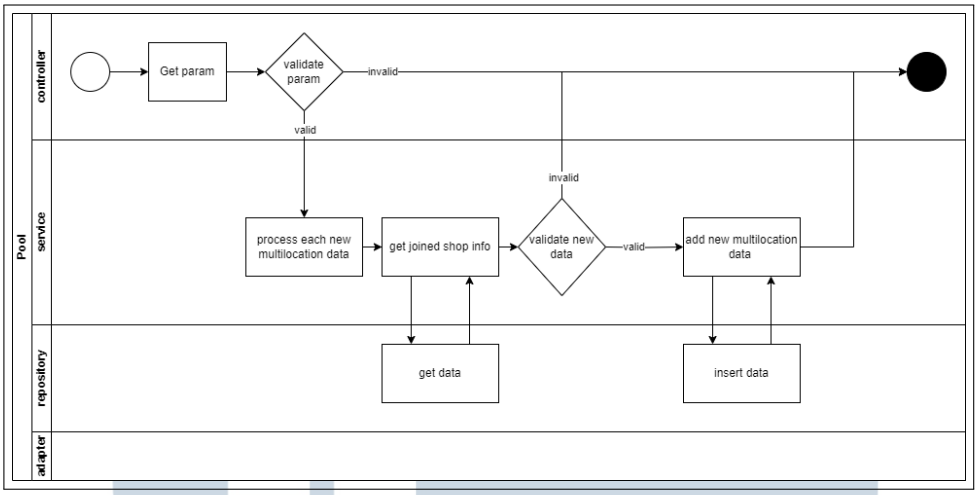
barang dapat dilakukan dengan optimal sehingga dapat menjangkau daerah-daerah sedangkan *switch multilocation* berfungsi untuk merubah *store code* gudang atau toko. Penukaran toko dilakukan jika terjadi penutupan toko atau performa toko tersebut menurun sehingga perlunya penggantian ke toko yang memiliki performa lebih optimal. Adapun *requirment* fitur sebagai berikut:

1. Fitur *add multilocation* dapat menambahkan gudang secara tepat.
2. Fitur *switch multilocation* dapat mengganti *store code* gudang secara tepat.
3. *Dashboard multilocation* menampilkan data yang tepat.
4. UI *dashboard multilocation* sesuai dengan permintaan *user*.



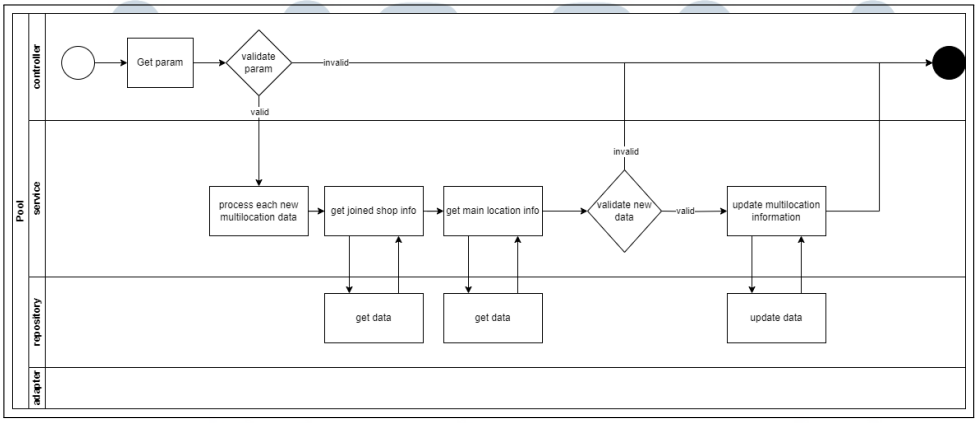
Gambar 3.3. *Swimlane Diagram Get Multilocation Info*

Gambar 3.3 adalah *swimlane diagram* dari proses *get multilocation* pada *dashboard Multilocation*. Pada *layer controller* semua parameter yang dikirim akan diproses dan dilakukan validasi jika lulus validasi maka akan lanjut ke *layer service*. Pada *layer service* parameter dari *layer controller* akan diteruskan ke *layer repository*. Pada *layer repository* akan dilakukan formulasi *query* berdasarkan parameter-parameter yang ada lalu *query* tersebut akan dijalankan. Data hasil *query* akan diteruskan ke *layer service* lalu akan dikembalikan ke *layer controller*. Setelah itu, *layer controller* akan mengirimkan data *multilocations* ke *user* dalam bentuk respons API yang sesuai dengan format.



Gambar 3.4. *Swimlane Diagram Add Multilocation*

Gambar 3.4 menunjukan proses bagaimana *add multilocation* bekerja. Proses dimulai dengan proses validasi parameter yang berisikan data *multilocation* baru lalu data tersebut diproses satu per satu. Didalam proses tersebut dilakukan validasi apakah datum *multilocation* baru *valid* atau datum tersebut belum pernah ada di *database*. Jika *valid* maka data tersebut akan dimasukkan ke *database* lalu akan dikirim respons API ke *user* dengan pesan berhasil. Sebaliknya jika *invalid* maka akan dikirim respons API ke *user* bahwa terdapat *error* dalam proses *add multilocation*.



Gambar 3.5. *Swimlane Diagram Switch Multilocation*

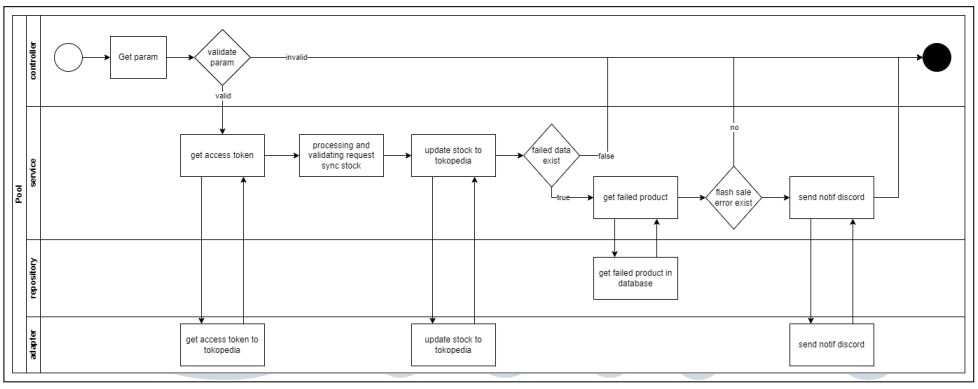
Gambar 3.5 menunjukan proses bagaimana *switch multilocation* bekerja. Proses dimulai dengan proses validasi parameter yang berisikan data *multilocation* baru. Lalu sistem akan mengecek apakah data tersebut sudah ada sebelumnya. Selanjutnya sistem akan mengecek apakah gudang yang akan diganti adalah *main location* jika tidak maka data *multilocation* akan diubah dengan yang baru lalu akan

dikirim respons API ke *user* dengan pesan berhasil. Sebaliknya jika gudang yang akan diganti adalah *main location* maka akan dikirim respons API ke *user* bahwa terdapat *error* dalam proses *switch multilocation*.

B. Membuat Fitur Notifikasi ke Discord

Fitur notifikasi yang dipindahkan dari Slack ke Discord berada di modul *order* seperti *order creation*. Terdapat peluang terjadinya kondisi yang krusial pada proses *order* seperti kejadian *hold order*, *hold kecamatan*, terbentuknya *order* dan *invalid* sku (*stock keeping unit*). Fitur notifikasi juga ditambahkan pada modul sinkronisasi stok produk, hal ini dikarenakan sering terjadinya kegagalan *update* stok produk yang disebabkan oleh promo sepihak dari vendor. Pada mulanya sku-sku yang terkena gagal karena promo sepihak tidak dapat dilihat oleh tim *merchandise* yang dapat menyebabkan terjadinya *incomplete order*. Adapun *requirment* fitur sebagai berikut:

1. Fitur notifikasi menampilkan secara tepat informasi mengenai produk yang mengalami *error* ke Discord.



Gambar 3.6. *Swimlane Diagram* Notifikasi *Flash Sale Error*

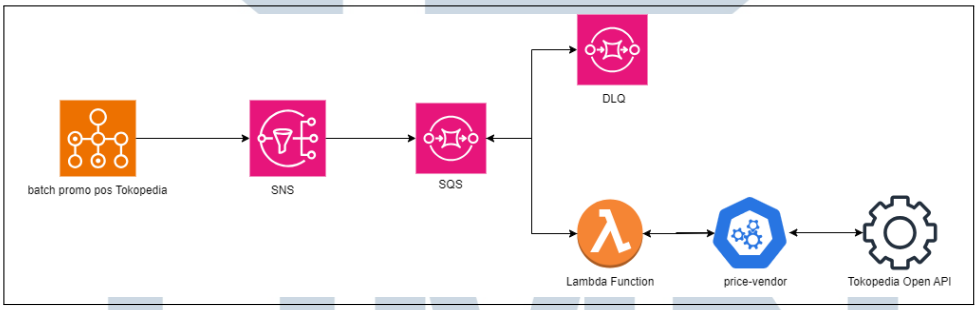
Gambar 3.6 menunjukkan alur dari notifikasi *flash sale error*. Proses dimulai dari sistem menerima parameter yang berisikan data produk yang ingin dilakukan *sync stock*, selanjutnya sistem akan mengambil *access token* dari tokpedia. Lalu sistem akan mengolah dan memvalidasi *request sync stock*. Setelah selesai diproses, sistem akan mengirim *payload update stock* ke Tokopedia, berikutnya *reponse* dari Tokopedia akan dicek apakah terdapat pesan *Flash Sale Error*. Jika ada maka akan dikirim notifkasi *error* ke Discord.

C. Membuat Fitur Batch Auto Retry Promo Pos Tokopedia

Fitur *batch auto retry promo* Pos Tokopedia yang dikembangkan memanfaatkan layanan AWS seperti SNS, Lambda, SQS, dan *Dead Letter Queue* (DLQ). SNS digunakan untuk mengirim pesan mengenai *promo* pos yang kemudian akan diteruskan ke Lambda. Dari Lambda pesan akan diteruskan ke *service* yang akan memproses logika bisnis *promo* pos. Proses *retry* akan terjadi jika terdapat pesan error balik tertentu dari *vendor*, proses tersebut akan diulangi sebanyak 5 kali. Jika proses pengajuan *promo* pos gagal sebanyak 5 kali maka pesan akan diteruskan ke DLQ. Adapun *requirment* fitur sebagai berikut:

1. Fitur *batch auto retry promo* POS Tokopedia dapat secara tepat menerapkan *promo* ke Tokopedia

Seluruh proses tersebut dapat dimonitor melalui layanan AWS *CloudWatch*. *CloudWatch* adalah layanan *monitoring*dari AWS, hal-hal yang dapat dimonitor adalah seluruh *resource* AWS dan aplikasi yang berjalan atau ada di AWS.



Gambar 3.7. Alur *Batch Promo* POS Tokopedia

Gambar 3.7 adalah alur dari *batch promo* POS Tokopedia. Proses dimulai dari formulasi *payload* promo-promo barang yang ingin dipasang di Tokopedia pada *repository batch promo* POS. Setelah *payload* yang selesai dibuat, *payload* tersebut akan dikirim ke SNS. SNS akan meneruskan ke SQS khusus yang menangani *promo* POS Tokopedia, SQS akan meneruskan *payload* ke *Lambda Function* yang berfungsi menentukan tujuan pengiriman dari *payload*. Dalam konteks ini *Lambda Function* akan meneruskan *payload* ke *repository* price-vendor. Pada price-vendor, *payload* akan diproses dan dirangkai ulang agar sesuai dengan format yang dapat diterima oleh Tokopedia. Setelah selesai dirangkai sesuai dengan strukturnya, *payload* tersebut akan dikirim ke Tokopedia *Open* API.

Proses *retry* akan terjadi apabila price-vendor mendapat *error message* tertentu pada respons balik API dari Tokopedia *open* API. Payload yang mengalami

error akan dimasukkan kembali ke SQS dan akan dilakukan *retry* sebanyak 5 kali. Apabila setelah 5 kali *retry* masih mendapat respons *error* yang sama maka *payload* akan dimasukkan ke DLQ.

D. Memindahkan Sebagian Service dari Repositori V1 ke V2

Clean Architecture adalah konsep arsitektur dalam pengembangan *software* yang mengedepankan modularitas tiap komponen menjadi beberapa *layer*[4]. Setiap layer memiliki perannya masing-masing dan hanya bisa berkomunikasi kepada komponen-komponen melalui *interface*.

Pada sistem Ruparupa, *interface* atau *layer* yang digunakan ada 4 yaitu *controller, service, repository, dan adapter*. Berikut fungsi dari setiap *layer*.

1. *Controller* berfungsi untuk menerima dan mengolah *payload* dari sebuah request ke suatu *endpoint*.
2. *Service* berfungsi untuk mengolah *payload* sesuai dengan logika bisnis.
3. *Repository* berfungsi sebagai penghubung antara *service* yang ingin mengakses atau menyimpan data ke *database*.
4. *Adapter* berfungsi sebagai penghubung antara sistem dengan *microservice* lain atau *open API service* suatu platform.

Adapun *requirment* dari pemindahan *service* dari repositori V1 ke V2 sebagai berikut:

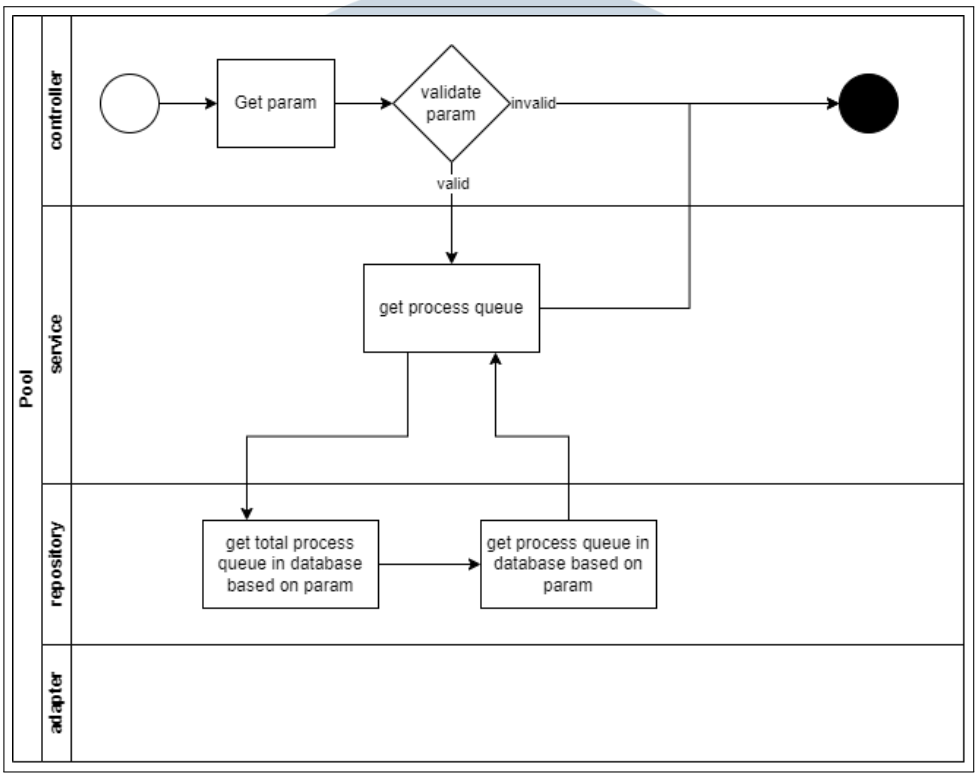
1. *Service* yang dipindahkan dapat bekerja seperti pada saat masih di repositori V1.

Service-service atau *endpoint* yang dipindahkan dari repositori V1 ke V2 adalah sebagai berikut.

1. GET get-bulk-process-queue → GET get-bulk-process-queue
2. POST edit-bulk-process-queue → POST edit-bulk-process-queue
3. GET get-bulk-process-log → GET get-bulk-process-log
4. POST edit-bulk-process-log → POST edit-bulk-process-log
5. GET tokopedia-get-discount → GET get-special-price-discount

6. POST tokopedia-edit-discount → POST edit-special-price-discount
7. POST tokopedia-item-update → POST tokopedia-item-update
8. GET tokopedia-get-item-all → GET tokopedia-get-item-all
9. GET tokopedia-get-item → GET tokopedia-get-item
10. GET shopee-get-item-all → GET shopee-get-item-all
11. GET shopee-get-item → GET shopee-get-item
12. GET shopee-get-discount → GET get-special-price-discount
13. POST shopee-edit-discount → POST edit-special-price-discount
14. GET shopee-get-item-detail → GET shopee-get-item-detail
15. PUT shopee-edit-item-detail → PUT shopee-edit-item-detail
16. GET shopee-get-store → GET get-store-list
17. GET vendor-get-warehouses → GET get-store-warehouse
18. GET tokopedia-get-store → GET get-store-list

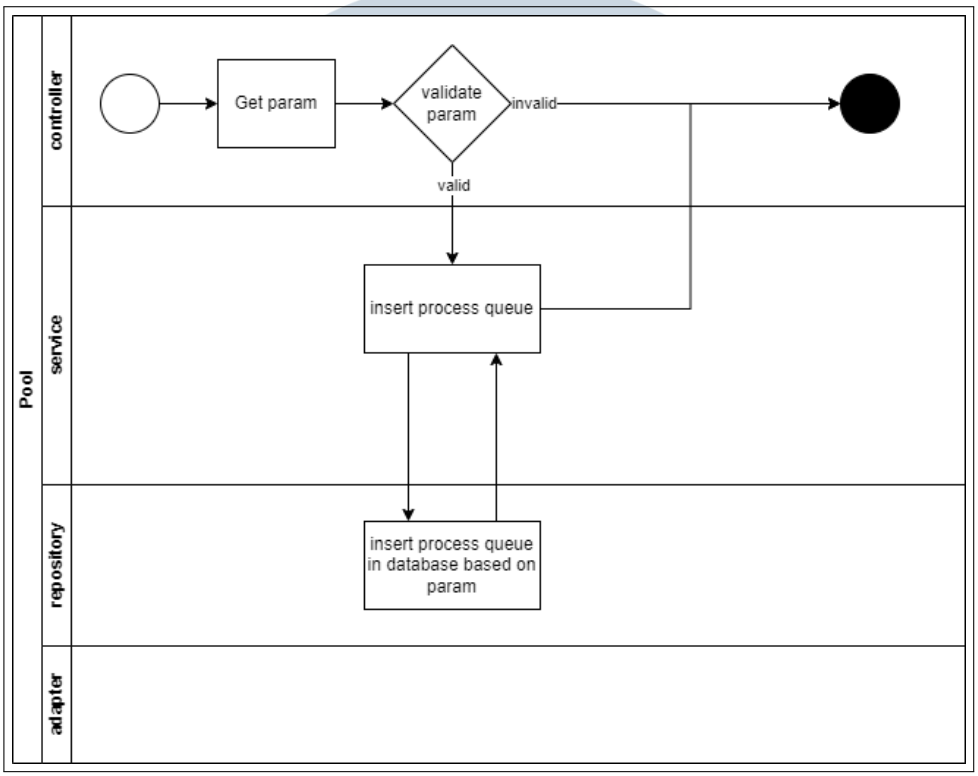
D.1 GET get-bulk-process-queue



Gambar 3.8. *Swimlane Diagram* GET get-bulk-process-queue

Gambar 3.8 menunjukkan alur dari GET get-bulk-process-queue. Proses dimulai dari sistem menerima parameter yang berisikan data seperti *filename, status* dan *vendor*. Parameter-parameter tersebut akan diteruskan sampai ke *repository layer*. Pada *repository layer* sistem akan meramu *query* sesuai dengan parameter yang ada. Data hasil *get query* akan diteruskan ke *layer service* lalu akan dikembalikan ke *layer controller*. Setelah itu, *layer controller* akan mengirimkan data *process queue* ke *user* dalam bentuk respon API yang sesuai dengan format.

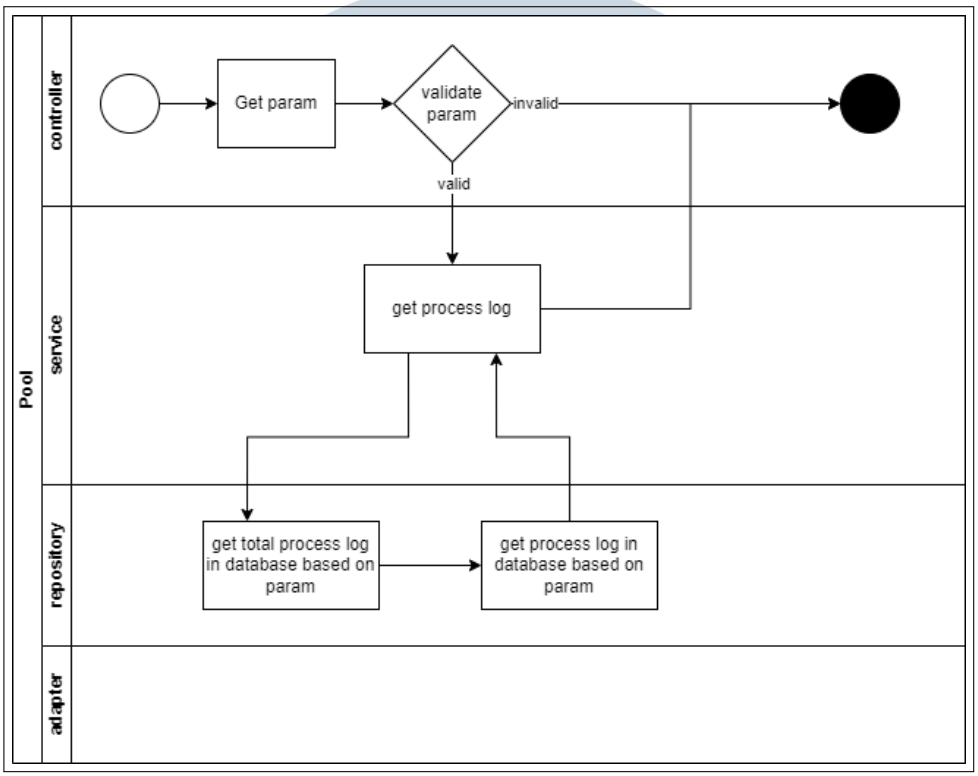
D.2 POST post-bulk-process-queue



Gambar 3.9. *Swimlane Diagram* POST get-bulk-process-queue

Gambar 3.8 menunjukkan alur dari POST get-bulk-process-queue. Proses dimulai dari sistem menerima parameter yang berisikan data seperti *filename, status* dan *vendor*. Parameter-parameter tersebut akan diteruskan sampai ke *repository layer*. Pada *repository layer* sistem akan meramu *query* sesuai dengan parameter yang ada. Data hasil *insert query* akan diteruskan ke *layer service* lalu akan dikembalikan ke *layer controller*. Setelah itu, *layer controller* akan mengirimkan ID *query process queue* ke *user* dalam bentuk respon API yang sesuai dengan format.

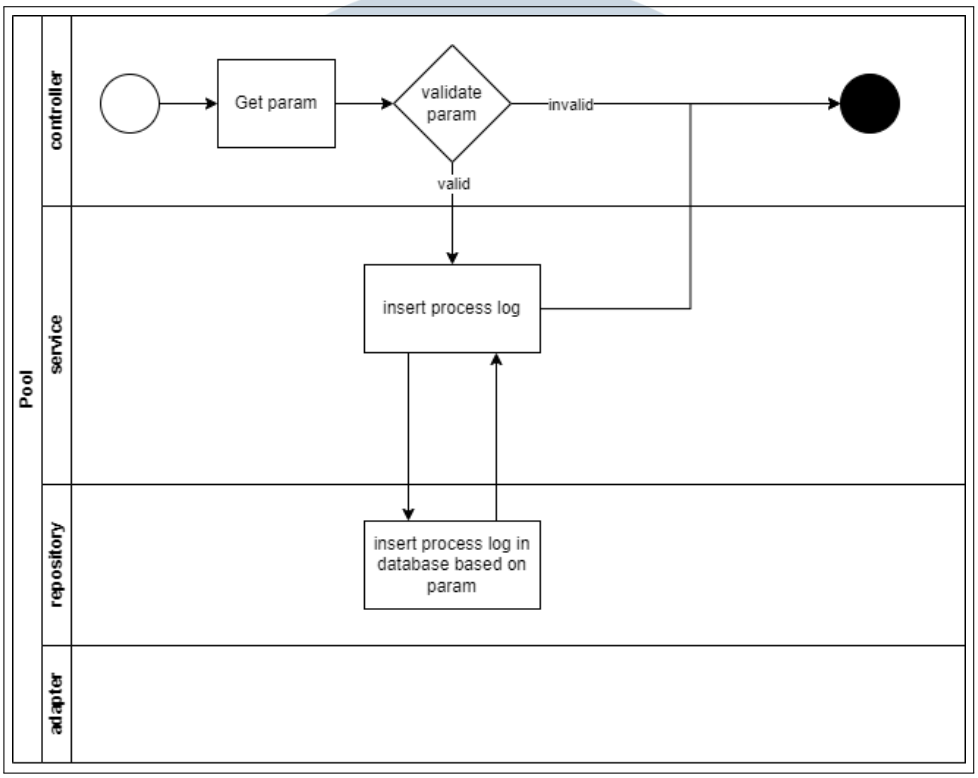
D.3 GET get-bulk-process-log



Gambar 3.10. *Swimlane Diagram* GET get-bulk-process-log

Gambar 3.10 menunjukkan alur dari GET get-bulk-process-log. Proses dimulai dari sistem menerima parameter yang berisikan data seperti *filename, status* dan *vendor*. Parameter-parameter tersebut akan diteruskan sampai ke *repository layer*. Pada *repository layer* sistem akan meramu *query* sesuai dengan parameter yang ada. Data hasil *get query* akan diteruskan ke *layer service* lalu akan dikembalikan ke *layer controller*. Setelah itu, *layer controller* akan mengirimkan data *process log* ke *user* dalam bentuk respon API yang sesuai dengan format.

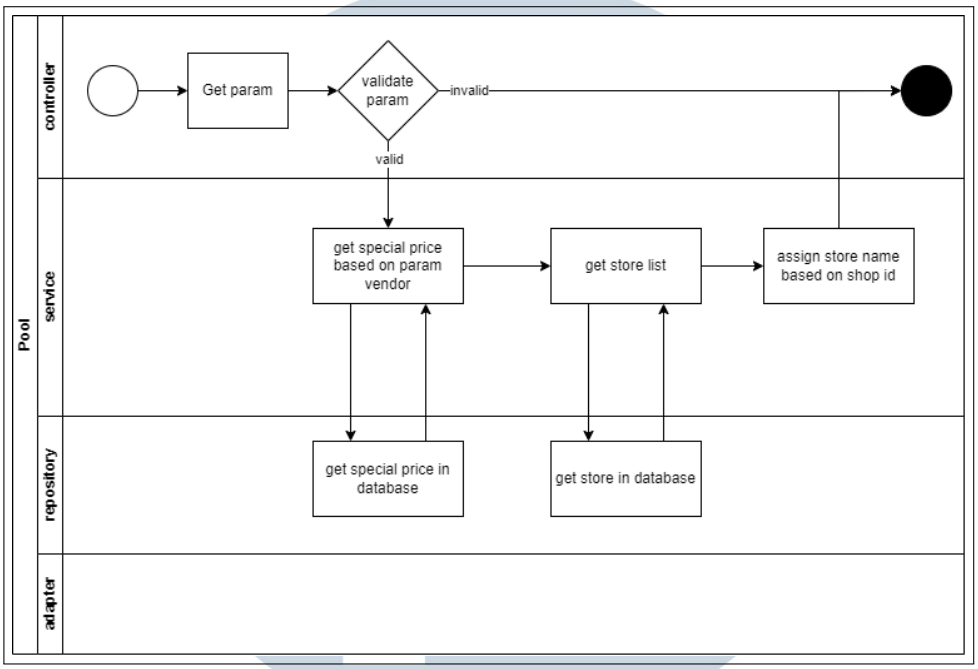
D.4 POST post-bulk-process-log



Gambar 3.11. *Swimlane Diagram* POST post-bulk-process-log

Gambar 3.11 menunjukkan alur dari POST post-bulk-process-log. Proses dimulai dari sistem menerima parameter yang berisikan data seperti ID *queue* dan isi *log*. Parameter-parameter tersebut akan diteruskan sampai ke *repository layer*. Pada *repository layer* sistem akan meramu *query* sesuai dengan parameter yang ada. Data hasil *insert query* akan diteruskan ke *layer service* lalu akan dikembalikan ke *layer controller*. Setelah itu, *layer controller* akan mengirimkan ID *query process queue* ke *user* dalam bentuk respon API yang sesuai dengan format.

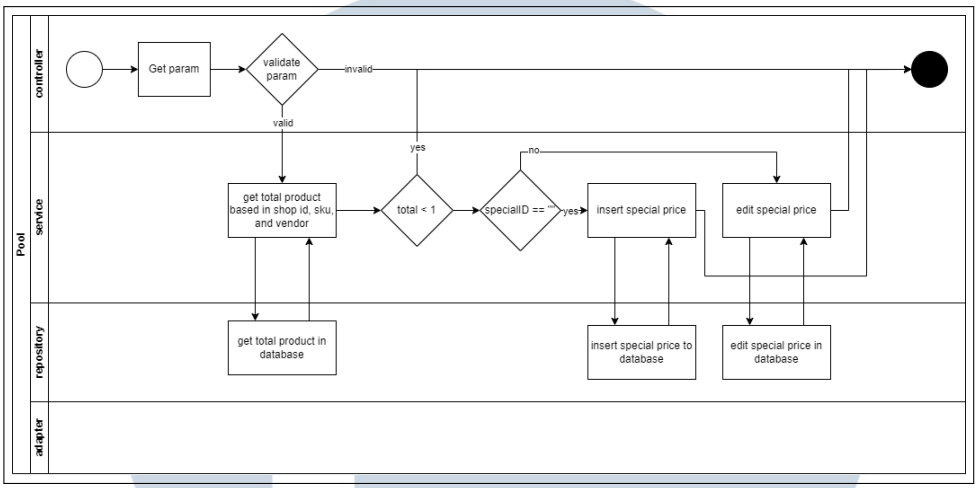
D.5 GET get-special-price-discount



Gambar 3.12. *Swimlane Diagram* GET get-special-price-discount

Gambar 3.12 menunjukkan alur dari GET get-special-price-discount. Proses dimulai dari sistem menerima parameter yang berisikan data seperti *vendor*. Selanjutnya pada *service layer*, sistem akan mengambil data *special price* dan *store list* di *database*. Lalu sistem akan melakukan *mapping* untuk menentukan nama toko. Setelah itu, *layer controller* akan mengirimkan data *special price* dari *service layer* ke *user* dalam bentuk respon API yang sesuai dengan format.

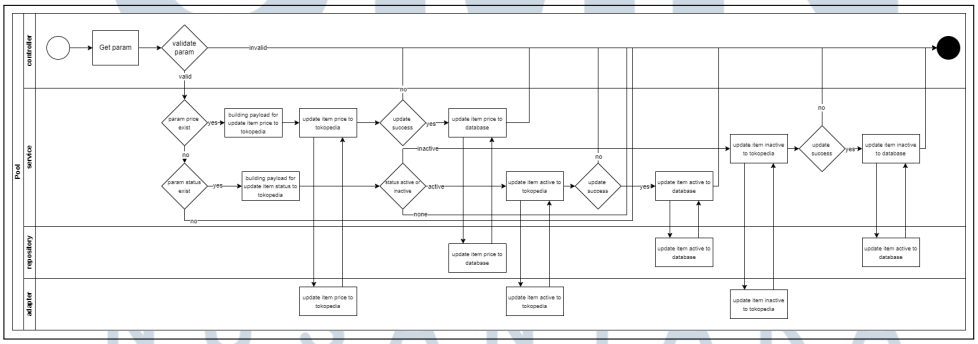
D.6 POST post-special-price-discount



Gambar 3.13. *Swimlane Diagram* POST post-special-price-discount

Gambar 3.13 menunjukkan alur dari POST post-bulk-process-log. Proses dimulai dari sistem menerima parameter yang berisikan data seperti ID *queue* dan isi *log*. Parameter-parameter tersebut akan diteruskan sampai ke *repository layer*. Pada *repository layer* sistem akan meramu *query* sesuai dengan parameter yang ada. Data hasil *insert query* akan diteruskan ke *layer service* lalu akan dikembalikan ke *layer controller*. Setelah itu, *layer controller* akan mengirimkan ID *query process queue* ke *user* dalam bentuk respon API yang sesuai dengan format.

D.7 POST tokopedia-item-update

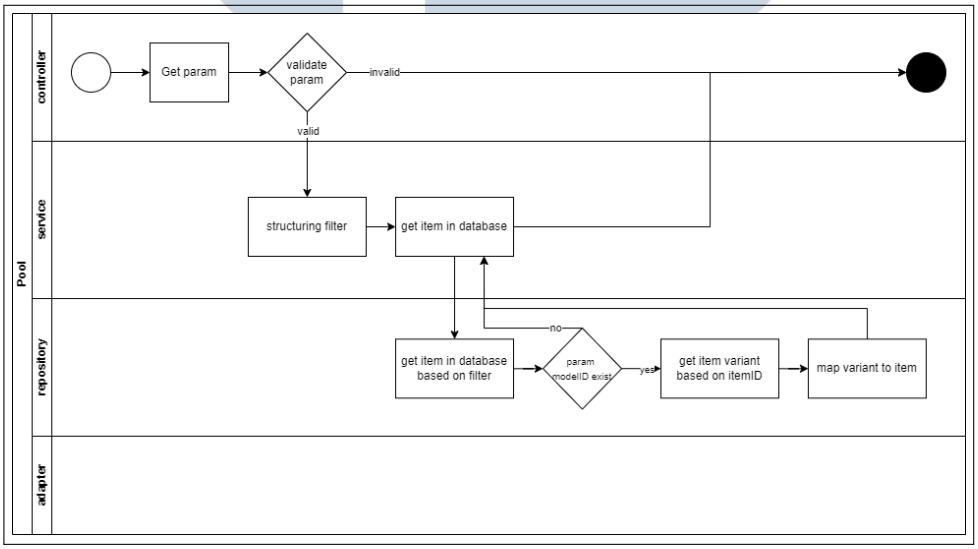


Gambar 3.14. *Swimlane Diagram* POST tokopedia-item-update

Gambar 3.14 menunjukkan alur dari POST tokopedia-item-update. Proses dimulai dari sistem menerima parameter yang berisikan data seperti status dan

harga. Selanjutnya pada *service layer*, sistem akan mengecek apakah terdapat parameter harga, jika ada maka sistem akan melakukan *update price* ke Tokopedia lalu jika *update* berhasil maka sistem akan melakukan *update price* barang tersebut ke *database*. Jika tidak ada parameter harga maka sistem akan mengecek apakah terdapat parameter status, jika ada maka sistem akan melakukan *update* status sesuai dengan status yang diinginkan. Jika status adalah *active* maka sistem akan mengirim *payload* ke *endpoint* aktivasi barang, jika status adalah *inactive* maka sistem akan mengirim *payload* ke *endpoint* deaktivasi barang. Setelah berhasil melakukan *update*, sistem akan melakukan *update price* atau status yang sesuai pada *database*. Selanjutnya respon berhasil atau gagal akan dikirim ke *layer controller* yang akan meneruskan respon ke *user* dalam bentuk respon API yang sesuai dengan format.

D.8 GET tokopedia-get-item

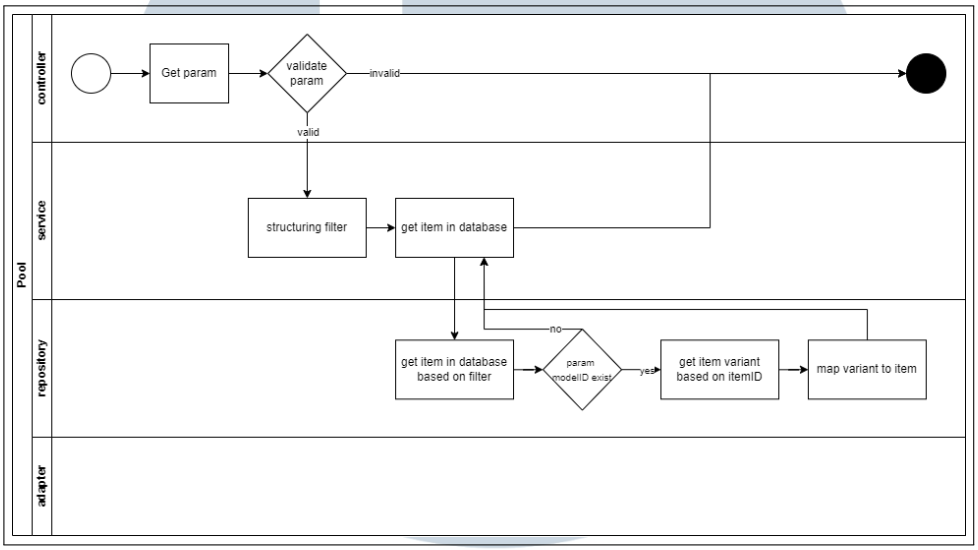


Gambar 3.15. *Swimlane Diagram* GET tokopedia-get-item

Gambar 3.19 menunjukkan alur dari GET tokopedia-get-item. Proses dimulai dari sistem menerima parameter yang berisikan data seperti modelID dan itemID. Selanjutnya pada *service layer*, sistem akan merangkai *filter* yang akan digunakan untuk mengambil data pada *database* produk tokopedia. Kemudian *filter* tersebut akan dikirim ke *repository layer*, pada layer tersebut sistem akan mengambil *item* sesuai dengan ItemID dan parameter lainnya dan jika terdapat parameter modelID maka sistem akan mengambil *item variant* sesuai dengan

modelID. Lalu sistem akan melakukan mapping properti variant ke properti *item*. Selanjutnya data *item* akan dikirim ke *layer controller* yang akan meneruskan respon ke *user* dalam bentuk respon API yang sesuai dengan format.

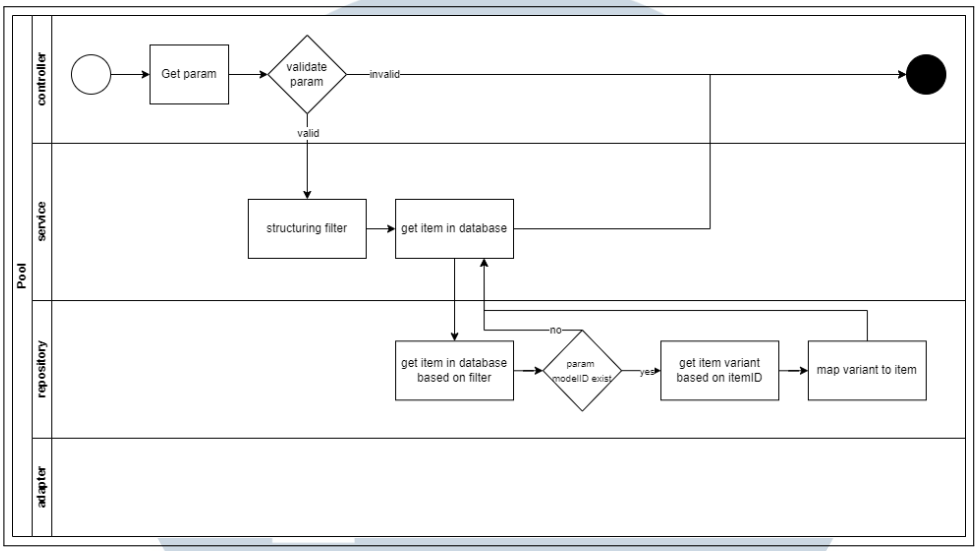
D.9 GET tokopedia-get-item-all



Gambar 3.16. *Swimlane Diagram* GET tokopedia-get-item-all

Gambar 3.18 menunjukkan alur dari GET tokopedia-get-item-all. Proses GET tokopedia-get-item-all mirip dengan proses GET tokopedia-get-item hanya berbeda pada proses pengambilan *item*. Proses get-item hanya mengambil satu *item* sedangkan get-item-all mengambil banyak *item*.

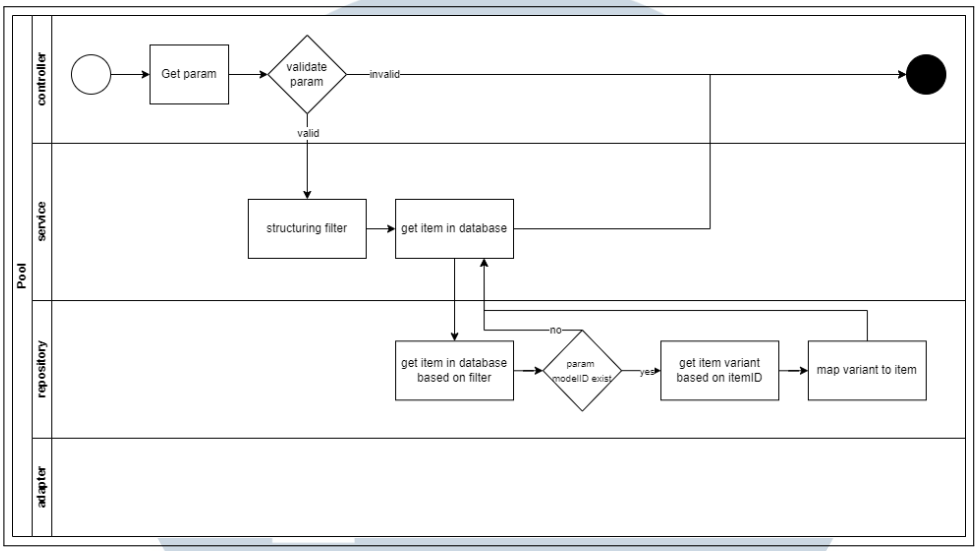
D.10 GET shopee-get-item



Gambar 3.17. *Swimlane Diagram* GET shopee-get-item

Gambar 3.19 menunjukkan alur dari GET shopee-get-item. Proses dimulai dari sistem menerima parameter yang berisikan data seperti modelID dan itemID. Selanjutnya pada *service layer*, sistem akan merangkai *filter* yang akan digunakan untuk mengambil data pada *database* produk shopee. Kemudian *filter* tersebut akan dikirim ke *repository layer*, pada layer tersebut sistem akan mengambil *item* sesuai dengan ItemID dan parameter lainnya dan jika terdapat parameter modelID maka sistem akan mengambil *item variant* sesuai dengan modelID. Lalu sistem akan melakukan mapping properti variant ke properti *item*. Selanjutnya data *item* akan dikirim ke *layer controller* yang akan meneruskan respon ke *user* dalam bentuk respon API yang sesuai dengan format.

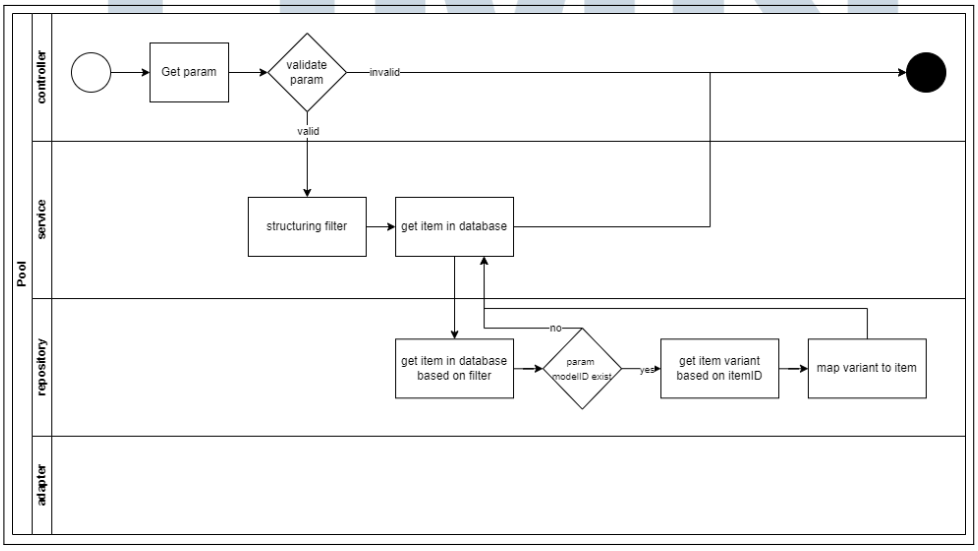
D.11 GET shopee-get-item-all



Gambar 3.18. *Swimlane Diagram* GET shopee-get-item-all

Gambar 3.18 menunjukkan alur dari GET shopee-get-item-all. Proses GET shopee-get-item-all mirip dengan proses GET shopee-get-item hanya berbeda pada proses pengambilan *item*. Proses get-item hanya mengambil satu *item* sedangkan get-item-all mengambil banyak *item*.

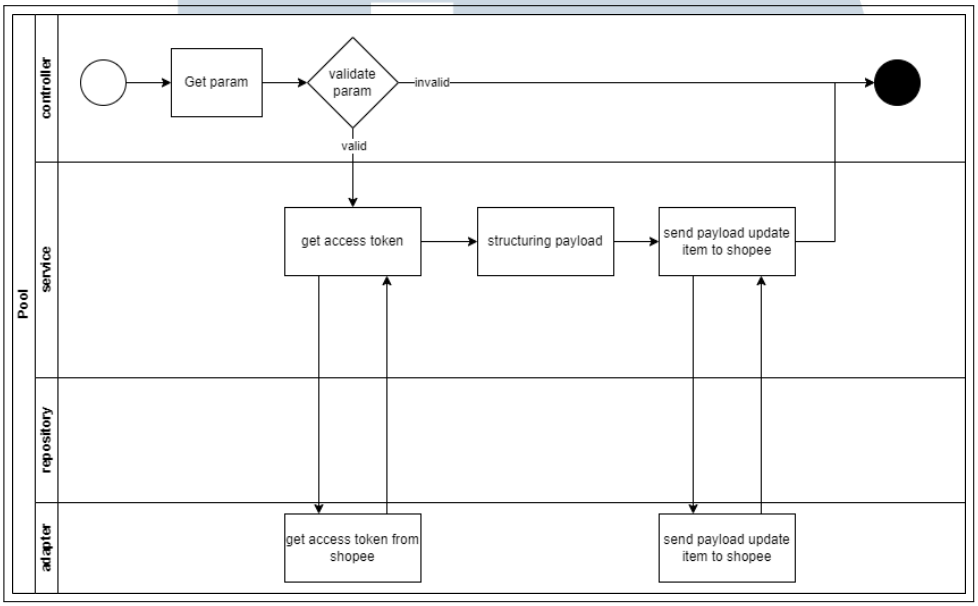
D.12 GET shopee-get-item-detail



Gambar 3.19. *Swimlane Diagram* GET shopee-get-item-detail

Gambar 3.19 menunjukkan alur dari GET shopee-get-item. Proses GET shopee-get-item-detail mirip dengan proses GET shopee-get-item namun struktur item yang dikirim dari GET shopee-get-item-detail lebih rinci daripada GET shopee-get-item.

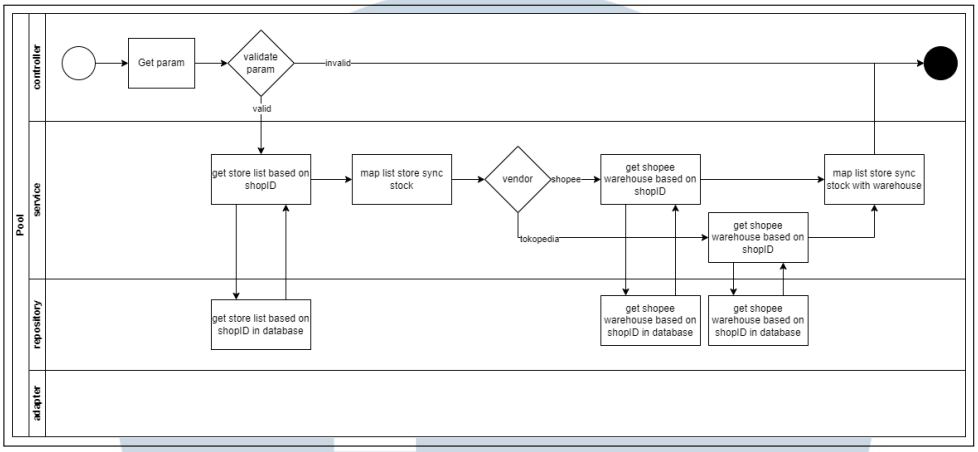
D.13 PUT shopee-edit-item-detail



Gambar 3.20. *Swimlane Diagram* PUT shopee-edit-item-detail

Gambar 3.20 menunjukkan alur dari PUT shopee-edit-item-detail. Proses dimulai dari sistem menerima parameter yang berisikan data seperti deskripsi dan dimensi barang. Selanjutnya pada *service layer*, sistem akan mengambil *access token* shopee kemudian proses dilanjutkan ke tahap pembuatan *payload update item detail*. Setelah itu sistem akan mengirim *paylaod* tersebut ke shopee. Selanjutnya respon berhasil atau tidak akan dikirim ke *layer controller* yang akan meneruskan respon ke *user* dalam bentuk respon API yang sesuai dengan format.

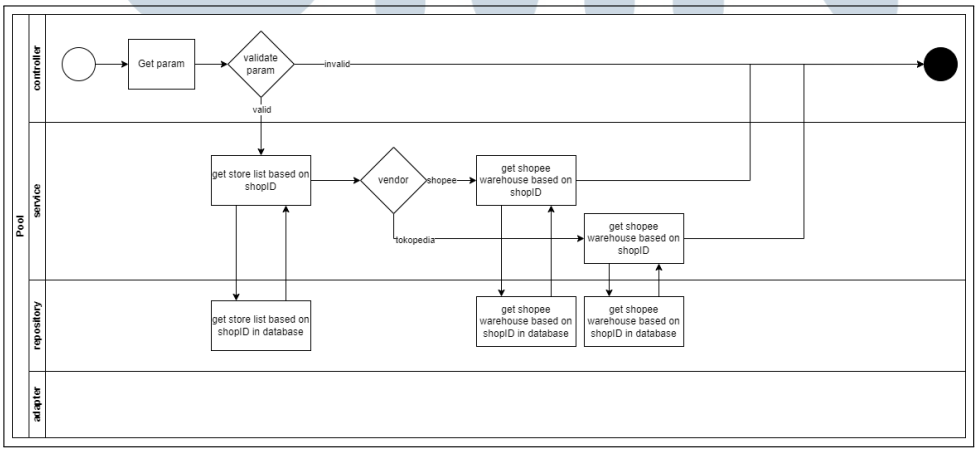
D.14 GET get-store-list



Gambar 3.21. *Swimlane Diagram* GET get-store-list

Gambar 3.21 menunjukkan alur dari GET get-store-list. Proses dimulai dari sistem menerima parameter yang berisikan data seperti nama *vendor* dan shopID. Selanjutnya pada *service layer*, sistem akan mengambil *list store* yang sesuai dengan nama *vendor* di parameter. Lalu dibuat *map* yang berisikan *store code* dengan status isSyncStock *boolean true*. Selanjutnya sesuai dengan nama *vendor*, sistem akan mengambil *list warehouse* berdasarkan shopID lalu *list warehouse* akan dicocokan *map sync store code*. Jika cocok *store* tersebut akan dipasang atribut isSyncStock. Selanjutnya data *list store* akan dikirim ke *layer controller* yang akan meneruskan respon ke *user* dalam bentuk respon API yang sesuai dengan format.

D.15 GET get-store-warehouse



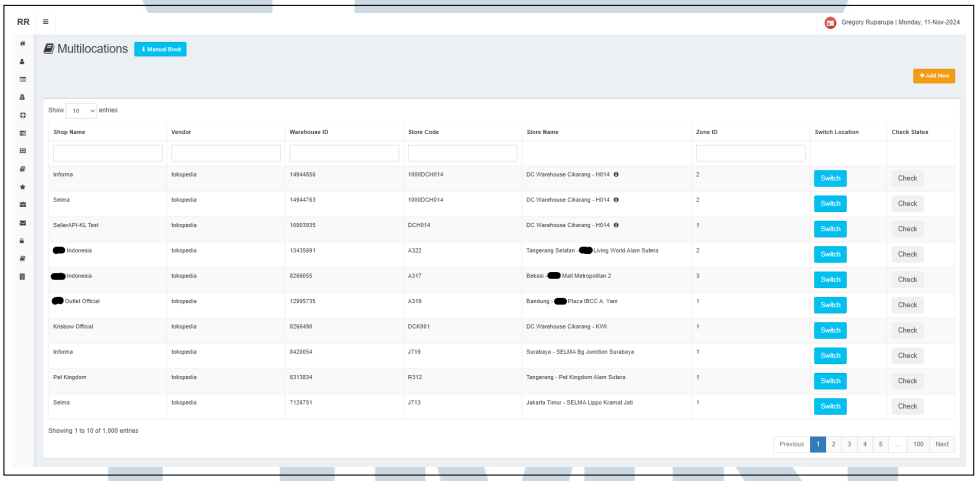
Gambar 3.22. *Swimlane Diagram* GET get-store-warehouse

Gambar 3.22 menunjukkan alur dari GET get-store-warehouse. Proses dimulai dari sistem menerima parameter yang berisikan data seperti nama *vendor* dan shopID. Selanjutnya pada *service layer*, sistem akan mengambil *list store* yang sesuai dengan shopID di parameter. Lalu sistem akan mengambil *list warehouse* berdasarkan shopID dan nama *vendor*. Selanjutnya data *warehouse* dikirim ke *layer controller* yang akan meneruskan respon ke *user* dalam bentuk respon API yang sesuai dengan format.

3.3.4 Implementasi

A. Membuat Fitur Add and Switch Multilocation

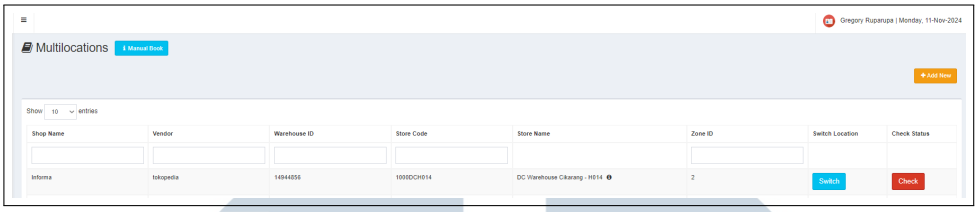
Berikut adalah implementasi dari fitur *Add and Switch Multilocation*:



Gambar 3.23. Dashboard Multilocation

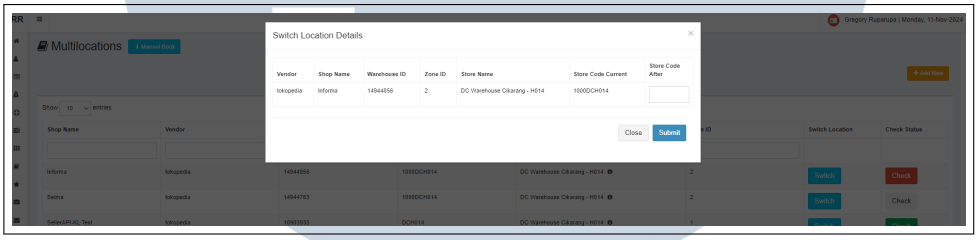
Tampilan *dashboard* Multilocation dapat dilihat pada Gambar 3.23, di dalam menu tersebut *user* dapat melihat *list multilocation* yang ada pada *shop-shop*. Terdapat informasi nama *shop*, nama *vendor*, id *warehouse*, *store code*, nama *store* atau gudang dan id zona gudang.

Gambar 3.24. Tanda Gudang Aktif

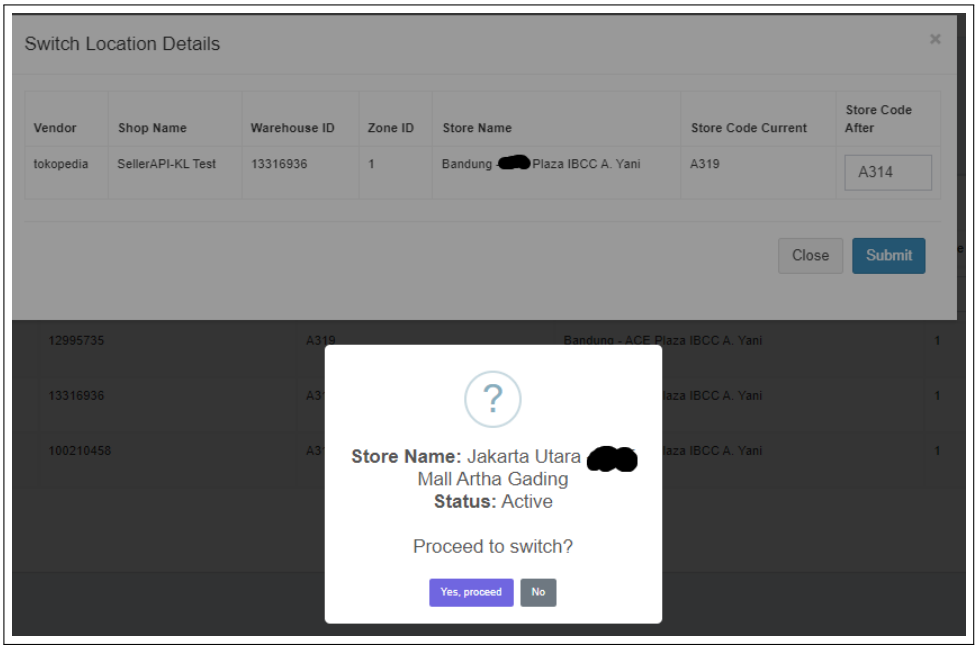


Gambar 3.25. Tanda Gudang Tidak Aktif

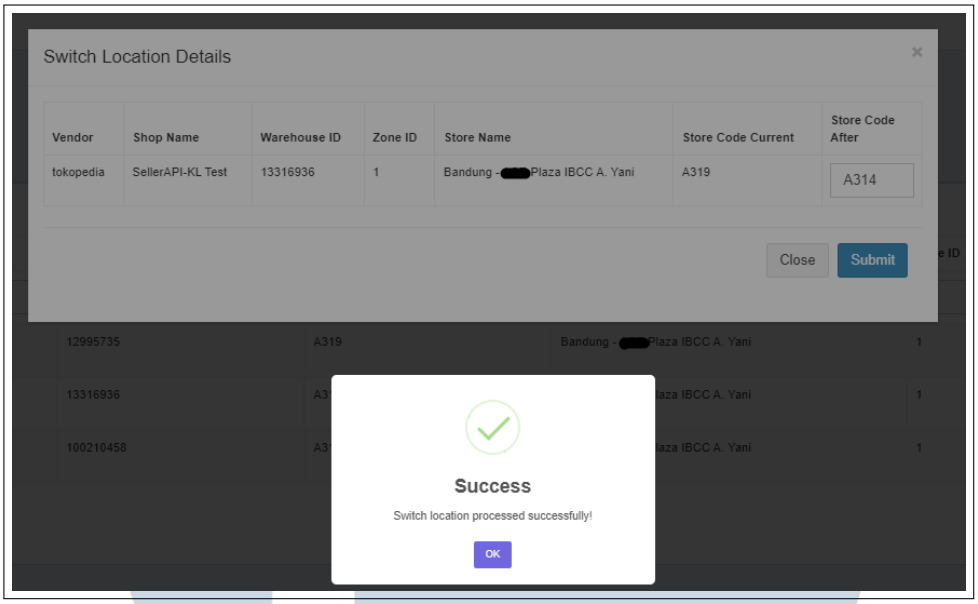
User dapat melihat status gudang yang diinginkan hanya dengan menekan tombol *check status*. Jika gudang masih aktif maka tombol akan berwarna hijau (Gambar 3.24) sedangkan akan berwarna merah jika gudang tersebut tidak aktif (Gambar 3.25). Terdapat button *switch* yang dapat digunakan untuk mengganti *store code* gudang.



Gambar 3.26. Tampilan Switch Multilocation

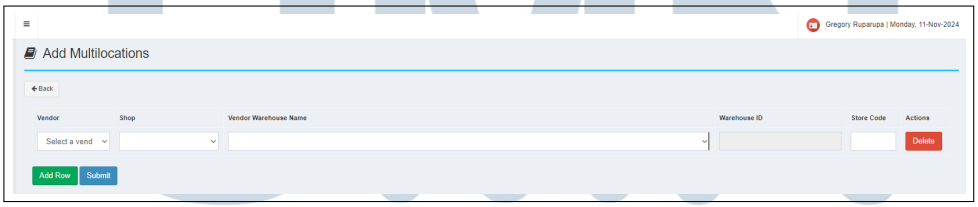


Gambar 3.27. Pop Up Konfirmasi Switch Location

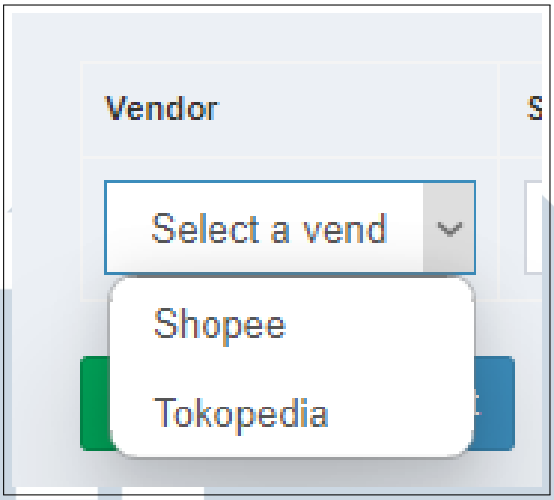


Gambar 3.28. Pop Up Status Proses Switch Location

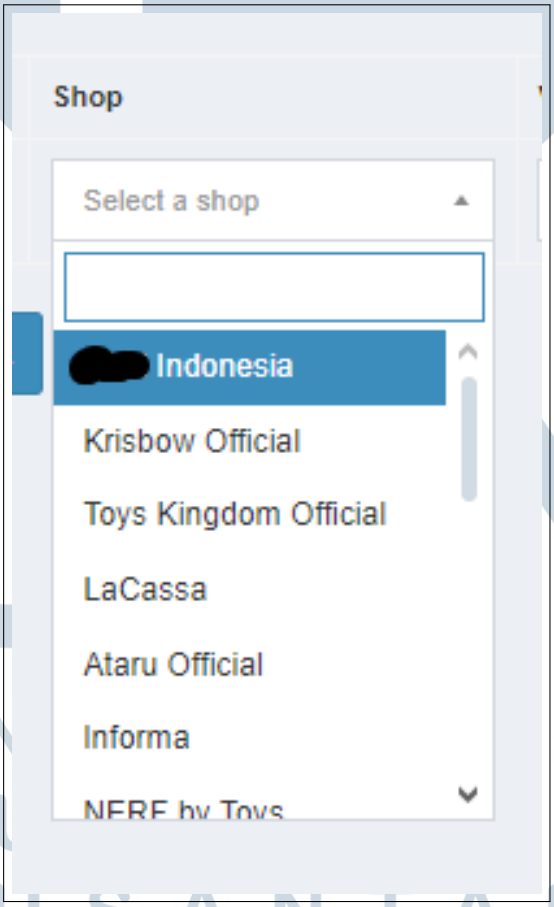
Tampilan *switch location* dapat dilihat pada Gambar 3.26, pada tampilan tersebut *user* dapat melihat detail informasi gudang dan terdapat *input field* untuk *store code* baru. Jika *user* telah memasukkan *store code* baru dan menekan tombol *submit* maka akan muncul *pop up* verifikasi yang berisikan pertanyaan konfirmasi, nama *store* dan status *store* yang baru (Gambar 3.27). Setelah *user* menekan tombol "*Yes, proceed*" maka akan muncul *pop up* yang berisikan status *switching* (Gambar 3.28).



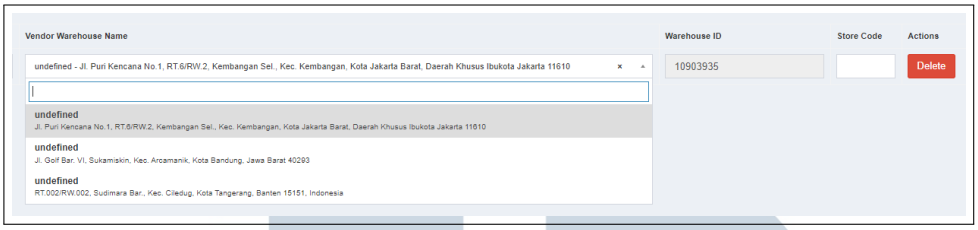
Gambar 3.29. Menu Add Multilocation



Gambar 3.30. *Dropdown Vendor*

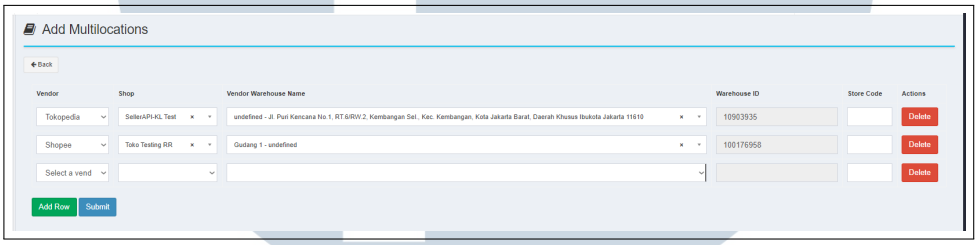


Gambar 3.31. *Dropdown Shop*

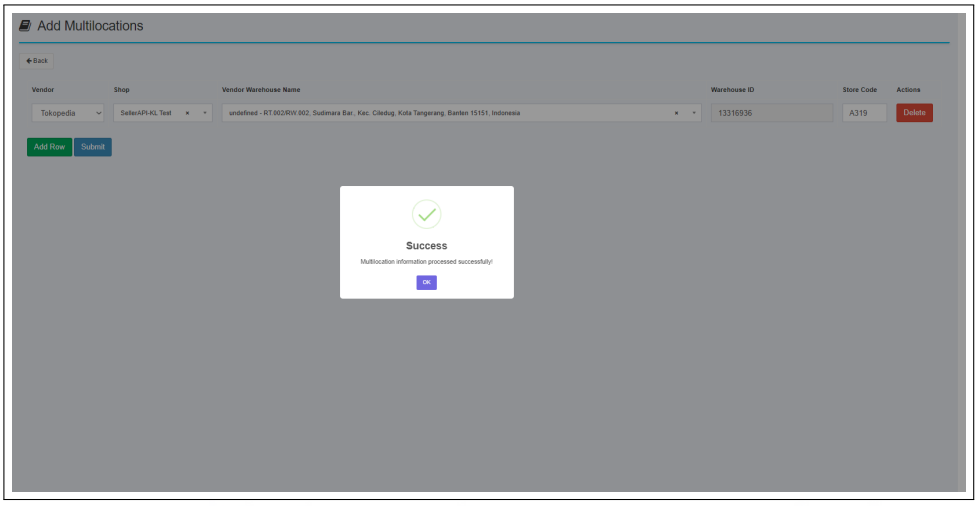


Gambar 3.32. *Dropdown Available Store*

Jika *user* menekan tombol ”*Add New*” maka tampilan akan berpindah ke menu *Add Multilocation* seperti pada Gambar 3.29. Pada tampilan tersebut terdapat pilihan *dropdown* untuk memilih *vendor* (Gambar 3.30), *shop* (Gambar 3.31) dan gudang yang tersedia (Gambar 3.32).



Gambar 3.33. *Add Multiple Locations*

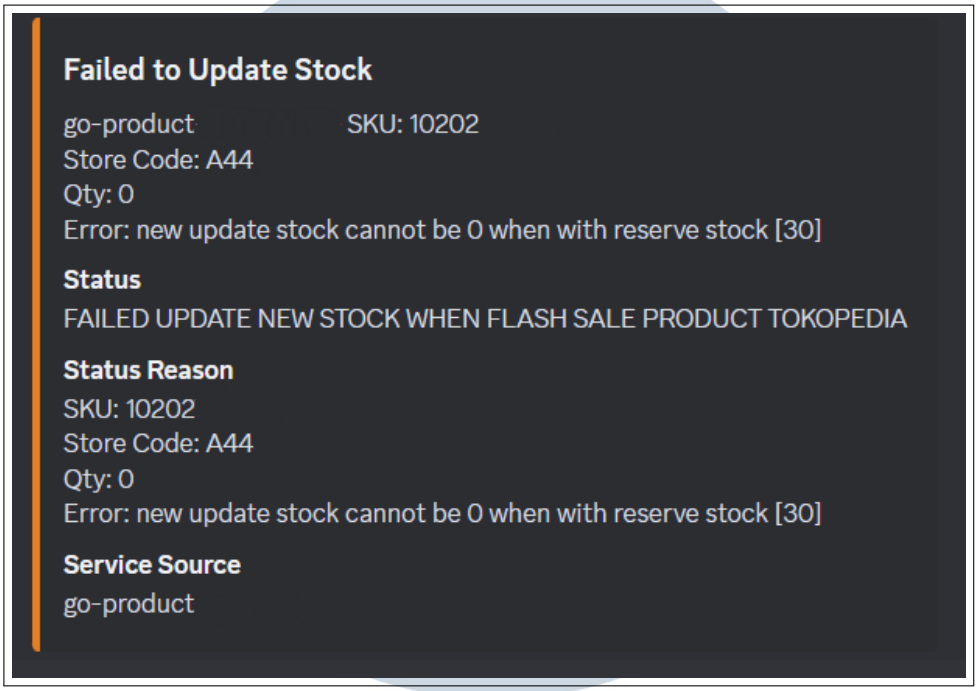


Gambar 3.34. *Pop Up* Sukses *Add Multilocation*

User dapat menambahkan lebih dari satu lokasi baru secara bersamaan seperti pada Gambar 3.33. Setelah *user* menekan tombol ”*submit*” maka akan muncul *pop up* status proses *add multilocation* (Gambar 3.34)

B. Membuat Fitur Notifikasi ke Discord

Berikut adalah implementasi fitur notifikasi ke Discord

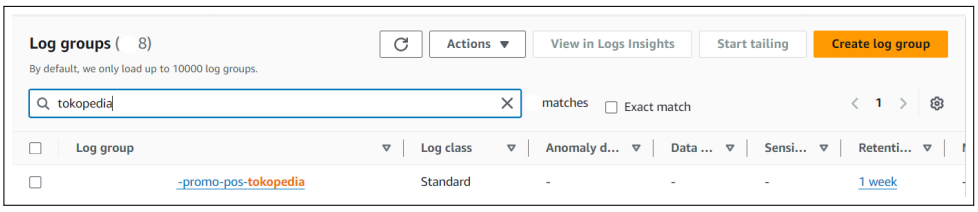


Gambar 3.35. Notifikasi Error Update Stock Saat Flash Sale

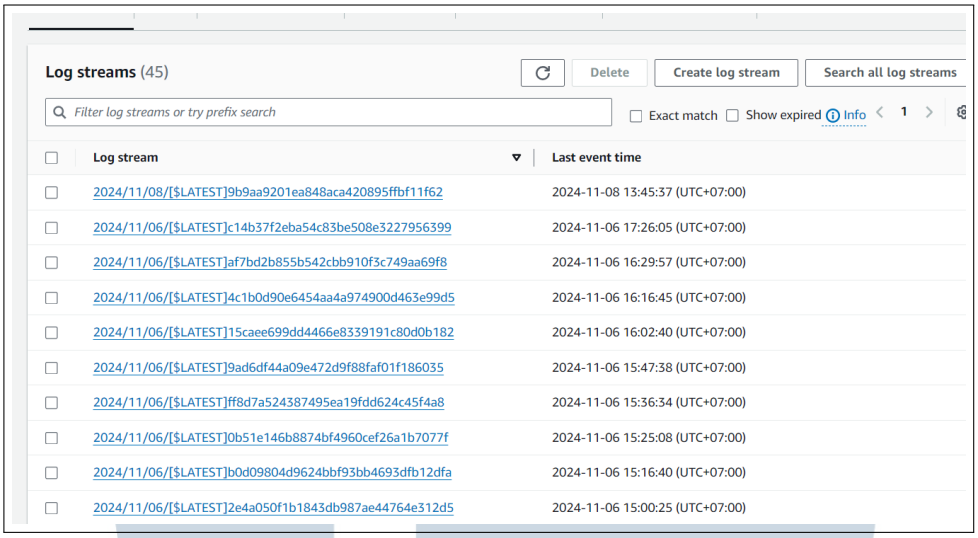
Contoh isi notifikasi gagal *update stock* karena promo sepihak dari *vendor* Tokopedia dapat dilihat pada Gambar 3.35. Isi notifikasi meliputi informasi mengenai SKU pada *store code* tertentu, kuantitas stok yang akan diubah (Qty) dan pesan *error*.

3.3.5 Membuat *Batch Auto Retry Promo Pos Tokopedia*

Berikut adalah implementasi dari fitur *Batch Auto Retry Promo Pos Tokopedia*:

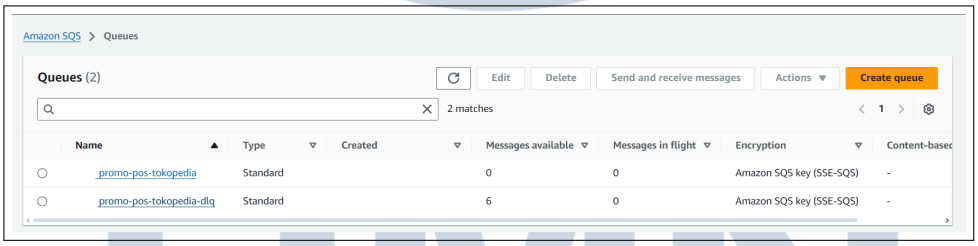


Gambar 3.36. *Log Promo POS Tokopedia*



Gambar 3.37. Isi *Log Promo POS Tokopedia*

Pada *CloudWatch*, pengguna dapat melihat berbagai *log* dari *Lambda* yang ada, sebagai contoh pengguna dapat melihat *log* dari *Lambda* yang digunakan untuk *Batch Promo* POS Tokopedia seperti pada Gambar 3.36 dan isi dari *log* tersebut seperti pada Gambar 3.37.



Gambar 3.38. SQS *Batch Promo* POS Tokopedia

Pengguna juga dapat melihat SQS dan DLQ pada menu SQS AWS. Pada menu tersebut pengguna dapat melihat status dan informasi tentang SQS yang digunakan dalam sistem *batch auto retry promo* pos Tokopedia serta jumlah *message* yang sedang diproses dan *message* yang masuk ke DLQ.

3.4 Kendala dan Solusi yang Ditemukan

Kendala-kendala yang ditemukan selama pelaksanaan magang di PT Omni Digitama Internusa, antara lain:

1. Minimnya dokumentasi pada repositori sebuah layanan yang menyebabkan butuhnya waktu ekstra untuk memahami maksud dari kode.

Upaya-upaya yang telah dilakukan dalam mengatasi kendala-kendala yang ditemukan adalah:

1. Mempelajari kode dan cara menjalankan layanan di komputer lokal secara mandiri dan dengan meminta bantu rekan kerja yang sudah berpengalaman.